Festschrift

LNAI 2605

Dieter Hutter
Werner Stephan (Eds.)

Mechanizing Mathematical Reasoning

Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday

Springer

Lecture Notes in Artificial Intelligence 2605

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Dieter Hutter Werner Stephan (Eds.)

Mechanizing Mathematical Reasoning

Essays in Honor of Jörg H. Siekmann
on the Occasion of His 60th Birthday

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Dieter Hutter
Werner Stephan
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
E-mail:{hutter, stephan}@dfk.de

Library of Congress Control Number: 2005921072

CR Subject Classification (1998): I.2.3, I.2, F.4.1, D.2.4

ISSN 0302-9743
ISBN 3-540-25051-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11399056 06/3142 5 4 3 2 1 0

Preface

These essays pay tribute to Jörg H. Siekmann on his 60th birthday, which gave reason to his friends, former and recent students, and colleagues to celebrate his career as a scientist who has contributed to many diverse fields. These are covered by the wide range of essays that make up this volume. The way the scientific contributions were grouped reflects the development of Jörg's research interests over the many years of his professional life. Moreover, many of them are directly influenced by his own work.

Part 1 starts with contributions on basic deduction techniques, from decision theories, rewriting, constraint solving and unification to entire systems for automated theorem proving. When Jörg started his academic career, the area of automated reasoning and also Jörg's early work were almost completely dominated by these disciplines. Today, after two or more decades of substantial improvements, techniques like those developed in his groups and those discussed in this volume provide the basic machinery for application-specific, more comprehensive support systems.

Shortly before and during his time in Saarbrücken Jörg became more and more involved in several forward-looking and application-driven developments that use "automated deduction inside." Part 2 focuses on various application scenarios like knowledge representation and retrieval, natural language processing and e-learning.

As opposed to applications in education and Web technology, for example, the use of deductive techniques in formal software development goes back to the 1970s. Jörg was in contact with the developers of these early approaches primarily through his collaboration with Peter Raulefs in Karlsruhe and Kaiserslautern. However, it was through the VSE project that he became active in this rapidly growing research area, which became an important role for his group at the DFKI.

Besides the more technical research issues mirrored in most of the contributions of this volume, Jörg was always concerned about the fundamental problem of explaining human intellectual behavior by means of computer models. In particular the study of emergent behavior and self-organization attracted him beyond the use of agent architectures for theorem proving or, in the other direction, the application of deductive techniques for agent planning. Part 4 presents work in the spirit of Jörg's research group on multiagent systems.

Compiling this volume has been a great pleasure. We are grateful that everyone who we asked to contribute responded positively, expressing their desire to honor Jörg H. Siekmann. We thank the authors for their patience during the long period between the first ideas on preparing this volume and its later publication. Special thanks goes to Jörg for providing us with a preliminary draft of his autobiography which illuminates various facets of his eventful life. The draft turned out to be an invaluable source when writing the appraisal of his career.

Jörg has had a major impact on our lives. I, Dieter, first met him in 1979 when I was an undergraduate student at the University of Karlsruhe, obliged to give a proseminar on Lenat's AM system. His inspiring enthusiasm and energy for this kind of AI system stirred my growing interest in automated theorem proving in general and guiding such provers in particular. Not surprisingly, after finishing my MSc thesis I registered as a PhD student of Christoph Walther and Peter Deussen, joining the development of the inductive theorem prover INKA. Years later, Jörg offered me a research position in Saarbrücken to incorporate proof guiding techniques into VSE; I accepted willingly.

At the time when Jörg moved to Karlsruhe I, Werner, was working in Wolfgang Menzel's group, which together with Deussen's group formed the institute of theoretical computer science. Despite countless controversial discussions on AI-related topics and developments in the common academic context, as well as ongoing changes in the German society of the early 1980s, we shared many basic views and soon became friends. But also, in a more restricted sense, my academic work was influenced by Jörg's often more than enthusiastic way of standing up for AI and, in particular, automated deduction. Being interested at that time in the semantics of programming languages and logics, the discussions with Jörg's group sharpened my view that computer-assisted deductive systems (together with "model based" analysis techniques) "animate" formal development techniques thereby enabling their application in software engineering.

Since then, our work at Jörg's research department at the DFKI has been guided by the search for a close integration of formal methodology and deductive support, bringing together two communities that, still today, are often far from working hand in hand. Throughout all the years, Jörg has been a constant source of inspiration, friendship and humor. In presenting Jörg with this book we hope that he will enjoy reading it, and since his work is by no means finished, we hope to return the favor by reading more of the essays he has promised to write in the coming years.

Dieter Hutter
Werner Stephan

Table of Contents

Logic and Deduction

Applications of Logics

Formal Methods and Security

Agents and Planning

A Portrait of a Scientist: Logic, AI and Politics

Dieter Hutter and Werner Stephan*

German Research Centre for Artificial Intelligence,
D-66123 Saarbrücken, Germany

1 Bückeburg

Mitnehmen kann man das Vaterland
An den Sohlen und an den Füßen
Das halbe Fürstentum Bückeburg
Blieb mir an den Stiefeln kleben.
So lehmichte Wege habe ich wohl
Noch nie gesehen im Leben.

Heinrich Heine: Ein Wintermärchen, 1844

At the time Alan Turing was engaged in deciphering the code of the Enigma in Bletchley Park and Konrad Zuse applied his patent for the first electronic computer called "Rechenvorrichtung" in Berlin, Jörg was born into the rural capital of the smallest Fürstentum of Germany, called Schaumburg-Lippe, a name even well educated Germans have probably never heard of. Jörg grew up as the oldest son of a family whose male providers had been joiners and curlers for centuries. There has never been a question that one day he would inherit the small family owned curler and joiners workshop.

But things turned out otherwise: Schaumburg-Lippe never became an independent Land again and the once respectable Siekmann family of joiners, curlers and church leaders was on the decline: mass furniture production became a highly capital intensive, fully automated business, where a certain Swedish company set the pace. In that process, almost all of the family-owned small and medium sized woodworking companies vanished and the once proud Schaumburg Lippesche Handwerkskammer dating back far into medieval times became obsolete.

But Jörg did not quite fit particularly well anyway: when he could not decide whether he wanted to be an artist or a scientist – an idea so inconceivable that his father threatened to cut off all his family ties – he took his juvenile poems and drawings to a family friend who looked at his paintings and poems with a stern expression and suggested: "Son, you better learn the trade of your fathers!"

So, Jörg became a joiner's apprentice in the nearby village of two hundred souls called Scheie, and after three and a half years he passed the traditional examinations and practical tests with some distinction: being now a well recognized member of the German chamber of handicraft entitled to call himself a Tischlergeselle.

Two years in the army, a further apprenticeship as a metalworker and welder finally qualified him to enter the Technikum Rosenheim, an engineering school

* We like to thank Jörg for freely using his "Autobiographic Notes" from which some material and most of the personal information is drawn and quoted without explicit indication.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 1–13, 2005.

for woodwork and furniture production, which bestowed the title of grad.Ing. Rosenheim on him. A small research and technology company hired him right away and they invented the "R-value", a ventilation measure, apparently still in use today, which classifies the ventilation capacity of (wooden) windows.

Being bored with the "science" of window manufacturing, still unwilling to accept the position as the head of his father's company, which did not flourish all too well anyway, he decided it was time for a fundamental change and a new start.

University life was walled up in those days by the Abitur, the German equivalent of the anglo-saxon A-levels, a watershed in the conservative Germany of Adenauer's days that divided those who have from those who have not. His first marriage (of which there were several more to come) broke up and he started again as a schoolboy in evening classes and eventually he was admitted as a student at the Braunschweig Kolleg, a prestigious German adult education centre. Three years later at the age of almost thirty all doors were finally open: he had his Abitur!

But life never seemed to evolve in a straight line with Jörg; politics dominated his life: this was the late sixties, the peak of the student revolt in Paris and Germany. Benno Ohnesorg, his fellow student from the Braunschweig Kolleg, was shot by a policeman during a student demonstration, Greece was controlled by the military dictatorship of Papadopoulos and his colonels. This is a period in Jörg's life he wisely concealed in his curriculum vitae and application letters to the university in a country that not only exported terms like Kindergarten and Eigenwert into the English language, but also the word Berufsverbote.

2 Göttingen and Essex

Zu Göttingen blüht die Wissenschaft,
Doch bringt sie keine Früchte.
Ich kam dort durch in stockfinstrer Nacht,
Sah nirgendwo ein Lichte.

Heinrich Heine: Der Tannhäuser, 1836

In nineteen seventy the dream of the "little joiner's boy" became true: he enrolled as a student in mathematics and physics at Göttingen University and he was accepted as a member and soon elected a senior tutor of the Akademische Burse.

The introduction to mathematics by Grauert and Brieskorn, Scheibe's lectures on time and relativity and the intellectual and political debates of the Akademische Burse were formative years. But it was logic and its introductory courses by Patzig and others that captured his imagination: apparently there are deeper and more eternal truths behind the appearance of everyday academic life.

The Vordiplom in mathematics and physics, evening classes in the English Language Lab, a few months at the Sound and Vibration research institute in Southhampton (where they implemented one of the fastest Fourier-Transformations of its time), and finally a one-year grant from the DAAD prepared for a master course in computer science in England at Essex University. Science and politics again: the final M.Sc. degree with distinction and the admittance

to Oxford as a PhD student of Dana Scott but also a course he taught at the student union on Rosa Luxemburg and Mandel's economic theory.

In the end the public lectures on artificial intelligence by Terry Winograd, Carl Hewitt, Roger Schank and Yorick Wilks at Essex sparked a new flame that should now last for a lifetime: if machines can think and we can talk to them – these were the years of Terry Winograd's SHRDLU and Nils Nilsson's SHAKEY – then surely this was a much greater scientific challenge than all of mathematics and physics taken together, and certainly on par with some of the grand problems in logic related to the fundamental barriers of human and machine thinking.

So when Pat Hayes joined the staff of Essex University and accepted him as his PhD student, all future plans with Oxford and Germany were abandoned. The excitement with the new subject was fuelled by the staff at Essex: Richard Bornat, Mike Brady, Jim Doran, Pat Hayes, Bernard Sufrin, Yorick Wilks and a constant stream of visiting scientists from Edinburgh, Sussex and also from America provided much of the early excitement for this new subject.

His thesis "Unification and Matching Problems" on unification theory for combinations of associativity, commutativity and idempotency introduced the notion of a unification hierarchy based on the cardinality of the set of most general unifiers. With his collaboraters Mike Livesey and Peter Szabo, Jörg elaborated a classification of this hierarchy, which now carries his name. The early work of Gordon Plotkin, the thesis of Gerard Huet, and his work with Peter Szabo and others finally established unification theory as a subject of its own, with annual workshops and subsections at AI, automated reasoning and mathematics conferences.

3 Karlsruhe

Eines Nachmittags ging Markgraf Karl Wilhelm im Hardtwald auf die Jagd, um seinen Aerger zu vergessen. Er traf einen Hirsch, verfolgte das Tier und ließ dabei sein Gefolge weit hinter sich. Vom langen Ritt ermüdet, setzte er sich schließlich auf einen Eichenstumpf mitten im Wald. Bald war er eingeschlafen. Erst nach Stunden fanden seine Jagdgenossen ihren schlafenden Herrn. Man weckte ihn, und als er sich umschaute, gefiel ihm der Ort so gut, dass er sagte: "In meinem Leben habe ich noch niemals besser geschlafen als hier. An diesem Platz möchte ich immer wohnen. 'Karls Ruhe' soll er künftig heißen. Und über diesem Baumstumpf will ich eine Kirche errichten, in der ich einstens zur ewigen Ruhe gebettet werde."

Historical saga of the foundation of Karlsruhe

In 1976 Jörg moved to Karlsruhe when AI slowly started to gain ground in Germany. The year before, the first informal German meeting on artificial intelligence was organised and a year later it was accepted formally as a working group of the GI, the German computer science society. Jörg was now an assistant and soon an associate (Hochschulassistent) in the institute of Peter Deussen at the computer science department in Karlsruhe.

His research area continued to be unification theory working in close collaboration with his friend Peter Szabo, but also and more importantly – at least in Jörg's values – the beginning of the automated theorem proving system with

the tongue-twister name Markgraph Karl Refutation Procedure (MKRP). He convinced Germany's funding agencies that he could build a theorem proving system that would not only outperform the strongest American systems by far, but establish a new paradigm of less search and more (mathematical) knowledge for theorem proving. He claimed that Deussen's book "Halbgruppen und Automaten" would be the first text book completely generated in natural language by a machine – a promise that turned out to keep him busy not only for the anticipated decade but obviously till the end of his active life.

We had the days of Carl Hewitt's PLANNER, the declarative versus procedural debate and Pat Hayes paper "An arraignment of theorem proving or a logician's folly". But the field of automated theorem proving was not particularly influenced by these debates and still dominated by Alan Robinson's resolution calculus. Its few and simple inference rules entrapped many researchers to believe that developing a successful general purpose strategy for theorem proving would be only a matter of time. Under the influence of the Essex debates and Pat Hayes' way of thinking, the new MKRP-system was supposed to be the first knowledge based theorem proving system to lead out of the trap of the merely search based approaches of the day. Bob Kowalski's connection graph seemed to be a good starting point for the new MKRP-system because of its immediate access to available resolution steps, and soon innumerable papers about connection-graph based theorem proving in general and all sorts of refinements in particular poured out of Jörg's group. The system developed well at first and soon it exhausted the computational resources of the computing faculty. Every Wednesday evening the faculties' single computer (occupying more than half of the basement of the faculties' building) was rebooted in a single-user mode for the sole reason of accommodating Germany's best theorem proving system within its four megabyte of virtual memory; and every Wednesday evening, Jörg's group reassembled in front of a VT100 terminal observing and soothsaying MKRP's behaviour on the latest examples of the deduction community.

The race was stiff with two major horses at that time: his friend Larry Wos with his much smaller team and their parsimonious but extremely well-engineered system OTTER versus the big elephant MKRP. Larry would call – usually very early in the morning – mocking a German accent: "Hey can you do zis, ve have just solved it" and then the group had a few days at most to prove a new challenge theorem from a given set of axioms. The sooner the answer "We have just done it as well, Sir" the better – so the day had twenty four hours after such a phone call to analyse the new proof and adjust the settings of the various refinement strategies such that MKRP would also find the proof[1]. The next challenge's twist was then to spot weaknesses in the opponents system, design a hard problem whose solution relied on a special technique within the weak spot of the competitor, solve it at leisure – and send it in reverse right over the Atlantic waiting for their phone call.

[1] You can always solve a difficult mathematical problem when another system has already succeeded given enough time: just analyse, set the parameters right, analyse again, add a new procedure, analyse again, etc.

While this went on for many years, the pendulum for the first prize sometimes swung to this side of the Atlantic and then back to the other. Both systems improved considerably – but none of the promised breakthroughs was forthcoming. Deussen's book was still waiting to be automatically generated.

The MKRP-effort showed that indeed you can build a knowledge based theorem proving system which prunes the search space by several orders of magnitude – but the traditional search based systems performed all in all just as well. As Larry Wos pointed out in a seminal debate at one of the CADE conferences: "We now have the ultimate system Ψ that proves a theorem without any search: it uses its efficient and knowledgeable supervisor OTTER to find a proof and then proceeds by using this knowledge to guide Ψ right through the search space". "MKRP was unfortunately still wrapped too much in the intellectual time warp of the sixties" as Jörg would comment on these developments later.

At that time, research in AI and on deduction in particular was not a mainstream business in Germany. There were yearly informal meetings on AI, until in 1981 Jörg initiated the annual German workshops on Artificial Intelligence (GWAI) with proper proceedings published by Springer. These annual meetings at the Hölterhoff Stiftung in Bad Honnef near Bonn stimulated the early excitement about AI in Germany and much of the proud and more often than not the over-important sense of self, called WIR-GEFÜHL in German, emanated from these – sometimes hilarious – meetings. In March 1982, Jörg and Wolfgang Bibel started the first German summer school on AI in Teisendorf. With more than 100 participants, it was a big success not only because Jörg became acquainted with his later wife, but also because the lecturers succeeded in transmitting their enthusiasm about AI to the convened young researchers always looking for a PhD thesis.

Politically the late 70's saw the growth of the German peace movement from a small circle of concerned scientists and peace activists into a mass movement: the planned deployment of Pershing missiles close to the eastern boarder and iron curtain reduced the effective early warning time from several hours down to a few minutes and the Soviet Union responded with the threat of an automatic launching policy – which fortunately was never fully implemented by either side. Several false alarms – some up to the highest threat level – were computed by the huge American early warning system and when these facts became public, several professors of jurisdiction and computer science, including Jörg, opened a law case against the German government at the Federal Court in Karlsruhe: some courageous American senators provided classified material for the German computer society of concerned scientists FIFF, of which Jörg was one of the founders. He made the material public, wrote several papers and a journal article with Karl Bläsius. He must have given a few hundred public talks, television interviews and speeches to the peace movement all over Germany and experienced for the first time the difference between giving a seminar talk and being a speaker in front of ten thousand people.

Politics would meet AI again when Peter Raulefs, Jörg and Graham Wrightson organized the International Joint Conference on AI (IJCAI) in 1983 at Karl-

sruhe: with more than two thousand participants it was the first major event of this size in Germany and widely covered by the media – not least because the accompanying industrial exhibition proudly displayed an empty Martin Marietta (Pershing) booth.

4 Kaiserslautern

P.T. aus Arizona
von dem Stamme der Apachen
lebte ziemlich gut in K-town, Germany.
War GI und bei der Army,
na, und Sehnsucht nach den Staaten
hatte P.T., der Apache, eigentlich nie.
Nur im Herbst, wenn Vögel schrien,
über K-town südwärts zogen,
sagte P.T. manchmal leise zu sich "Uff".
Und dann trank er sehr viel Bourbon,
stieg in seinen alten Chrysler
und fuhr rüber nach Karlsruhe in den Puff.
P.T. P.T. Das hat dem P.T. gutgetan ...

Franz-Josef Degenhardt: P.T. Arizona, 1968

In 1980 the department of computer science of the University in Kaiserslautern advertised the first professorship for AI in Germany and after the usual tiresome medival "rituals", Jörg was offered the job and in 1983 he moved from Karlsruhe to Kaiserslautern with his newly wedded wife and his coltish dog called Minsky.

To us, the next generation of scientists, who found AI already an established subject when we were students, it is probably Jörg's lecture series 'Introduction to Artificial Intelligence' that is most vivid in memory. By the mid eighties the field was thriving and banging at the doors of the scientific establishment, but it was still provocative in its general claims regarding the nature of human versus machine thinking.

The lecture at its peak drew sometimes more than five hundred students from all over Germany and many other European countries to Kaiserslautern, with students occupying the floor, the windows – wherever there was additional space – completely electrified by the subject and the atmosphere generated by this strange and witty missionary of a futuristic technology[2] with his hand-crafted slides decorated with flowers in the style of the sixties.

Jörg appeared on television, newspapers and radio shows: the AI hype had finally infected Germany as well and the bearded messiah with his dog Minsky became a familiar sight[3]. The 1984 paper on the subject of AI and its future invited by the OECD, has been printed and reprinted many times and was

[2] The lecture of the 80's was actually filmed and made publicly available as videotapes. His AI-lecture today, more mature and sober now (and available on the web), uses Stuart Russell's textbook on AI as its base.

[3] There is the funny event, when Jörg was invited for one of his well-paid AI-intros to German industrialists, in this case called "Schock der Moderne", and he hesitated to go as he had to care for his dog that day. So they sent two chauffeur driven big black Mercedes cars headed by a motorbike leading the convey to little Kaiserslautern University: one with the back seat removed for his dog and the other one for himself.

"probably the only paper I ever wrote that was really read by others and had some influence", as Jörg used to say. It was certainly read by the officials of the GI who threatened to expel him from the German computer science society, if he would continue to announce publicly that there was no difference in principle between human and machine thinking[4].

However, in practice Jörg was now able to observe the disturbing and most obvious difference between human and machine thinking: In 1985 daughter Helen was born and all along the years Jörg intensely studied and proudly reported the progress and evolution of this young brain built on protoplasm rather than silicon, whereas his primal scientific child, MKRP, would not at all live up to his expectations.

On an initiative of Jörg together with Peter Deussen, Peter Raulefs, and Wolfgang Wahlster, a new collaborative research centre of the DFG (the German national science foundation), called Sonderforschungsbereich 314, had started in 1985. Not only was its title "Künstliche Intelligenz" (AI) still provocative, it also violated all the rules since it was not only one of the largest SFB's ever, but it also spread over three universities (Karlsruhe, Kaiserslautern and Saarbrücken) who were soon to become major centres of AI-research in Germany besides the strong groups in Hamburg. Peter Deussen became its first chairman: KI – the German acronym for AI – had finally entered the territory of the scientific establishment and many of the later institutions (like the DFKI and others) can be traced back to this research initiative.

This SFB formed the basic framework for the development of MKRP. Much effort was spent in order to resolve the weaknesses in dealing with equational theories. Starting already in Karlsruhe, a difference reduction approach was developed and integrated in MKRP. However, since it could not compete with the upcoming term rewriting systems, horses were changed again and some sort of term rewriting was integrated into MKRP. The exploration of the theoretical properties of the connection-graph calculus attracted many researchers not only in Jörg's group and also caused some heated arguments about the first origins of (sometimes incorrect) proofs. Although theorem provers based on connection-graphs don't exactly flourish any more, we now know that connection-graphs are confluent and weakly complete. The cumbersome progress in developing strategies for MKRP promoted the upturn (and revival) of more basic research topics like unification theory or sorted logics in his group. By 1990 they had coded and proved, as promised, much of Deussen's textbook on automata theory and transformed and finally translated these proofs automatically into natural language as well – but, to anyone involved, the shortcomings were all too apparent: this was not a mathematical assistant system by anybody's standard and more seriously, the research paradigm of the seventies and eighties – search based or partially knowledge guided as in MKRP – seemingly did not permit the construction of one either. A new paradigm had to be found!

[4] He continued to do so, the motion was nonetheless cancelled – and now 20 years later, Jörg was honoured as a fellow of the GI in honour of his contributions to the field of AI and his work for the GI.

And there is a spot in Jörg's heart that makes it different: When the MKRP-effort did not live up to expectations he did something unusual: he announced publicly that they had failed[5] and asked the funding agency if they were allowed to use the rest of the money to look for alternatives[6]. Strangely this was granted.

Meanwhile AI had finally established itself in the German scientific community. In the mid eighties Jörg persuaded Springer to have a new series of lecture notes on AI in order to increase the international recognition of AI (Germany being late by at least twenty years in comparison to England and the US with its legendary Dartmouth Conference in 1956). It was the wise decision of Hans Wössner of the Springer Company to integrate LNAI as a part of the larger LNCS series. With Jörg being the general editor – now jointly with Jaime Carbonell – LNAI became the most widely distributed series on AI worldwide.

Within the German computer science association (Gesellschaft für Informatik) AI had developed from a small working group into a well recognized Fachbereich until – under Jörg's chairmanship – it had more than 4000 members. Time had come to push for more. In one of the most dramatic and impassioned presidential management committee meetings of the GI, Jörg negotiated a new structure for the GI: the good old society was now to rest forever upon four pillars instead of the previous three divisions of Computer Science[7]: 1. Theory, 2. Software and 3. Hardware. But who was to be the number one? When the meeting was on the brink of collapse and Jörg threatened to form an independent AI society , the chairman of the theory division, Wilfried Brauer, suggested in a brilliant and hilarious motion that THEORY would be willing to become number ZERO – so AI could become number one and software and hardware would follow suit as section three and four. The menacing threat of an independent AI society[8] was off the table and later on, section number ONE became one of the best organised and largest AI-societies worldwide.

In an unusually farsighted move the German government had commissioned an expert advisory review in the seventies on the state of German industry (and universities) with results that became apparent to everyone only ten or fifteen years later. The report stipulated that while Germany's manufacturing was still healthy in its traditional areas such as car building, chemistry or mechanical engineering, it was in danger of losing its competence in fields based on more recent research such as computer science, genetic engineering, new materials to

[5] "Look, Mr. President, Sir, we can get a man on the moon, but to do so, we need n-billion dollars. And if after m years the man is not on the moon, you have to say so: Sorry Sir, there was a certain amount of risk involved, and we have failed" – this is his favourite story line.

[6] It sounds easy, but MKRP had a certain amount of visibility, even in the German media where "A computer-generated mathematical textbook" played the role of the "man on the moon". Older subjects like physics and chemistry have an established record of honourable failure, but in computer science and AI it still appears to be rare.

[7] Actually called under the much broader name Informatics in Germany right from its start.

[8] Actually as in almost all of the other industrial nations at the time.

replace the end of the iron age, molecular biology or the life sciences. Likewise, the report stated, German universities – still captured within their Humboldian values and traditions, – may still be better than their reputation, but too slow to adapt and to open up to new subjects.

The result of these findings was the decision to found about two dozen so-called an- Institutes, i.e. research institutions on the campus of a university but legally separated, which could act much faster than over-bureaucratised German universities. These should be able to build a bridge between industrial research labs and production on the one hand and basic university research on the other.

This was a big chance for AI as well and Jörg negotiated with the German ministry to include AI in the list of "new" subjects. Michael Richter, Jörg and other colleagues from Kaiserslautern filed a bid for such an institute and when they joined forces with Wolfgang Wahlster from the nearby university in Saarbrücken, they finally won the national competition. The German Research Centre for Artificial Intelligence, the DFKI GmbH, was born as a research company (Ltd) with almost all big firms from Germany as actual shareholders and with funds for an initial period of ten years equivalent to about 100 million US$. Within the next fifteen years the institute grew into one of the largest, most innovative and in some areas internationally leading AI research labs still situated both at Kaiserslautern and Saarbrücken with more than two hundred researchers today.

5 A New Start: Saarbrücken

"Louis, I think this is the beginning of a wonderful friendship."

Michael Curtiz: Casablanca, 1942

Offended by the fact that the smallest and poorest Länder of the federal republic, Saarland and Rheinland-Pfalz, had won the AI-race, some other states of Germany opened AI- institutions of their own, and Berlin offered Jörg a chair and the founding directorship for another AI institute. Fortunately a chair and the accompanying AI research department within the DFKI at Saarbrücken were vacant as well and finally Jörg moved to Saarbrücken to become one of the local DFKI directors joining Wolfgang Wahlster, Hans Uszkoreit and Gert Smolka. He received a joint position for the chair of AI in the computer science department of the university and his research department at the DFKI.

The challenges of a large research institute depending on external funding accelerated the diversification of research topics in Jörg's groups. Starting already in Kaiserslautern, knowledge representation and description logics became favourite research topics in one of his groups. The research had a strong theoretical touch and has been internationally recognized for its classification of description logics with respect to their complexity classes. It also resulted in one of the fastest (at that time) classifier systems called Kris. Another part of Jörg's group started research in multiagent systems. Its first achievement[9] was

[9] The system won a gold medal in the system competition at one of the MAS conferences.

the development of a general purpose layered architecture called INTERRAP that combines deliberative and reactive reasoning with multiagent (i.e. social) planning. The system is still used inside many industrial applications including the seminal transportation domain which sparked off the work on holonic multiagent systems. Software verification has always been a prominent application area for automated deduction. So, when the German security agency BSI advertised funding for the construction of a "national" tool for formal software development, Jörg enticed us to move from Karlsruhe to Saarbrücken in order to merge and amend the already existing theorem provers KIV and INKA to form the kernel of an integrated Verification Support Environment (VSE). During the following years safety and security problems became more and more a real issue in industry and the industrial applications of VSE with its engineering problems of verification in the large became a major part of daily business. The practical challenges of evolutionary formal software development spawned the work on "management of change" that turned out to be of much wider applicability.

Recently a new research lab on e-learning opened its doors for the development of a datamining tool called DAMIT and an internationally recognized learning environment for mathematics called ActiveMath, which was recently honoured as the best system of its kind by the funding EU-authorities.

It is to Jörg's credit that all these groups are now established and flourish in his department and most importantly: they interact and plenty of interdisciplinary papers have come out of it. Besides the dramatic increase of research issues under Jörg's responsibility it is perhaps his continuous effort to reconcile diverging, conflicting, and more often than not inconsistent aims and values that best characterizes his time in Saarbrücken. In particular this is true with respect to his scientific, social, and political enthusiasm of the past. Early socio critical reflections more than ever were confronted with the needs of the DFKI as an institution that became a global player demanding an annual budget of up to four or five million Euros he had to raise for his department. The struggle for peace and disarmament becomes more difficult in a situation where relevant parts of the research budgets all over the world stem from the various departments of defence, and, inevitably, the German armed forces suddenly can be found among the customers of the DFKI.

The ideal self-determined life of a scientist, which Jörg possibly had in mind when entering the academic stage, differs much from the extremely disciplined time management necessary to fulfil DFKI's management duties, international obligations, and the usual professorial duties of university life. Scientific discovery as an end in itself leads not necessarily to well engineered solutions for problems and furthermore scientific solutions have to be transformed into products for the actual market of technological innovations. Leading a large department at an application driven research institution gives rise to the question of how well do we ride on the technology wave: being too late there is the usual punishment that someone else has received the research grant or industrial contract – but being too early will not win any industrial contract either.

As so often before, Jörg did not choose the straight and narrow way of resolving all these conflicts by ultimately adjusting his conception of life in this or that direction. May be it's his way to live with the tension of antagonistic forces that made him view it all from a certain distance[10] and is one source of his well-known behaviour in every day discussions and private conversations[11], as well as in serious confrontations. They often get straight to the heart of the problem thereby opening the way for unconventional solutions but sometimes run the risk of damaging a personal relationship that has grown over many years. But knowing what academic life is like[12], it is astonishing: even now, after more than fifteen years of very close collaboration and competition, they still work together effectively and, in particular, the friendship between Jörg, Hans Uszkoreit and Wolfgang Wahlster saved the DFKI more than once in a moment of existential crisis.

At the university in his basic research group, the aftermath of MKRP's failure dominated the discussions of the early nineties. Why did the MKRP-effort fail? Well, it was certainly not a complete failure, but then: why did it not live up to its expectations? After all, it was based on mainstream research assumptions of artificial intelligence, i.e. transporting the water of knowledge based techniques into the intellectual desert of search based automated theorem proving. In Jörg's opinion it is not knowledge-based AI that failed, but their own apparent lack of radicalism. While on the bottom of MKRP there was a graph based, first order theorem proving mechanism that was optimized for a non-informed search, there was the plan of a superviser module incorporating the necessary domain knowledge in mathematics and controlling effectively the logic engine. But the distance between this general mathematical knowledge to be represented in the supervisor and the low level of abstraction of the logic engine was just too much and the supervisor module never went into existence. Jörg's favourite variation on McCarthy's quote "Nothing can be explained to a stone" was "Nothing can be explained to a first order theorem prover".

A paradigm shift, as Jörg used to phrase it, was again on the agenda: instead of investigating calculi and their search spaces, the representation of mathematical knowledge itself became the favourite research topic. Ideas were tossed around to raise the abstraction level of the representation and to encapsulate chunks of mathematical knowledge such that they could be chained into an abstract proof. The idea of proof planning, developed by Alan Bundy to combine tactic-based theorem proving with AI planning techniques, now fell on fertile soil: "knowledge based proof planning" became the new battle cry of this research group.

When Saarbrücken applied for an interdisciplinary collaborative research centre on "Ressource-adaptive cognitive processes" (SFB-378), Jörg saw his second chance: a new project called ΩMEGA was approved and for a funding period

[10] "...seven professors and directors competing with their publication record and their respective annual research budgets is a sight not totally unfamiliar from the baboon's hill in the local Kaiserslautern Zoo, where the silver back may change over night" is his favourite quotation.

[11] His so-called "Waldspaziergänge" with those who are supposed to deviate.

[12] If you are unfamiliar with these mechanisms, David Lodge's books (e.g. THINKS, Penguin Books Ltd, 2002) provide a good source of background reading.

of twelve years he had the chance to start all over again to realise his dream. The ΩMEGA project, by now with additional funding from other sources again one of the largest all-out efforts to build a proof assistant and mathematical support system, is carried out at the University (and not at the DFKI with its application driven pressure) as an independent research group. It is here where Jörg's heart can be found – and they still have another four years to go.

Apart from his role in building and establishing artificial intelligence in Germany, Jörg has been very active in recent years in another academic / political / institutional endeavour: Logic and AI. The community divided the field into dozens of societies, conferences and workshops. While this specialization and these factions are not necessarily a disadvantage, there is a lack of unity. So upon Jörg's initiative the International Federation for Computational Logic (IFCoLog) was set up with Dana Scott as the founding president. The European Network of Excellence on Computational Logic (CoLog) provided most of the support for this far reaching international effort to unite and establish computational logic as a subject of its own, on a par with maths, physics, chemistry and the other academic disciplines. With Moshe Vardi now as the acting president of IFCOLOG and FLOC as its mayor unifying event, things are now – after many years of travelling, convincing people, getting the finances right – in good shape.

In a similar vein he joined forces recently with Dov Gabbay and his mission to establish logic as a unifying foundational subject not just for mathematics as in the past, but for the much grander agenda of computer science, AI, the cognitive sciences and practical as well as jurisdictional reasoning. As chairmen of CoLoG-Net, they started a new journal for applied logic (JAL) that encompasses and unites for the first time the different logic factions, and they established together with John Woods and Johan van Benthem the new red series as a mirror to the seminal yellow series on mathematical logic by Elsevier: a new landscape of computational logic is opening up and this may well be the new continent that is to survive the tides of today.

But time has come for Jörg now to take stock and harvest and, with about seventy successful PhD's supervised, about hundred and fifty papers and books and innumerable publications in collaboration with his research assistants, there is plenty to chose from[13].

At Jörg's 60th birthday party at the University and DFKI – customarily a stiff academic event with plenty of mental incense around – his former PhD students formed a mixed choir for their self-composed song, which we do not want to deny to the well intended reader of an intellectually demanding volume such as this. It may be worth mentioning, that Jörg's favourite anecdote about his friend and greatly admired colleague Alan Bundy records an incident at the 7th CADE at Kaiserslautern, when Alan struggled with a non-operating microphone in his invited talk. Having received a replacement (which did not work either), he finally grasped the microphone, switched it off and in true Frank-Sinatra-esque style sang spontaneously: "I'll do it my way..." – and made history. So finally here is our version of the song to the well-known tune:

[13] http://www-ags.dfki.uni-sb.de

JÖRG'S WAY

And now, the end is near;
And so you face the final lecture.
My friend, we'll say it clear,
We'll state our case, it's no conjecture.

You've lived a life that's full.
Aimed at awards and ev'ry female,
But more, much more than this,
You did it your way.

Regrets you have had none;
Attacked colleagues and other 'primates'.
Your faith, it has been born
In the sixties anarchic climate.

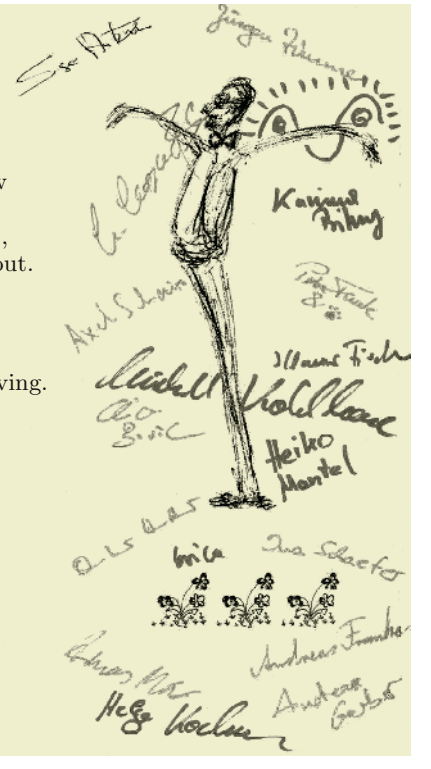
You planned all flow'ry slides
To put AI on every highway
But more much more than this
You did it your way.

Yes, there were proofs, I'm sure you knew
That you just took out of the blue.
But through it all, when there was doubt,
You planned them all and worked them out.
We faced it all, and we stood tall
And did it your way.

You've moaned, shouted and cried,
You've had your thrill, to show good proving.
Your trip, your magic fight,
We found it all so amusing.

To think you did all that;
And may we say – not in a shy way,
Oh No, oh no not you,
You did it your way.

For what is Jörg, what is his pride?
Funding, first talk, and his first night.
To present things, you did not know
You had your team, to run the show
Your record shows how well it goes
On your AI way.



6 Epilogue

Hopefully his energy will not fade too soon, so his plans and actual collaborations will come to fruition in the new book series "Principia Mathematica Mechanico". It encompasses logic but also AI's contribution to this age-old dream of the possibility for an exact science, which came into life with Euclid's Elements, explicitely formulated in Leibnitz vision and Frege's realization of the Begriffsschrift and finally culminated in Whitehead and Russel's Principia Mathematica. This series will tell the story of *our* century's contribution – logic, computer science, and AI – to this quest for a science which is built upon exact and formal logical foundations.

Some Reflections on Proof Transformations*

Peter B. Andrews

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. Some general reflections on proof transformations lead to the abstract concept of a quintessential proof of a theorem. A quintessential proof embodies the essential features of many intertranslatable proofs of the theorem. While reasonable candidates for quintessential proofs for normal proofs have been developed, more work is needed with regard to non-normal proofs. Work on proof transformations seems likely to lead to important progress in understanding proofs.

1 Introduction

In this paper we shall focus more on questions and challenges than on solutions and techniques. We will stir the intellectual stew a little, try to provide some perspective as seasoning, and speculate on the flavor of some future developments in this field.

2 Information Libraries and Representation Issues

Proof transformations can play an important role in enhancing the usefulness of automated reasoning systems. Before discussing proof transformations, we discuss some aspects of automated reasoning systems which are related to the contexts in which proofs will be transformed.

As tools for automated deduction and other aspects of automated reasoning develop to the point where they are useful, there will be a tremendous drive to develop automated libraries of information which will permit one to retrieve not only information which has been stored in the libraries, but also verifiable consequences of that information. One can envision such libraries for mathematics, computer science, physics, chemistry, biology, genetics, medicine, engineering disciplines, economics, and other disciplines and subdisciplines.

Consider mathematics, as an example. While automated theorem provers may occasionally discover interesting new theorems, the major applications of automated reasoning for mathematics will be in helping those who are learning, applying, and developing mathematics, and in refereeing papers. At present,

* This material is based upon work supported by the National Science Foundation under grants CCR-9732312 and CCR-0097179. It is based on a lecture which was given June 19, 2001 to the Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs which was held in conjunction with IJCAR-2001 (International Joint Conference on Automated Reasoning) in Siena, Italy.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 14–29, 2005.

most of the work in refereeing a mathematical paper is usually devoted to checking that it is actually correct, and sometimes even very knowledgeable referees fail to spot errors. A referee should be able to rely on a computer to check that the work in the paper is correct, and simply be concerned with the significance of the results and the quality of the exposition in the paper. Since the major work of searching for the right ideas has been done by the author of a paper that is being refereed, checking the details of the proof is a far less ambitious task than finding it in the first place, and checking such proofs seems like a very reasonable task for the automated theorem provers of the future.

Of course, the computer will need not only adequate reasoning power, but the ability to translate the paper into its own notation, and a library containing all the mathematical information (definitions, theorems, proofs, examples, etc.) which is assumed by the author of the paper.

Many other applications of automated reasoning involving mathematics will also require libraries of mathematical information.

As we contemplate these information libraries which have deductive reasoning capabilities built into them, we must consider the problem of representation. All that information must be represented in some way. It would be very nice if good standards for representing information in such libraries could be established and agreed upon before work on developing the libraries goes very far, but we really don't know enough, and it's generally impossible to get everyone involved to agree on such things.

Serious thought should be given to designing good formats for representing information, but we shouldn't expect general agreement or acceptance. Uniformity will come only gradually through an evolutionary process. We will simply have to work with whatever formats have been developed for representing information, and have procedures for transforming from one format to another.

There's another problem associated with libraries: that of finding the information in them that may be useful to us. We can imagine a library in which a wealth of mathematical theorems have been stored as well-formed formulas (wffs) of a particular logical system, but it might be very difficult to find a particular theorem in this library if we did not know exactly how it was represented. No well defined and logically robust method of classifying theorems has yet been developed. Mathematicians sometimes have trouble finding out whether a theorem they have proved has been proved before, because there is no general index of mathematical theorems, and it seems hard to devise a good plan for developing such an index. Current mechanisms for finding whether a theorem has been proved before mainly rely on consulting experts.

For theorems expressed as wffs in a particular logical system, we would like wffs to be regarded as expressing the same theorem if they can be obtained from each other by making trivial logical changes, such as applying basic tautologies like the contrapositive law. Of course, this notion of sameness must be transitive, and we must take care that the notion of sameness does not become too broad by applying trivial changes repeatedly. We need a much narrower notion of

sameness than logical equivalence, since all theorems of a formal system are logically equivalent.

One possibility is to have a normal form for each wff, and say that two provable wffs express the same theorem if and only if they have the same normal form. A variety of proposals for normal forms for classifying theorems might be made, and we need criteria for accepting or rejecting them, and choosing among them.

3 Proof Transformations

Now let us focus on proof transformations. Some proof transformations will be needed simply because of differences in the ways proofs are represented in different systems. However, even if we had universally accepted standards for representing proofs, we would still want to transform proofs. Different styles, formats, or representations may be most suitable or simply preferred for different purposes or in different contexts, and the user should be able to freely choose whatever is most congenial.

There are various types of proof transformations. We next review some of them.

We may wish to translate a proof in one logical system into a different logical system. These systems might differ with respect to the basic language, the rules of inference, or both. The language might be a formal language of logic or natural language.

One example of such a transformation is cut-elimination. We translate a proof from a system in which there is a cut rule to the system obtained from that one by deleting the cut rule. Obviously, the study of proof transformations goes right back to Gentzen and Herbrand, and was an integral part of proof theory in their era.

We may wish to adjust the level of detail of the proof, either globally or locally. Ideally, automated systems for presenting proofs should allow the reader of a proof to see whatever degree of detail is desired for each part of the proof.

In particular, we may wish to provide more details. We may wish to have a system which automatically translates proofs from natural language into formal proofs with justifications for all the logical details. A nice example of an intuitively elegant proof which presents significant challenges for such a system is the proof of Euler's Formula for Polyhedra presented in [12, pp. 236-240]. The proof starts as follows:

> "let us imagine the given simple polyhedron to be hollow, with a surface made of thin rubber. Then if we cut out one of the faces of the hollow polyhedron, we can deform the remaining surface until it stretches out flat on a plane. Of course, the areas of the faces and the angles between the edges of the polyhedron will have changed in this process. But the network of vertices and edges in the plane will contain the same number of vertices and edges as did the original polyhedron, while the number of polygons will be one less than in the original polyhedron, since one face was removed. "

Let g be an iterate of f, and let x be the unique fixed point of g.
$g\,x = x$.
$f\,[g\,x] = f\,x$.
$g = f \circ \ldots \circ f$, so $f \circ g = g \circ f$, so
$f\,[g\,x] = g\,[f\,x]$.
$g\,[f\,x] = f\,x$.
Thus, $f\,x$ is also a fixed point of g. Since x is the *unique* fixed point of f,
$f\,x = x$.
Therefore, f has a fixed point.

Fig. 1. Informal proof of THM15B.

Obviously, one of the many challenges for the translation system is to transform the reference to "thin rubber" into appropriate topological terminology.

Alternatively, we may wish to transform a very detailed proof into a shorter, "higher level", proof. For example, a complete proof of THM15B which was found by the automated theorem proving system TPS [6] is displayed in Figure 2. THM15B says that if some iterate of a function has a unique fixed point, then that function has a fixed point. A shorter, "higher level", proof of this theorem, which is easier for people to understand, is displayed in Figure 1. This proof has the same logical content as the proof found by TPS, but it is not simply a drastic condensation of that proof. It contains the assertion "$g = f \circ \ldots \circ f$", which suggests certain ideas and helps the reader to understand why $f \circ g = g \circ f$ without making an explicit reference to the inductive definition of the set of iterates of a function. This illustrates the fact that when we transform a proof, we may wish not only to change the level of detail in the presentation, but also to add features which are intended to help the reader understand the proof.

Similarly, when we analyze a proof presented in natural language, we may need to distinguish between the proof itself and "comments" which are intended to help the reader understand the proof.

Some proof transformations may simply be used to improve the way the proof is presented. Sometimes one can do this simply by translating a proof into a different format and then back again.

Proof transformations may be used to change the format in which a proof is presented. We should be able to choose between proofs presented in a traditional linear style, proofs arranged as trees, and various other formats. Radically new proof formats may be developed. For example, we can anticipate the development of facilities for using diagrams in mathematical proofs in ways that involve no compromise with complete rigor. (See [8] for persuasive arguments pertinent to this.) At present we encourage our students to use Venn diagrams to illustrate theorems about sets, but we tell them that arguments involving Venn diagrams are not really proofs. However, this could change. We can anticipate the development of computer programs which have very precisely defined facilities for manipulating Venn diagrams, and rigorous metatheories suporting these programs which guarantee the soundness of the conclusions reached by using their facilities. The relevant theory for Venn diagrams is well developed [27, 60]. Similarly, there may be additional types of arguments, like diagram chasing

(1)	1	$\vdash$	$\exists g_{\iota\iota}.\ \mathrm{ITERATE}+\ f_{\iota\iota}\, g$ $\wedge \exists x_\iota.\ g\, x = x$ $\wedge \forall z_\iota. g\, z = z \supset z = x$	Hyp
(2)	1,2	$\vdash$	$\mathrm{ITERATE}+\ f_{\iota\iota}\, g_{\iota\iota}$ $\wedge \exists x_\iota. g\, x = x \wedge \forall z_\iota. g\, z = z \supset z = x$	Choose: $g_{\iota\iota}$ 1
(3)	1,2	$\vdash$	$\mathrm{ITERATE}+\ f_{\iota\iota}\, g_{\iota\iota}$	RuleP: 2
(4)	1,2	$\vdash$	$\exists x_\iota. g_{\iota\iota}\, x = x \wedge \forall z_\iota. g\, z = z \supset z = x$	RuleP: 2
(5)	1,2	$\vdash$	$\forall p_{o(\iota\iota)}. p\, f_{\iota\iota} \wedge \forall j_{\iota\iota}[p\, j \supset p.\, f \circ j] \supset p\, g_{\iota\iota}$	EquivWffs: 3
(6)	1,2	$\vdash$	$\forall p_{o(\iota\iota)}.\quad p\, f_{\iota\iota} \wedge \forall j_{\iota\iota}[p\, j \supset p.\lambda x_\iota\, f.\, j\, x]$ $\supset p\, g_{\iota\iota}$	EquivWffs: 5
(7)	1,2,7	$\vdash$	$g_{\iota\iota}\, x_\iota = x \wedge \forall z_\iota. g\, z = z \supset z = x$	Choose: x_ι 4
(8)	1,2,7	$\vdash$	$g_{\iota\iota}\, x_\iota = x$	RuleP: 7
(9)	1,2,7	$\vdash$	$\forall z_\iota. g_{\iota\iota}\, z = z \supset z = x_\iota$	RuleP: 7
(10)	1,2,7	$\vdash$	$g_{\iota\iota}[f_{\iota\iota}\, x_\iota] = f\, x \supset f\, x = x$	UI: $[f_{\iota\iota}\, x_\iota]$ 9
(11)	1,2	$\vdash$	$[\lambda j_{\iota\iota}. f_{\iota\iota}[j\, x_\iota] = j.\, f\, x]\, f$ $\wedge \forall j[\ [\lambda j.\, f[j\, x] = j.\, f\, x]\, j$ $\supset [\lambda j.\, f[j\, x] = j.\, f\, x].\lambda x\, f.\, j\, x]$ $\supset [\lambda j.\, f[j\, x] = j.\, f\, x]\, g_{\iota\iota}$	UI: $[\lambda j_{\iota\iota}. f_{\iota\iota}[j\, x_\iota] = j.\, f\, x]$ 6
(12)	1,2	$\vdash$	$f_{\iota\iota}[f\, x_\iota] = f[f\, x]$ $\wedge \forall j_{\iota\iota}[\ f[j\, x] = j[f\, x] \supset f[f.\, j\, x] = f.\, j.\, f\, x]$ $\supset f[g_{\iota\iota}\, x] = g.\, f\, x$	Lambda: 11
(13)		$\vdash$	$f_{\iota\iota}[f\, x_\iota] = f.\, f\, x$	Assert REFL=
(14)	14	$\vdash$	$f_{\iota\iota}[j_{\iota\iota}\, x_\iota] = j.\, f\, x$	Hyp
(15)		$\vdash$	$f_{\iota\iota}[f.\, j_{\iota\iota}\, x_\iota] = f.\, f.\, j\, x$	Assert REFL=
(16)	14	$\vdash$	$f_{\iota\iota}[f.\, j_{\iota\iota}\, x_\iota] = f.\, j.\, f\, x$	Subst=: 15 14
(17)		$\vdash$	$f_{\iota\iota}[j_{\iota\iota}\, x_\iota] = j[f\, x]$ $\supset f[f.\, j\, x] = f.\, j.\, f\, x$	Deduct: 16
(18)		$\vdash$	$\forall j_{\iota\iota}.\ f_{\iota\iota}[j\, x_\iota] = j[f\, x]$ $\supset f[f.\, j\, x] = f.\, j.\, f\, x$	UGen: $j_{\iota\iota}$ 17
(19)		$\vdash$	$f_{\iota\iota}[f\, x_\iota] = f[f\, x]$ $\wedge \forall j_{\iota\iota}.\ f[j\, x] = j[f\, x]$ $\supset f[f.\, j\, x] = f.\, j.\, f\, x$	RuleP: 13 18
(20)	1,2	$\vdash$	$f_{\iota\iota}[g_{\iota\iota}\, x_\iota] = g.\, f\, x$	MP: 19 12
(21)		$\vdash$	$g_{\iota\iota}[f_{\iota\iota}\, x_\iota] = g.\, f\, x$	Assert REFL=
(22)	1,2	$\vdash$	$g_{\iota\iota}[f_{\iota\iota}\, x_\iota] = f.\, g\, x$	Subst=: 21 20
(23)	1,2,7	$\vdash$	$g_{\iota\iota}[f_{\iota\iota}\, x_\iota] = f\, x$	Subst=: 22 8
(24)	1,2,7	$\vdash$	$f_{\iota\iota}\, x_\iota = x$	MP: 23 10
(25)	1,2,7	$\vdash$	$\exists y_\iota. f_{\iota\iota}\, y = y$	EGen: x_ι 24
(26)	1,2	$\vdash$	$\exists y_\iota. f_{\iota\iota}\, y = y$	RuleC: 4 25
(27)	1	$\vdash$	$\exists y_\iota. f_{\iota\iota}\, y = y$	RuleC: 1 26
(28)		$\vdash$	$\exists g_{\iota\iota}[\ \mathrm{ITERATE}+\ f_{\iota\iota}\, g$ $\wedge \exists x_\iota.\ g\, x = x \wedge \forall z_\iota. g\, z = z \supset z = x]$ $\supset \exists y_\iota. f\, y = y$	Deduct: 27
(29)		$\vdash$	$\forall f_{\iota\iota}.\ \exists g_{\iota\iota}[\ \mathrm{ITERATE}+\ f\, g$ $\wedge \exists x_\iota.\ g\, x = x \wedge \forall z_\iota. g\, z = z \supset z = x]$ $\supset \exists y_\iota. f\, y = y$	UGen: $f_{\iota\iota}$ 28

Fig. 2. Formal proof of THM15B found by TPS.

in category theory, which could be made into formal methods of proof. Of course, we would like to be able to translate between such proofs and more traditional logical formats.

4 Criteria for Correctness

Now let's think about proof transformations from a general perspective. One natural question to ask about proof transformations concerns criteria for correctness. How do we know, or decide, whether a translation of a proof is correct, or satisfactory? Of course, different criteria may be appropriate for different purposes.

We can take the point of view that when we translate a proof, we are just changing the way it is presented, but the essential underlying proof should remain the same. Thus we are asking, if we are given two concrete proofs, how do we decide whether they are "essentially the same", although they may differ in superficial ways?

When we translate statements from one natural language to another, our criterion is that we should preserve the meaning. When we translate a proof, what is our criterion?

One criterion we might adopt is that the translated proof should at least prove the same theorem as was proved by the original proof. Even this may not be trivial. We might regard the theorem proved as the same even if our representations of the theorem in the two proofs are not identical. This is closely related to the problem of classifying theorems, since two statements express the same theorem only if these statements of the theorem are classified together.

Another basic criterion is that a translation of a correct proof should be a correct proof.

For certain purposes, we may be satisfied if the result of a proof translation is a correct proof in the desired format of the same theorem. However, in some contexts we might like to impose stronger criteria for correctness of the translation. Perhaps the proofs should use the same "methods" or "key ideas" as these are manifested in each context. What are the "key features" of a proof which should be preserved by any correct translation procedure?

Some possible answers include general methods, tacticals, and tactics which can produce the proofs. Alternatively, it might be suggested that the terms with which quantifiers are instantiated should be the same.

5 Quintessential Proofs

Imagine that we have many different proofs of the same theorem in a variety of styles, formats, levels of detail, and formal systems, but which can all be translated into each other in ways that satisfy whatever criteria we have for correctness. Is there some identifiable set of features which are common to these proofs and which characterize this set of proofs? If so, we can associate with this set of features the abstract idea of the *quintessential proof* which is manifested

by all the particular proofs mentioned above. A quintessential proof contains the essential content of the proof, which must be embodied in a particular form to obtain a concrete proof. We may regard every proof as a particular manifestation of the quitessential proof which underlies it.

If we can find some way of representing the features which characterize the quintessential proof, we can say that a translation or transformation of a proof may be regarded as correct if it preserves the features of its quintessential proof.

Ideally, one should be able to use a quintessential proof as a hub, like the airport which an airline uses as its hub, and have all translations of a proof from one format to another be accomplished in a uniform manner by translating from the initial format into the quintessential proof, and then into the alternative format. One would like to have general methods of producing proofs in many formats from the quintessential proof so that one could automatically generate a particular proof simply by specifying the quintessential proof and various aspects of the presentation, such as the logical system to be used, the style, and the level of detail for various parts of the proof. Ideally, the reader of the proof should be able to adjust various aspects of the proof presentation while reading it, like the user of a map program who can zoom in on a specified part of the map.

Let us consider some ideas which are relevant to the problem of representing the quitessential features of proofs.

In [2], the idea of a *plan* for a proof of a theorem of classical first-order logic was introduced. A plan was a quadruple consisting of a normalized form of the theorem to be proved, a replication scheme describing how often various quantifiers needed to be replicated, a mating [3] of the literals of the replicated theorem, and the substitution for variables associated with the mating. The plan provided all the information needed to produce various proofs of the theorem in natural deduction style.

However, the way this information was represented in these plans did not facilitate proving certain metatheorems, or generalize gracefully to higher-order logic, where substitutions for variables can introduce new quantified variables for which additional substitutions may be made. Therefore, building on ideas in [7], Dale Miller developed [48–50] the idea of an *expansion tree proof*, known more briefly as an *expansion proof*, which represents the same information more elegantly and works equally well in higher-order logic. An expansion proof is a generalization of the notion of a Herbrand expansion of a theorem of first-order logic. It provides a very elegant, concise, and nonredundant representation of the relationship between the theorem and a tautology which can be obtained from it by appropriate instantiations of quantifiers, and which underlies various proofs of the theorem. (A closely related concept is that of a *dual expansion proof*, which represents in a similar way the relationship between a refutable wff and the contradiction buried within it.)

Let's review what an expansion proof is.

A familiar way of representing a wff of first-order logic is to regard it as a tree (i.e., a connected graph with no cycles) growing downward from a topmost node which is called its root. With each node of the tree is associated a propositional

connective, quantifier, or atom, and if the node is not associated with an atom, there are subtrees below the node which represent the scopes of the connective or quantifier. Such a tree can be enhanced with additional structure to produce an expansion tree for the wff.

The wff represented by a tree Q as described above is called the *shallow formula* $\mathrm{Sh}(Q)$ of the tree (and of the node which is its root). With each expansion tree Q is also associated a *deep formula* $\mathrm{Dp}(Q)$ which represents the result of instantiating the quantifier-occurrences in Q with terms which are attached as labels to the arcs descending from their nodes.

A node of a tree is called an *expansion node* if it corresponds to an essentially existential[1] quantifier, and a *selection node* if it corresponds to an essentially universal quantifier. (In dual expansion trees, the roles of universal and existential quantifiers are interchanged.) Let finitely many arcs labeled with terms descend from each expansion node, so that if the expansion node has shallow formula $\exists \mathbf{x}\, \mathbf{B}$, and if $\mathbf{t}$ is the term labeling an arc descending from that node, then the type of $\mathbf{t}$ is the same as that of $\mathbf{x}$, and the node at the lower end of that arc has as shallow formula the β-normal form of $[[\lambda \mathbf{x}\ \mathbf{B}]\ \mathbf{t}]$, i.e., the result of instantiating the quantifier with the term $\mathbf{t}$. The term $\mathbf{t}$ is called an *expansion term*, and $\mathbf{x}$ is called an *expansion variable*. Selection nodes satisfy a similar condition, except that only one arc may descend from a selection node, and the term labeling it must be a suitably chosen parameter (which is not free in $\mathrm{Sh}(Q)$) called a *selected parameter*.

If Q is an expansion node of an expansion tree and Q_1, ... Q_n are the nodes immediately below Q, then $\mathrm{Dp}(Q)$ is $[\mathrm{Dp}(Q_1) \vee ... \vee \mathrm{Dp}(Q_n)]$. (In a dual expansion tree, however, $\mathrm{Dp}(Q)$ is $[\mathrm{Dp}(Q_1) \wedge ... \wedge \mathrm{Dp}(Q_n)]$.) The deep formula of a leaf of such a tree is the same as the shallow formula of that leaf.

An expansion tree is an *expansion proof* for its shallow formula if it satisfies certain conditions, including the condition that there is a mating of its nodes which establishes that its deep formula is a tautology. We refer the reader to the cited references for the technical details, and simply give a rather trivial example.

Consider the wff

$$\forall x Px \supset Pa \wedge Pb \tag{1}$$

We shall temporarily regard $\sim$, $\wedge$, $\vee$, $\forall$ and $\exists$ as primitive, and write (1) as

$$\exists x \sim Px \ \vee\ [\, Pa \ \wedge\ Pb\,] \tag{2}$$

An expansion tree (indeed, an expansion proof) for (2) is shown in Figure 3. The deep formula for this expansion tree is

$$[\sim Pa \vee\ \sim Pb\,] \ \vee\ [\, Pa \ \wedge\ Pb\,] \tag{3}$$

which is a tautology from which the wff (2) can readily be derived.

Of course, an alternative way to establish (1) is to derive a contradiction from the wff

$$\forall x Px \ \wedge\ [\sim Pa \vee\ \sim Pb\,] \tag{4}$$

[1] See [4, p. 123].

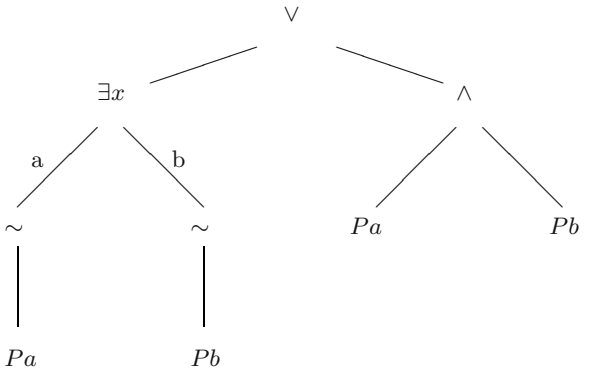


Fig. 3. A simple expansion tree.

which is equivalent to the negation of (1). A dual expansion tree (indeed, an expansion refutation) for (4) is in Figure 4. Its deep formula is

$$[\,Pa \wedge Pb\,] \wedge [\,\sim Pa \vee \ \sim Pb\,] \tag{5}$$

which is a contradiction.

Miller showed that a wff of type theory [or first-order logic] is a theorem of elementary type theory[2] [or first-order logic] if and only if it has an expansion proof. An expansion proof of a theorem contains the essential information which is needed to construct proofs of the theorem in a variety of styles.

It should be noted that if one simply uses the information in an expansion proof to generate a proof in natural deduction style, it will be normal; similarly, a sequent-style proof generated from an expansion proof will be cut-free.

If we confine our attention for the moment to normal proofs in classical first-order logic or elementary type theory, and we use expansion proofs to represent quintessential proofs, we have a criterion of correctness for translations of such proofs: the translation is correct if both proofs induce the same expansion proof.

One can find in the literature on proof transformations a variety of additional ideas which are relevant to the question of how quintessential proofs can be represented. For example, the TRAMP system [47] translates outputs from various automated theorem provers into refutation graphs, which are then translated into natural deduction proofs at the assertion level. Ideas for representing quintessential proofs in a variety of non-classical as well as classical logics may be found in [55] and [57].

While every theorem of first-order or higher-order logic has a normal or cut-free proof, certain important features in the organization of certain proofs, such

[2] We use *elementary type theory* as a name for the logistic system obtained by deleting the axioms of extensionality, descriptions, choice, and infinity from the system of [10]. Thus, elementary type theory includes only axioms 1-6 of [10], and it simply embodies the logic of propositional connectives, quantifiers, and λ-conversion in the context of type theory. Elementary type theory is presented under the name $\mathcal{T}$ in [1].

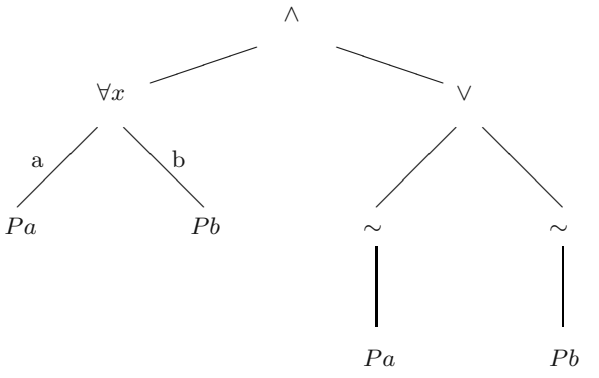


Fig. 4. The dual expansion tree.

as the use of lemmas, do not fit well into the context of normal proofs, and we need a concept more general than expansion proofs to represent quintessential non-normal proofs adequately.

We need much more investigation to clarify what are the important features of proofs in various logical systems and styles of proof presentation, as well as ways of representing these features.

Just as we need high level proofs in object languages, we need better ways to describe proofs at a high level in our meta-language.

6 A Challenge: Two Formulations of Cantor's Theorem

Here's a challenge for high level proof translation techniques. We want to translate a proof between systems which have different definitions for the same intuitive concept. The theorem is Cantor's Theorem that the power set (set of all subsets) of a set is bigger than the set. The details of the proofs depend on how one defines "bigger", which can be done very naturally in two different ways. Let's examine the precise formulations of Cantor's Theorem and its proof which we get with each definition.

The first definition says that a set W is bigger than a set U iff there is no surjection from U onto W. This leads to:

The Surjective Cantor Theorem: There is no surjective function from a set onto its power set.

Proof: Let U be a set and let $W = \mathcal{P}(U)$ (the power set of U). Suppose there is a function $g : U \to W$ such that (1) g is surjective. Let (2) $D = \{x \mid x \in U \text{ and } x \notin gx\}$. By (1) there is a j in U such that (3) $gj = D$. If $j \in D$, then $j \notin D$ by (2, 3). If $j \notin D$, then $j \in D$ by (2, 3). Thus we get a contradiction in each case, so there can be no such g. ■

The second definition says that a set W is bigger than a set U iff there is no injection from W into U. This leads to:

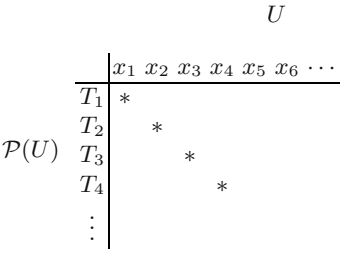


Fig. 5. The Diagonal Argument.

The Injective Cantor Theorem: There is no injective function from the power set of a set into the set.

Proof: Let U be a set and let $W = \mathcal{P}(U)$. Suppose there is a function $h : W \to U$ such that (1) h is injective. Let (2) $D = \{ht \,|\, t \in W \text{ and } ht \notin t\}$. Note that (3) $D \in W$. Now suppose that (4) $hD \in D$. Then by (2) there is a set t such that (5) $t \in W$ and (6) $ht \notin t$ and (7) $hD = ht$. Therefore (8) $D = t$ by (1, 7), so (9) $hD \notin D$ by (6, 8). This argument (4-9) shows that (10) $hD \notin D$. Thus (11) $hD \in D$ by (2, 3, 10). This contradiction shows that there can be no such h. ■

The Surjective Cantor Theorem is a classic example for higher-order theorem proving, and it has been automatically provable for many years [7]. On the other hand, the Injective Cantor Theorem is much more difficult to prove automatically by methods which have been implemented thus far, since (as discussed in [5]) it has greater quantificational depth. Thus far a proof of it has not been generated automatically.

Intuitively, the quintessential proof of Cantor's Theorem is the diagonal argument associated with Figure 5. We suppose there is some sort of correspondence between the members $x_1, x_2, x_3, \cdots$ of the set U and the subsets $T_1, T_2, T_3, \cdots$ of U, and let $D = \{x_i \mid x_i \notin T_i\}$. It is easy to see that D differs from each subset T_i of U, but D is a subset of U, and this yields a contradiction.

Can the quintessential proof be used to generate the proofs of the Surjective and Injective Cantor Theorems?

In the case of the Surjective Cantor Theorem, we assume there is a surjective function $g : U \to \mathcal{P}(U)$ such that $gx_i = T_i$ for each i, so $D = \{x_i \mid x_i \notin T_i\} = \{x \mid x \notin gx\}$.

In the case of the Injective Cantor Theorem, we assume there is an injective function $h : \mathcal{P}(U) \to U$ such that $hT_i = x_i$ for each i, so $D = \{x_i \mid x_i \notin T_i\} = \{ht \mid ht \notin t\}$.

Thus, the quintessential proof shows us how to generate the key idea in each of these proofs. Of course, we would need some good formal representation of the quintessential diagonal argument in order to mechanize the process of producing these proofs in this way.

7 Modifying Proofs

Let's consider another motivation for trying to find good ways to represent quintessential proofs. When one is trying to prove a theorem interactively or semi-interactively, one often finds it necessary to make changes in a proof which is partially constructed. One may wish to change a definition or a lemma, and make all additional changes which consequentially become necessary in a perhaps lengthy proof. One should be able to just change the appropriate features of the quintessential proof, and automatically regenerate the partial proof in the form that the user is working with. Thus, quintessential proofs could play a fundamental role in modifying proofs as well as in translating them to other formats. Note that for this application of quintessential proofs to be practical, quintessential proofs would have to be defined for incomplete as well as for complete proofs.

8 Conclusion

Work on proof transformations is extremely important. Inevitably, there is a linkage between learning more about transformations of proofs, learning more about how proofs can be represented, and learning more about the essential nature of proofs. It seems quite likely that work oriented toward proof translations will lead to a deeper understanding of the essential nature of proofs. This may generate a new flowering of proof theory. And, of course, as we come to understand proofs better, we may also acquire a better understanding of how to search for them.

References

1. Peter B. Andrews. Resolution in Type Theory. *Journal of Symbolic Logic*, 36:414–432, 1971.
2. Peter B. Andrews. Transforming Matings into Natural Deduction Proofs. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 281–292, Les Arcs, France, 1980. Springer-Verlag.
3. Peter B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28:193–214, 1981.
4. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.
5. Peter B. Andrews, Matthew Bishop, and Chad E. Brown. System Description: TPS: A Theorem Proving System for Type Theory. In Mcallester [44], pages 164–169.
6. Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16:321–353, 1996.
7. Peter B. Andrews, Dale A. Miller, Eve Longini Cohen, and Frank Pfenning. Automating Higher-Order Logic. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, Contemporary Mathematics series, vol. 29, pages 169–192. American Mathematical Society, 1984.

8. Jon Barwise and John Etchemendy. Visual Information and Valid Reasoning. In Walter Zimmermann and Steve Cunningham, editors, *Visualization in Teaching and Learning Mathematics*, pages 9–24. Mathematical Association of America, 1991.
9. Daniel Chester. The Translation of Formal Proofs into English. *Artificial Intelligence*, 7:261–278, 1976.
10. Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.
11. Yann Coscoy, Gilles Kahn, and Laurent Théry. Extracting Text from Proofs. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications : Second International Conference on Typed Lambda Calculi and Applications, TLCA '95*, volume 902 of *Lecture Notes in Computer Science*, pages 109–123, Edinburgh, United Kingdom, 1995. Springer-Verlag.
12. Richard Courant and Herbert Robbins. *What is Mathematics?* Oxford University Press, 1941.
13. B. I. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. In McCune [45], pages 57–60.
14. Bernd I. Dahn and Andreas Wolf. A Calculus Supporting Structured Proofs. *Journal for Information Processing and Cybernetics (EIK)*, 30:261–276, 1994.
15. Bernd I. Dahn and Andreas Wolf. Natural Language Representation and Combination of Automatically Generated Proofs. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 175–192. Kluwer Academic Publishers, March 1996.
16. A. Edgar and F.J. Pelletier. Natural Language Explanations of Natural Deduction Proofs. In *Proceedings of the First Pacific Rim Conference on Computational Linguistics*, pages 269–278, Vancouver, 1993.
17. Uwe Egli and Stephan Schmitt. Intuitionistic Proof Transformations and Their Application to Constructive Program Synthesis. In Jaques Calmet and Jan Plaza, editors, *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)*, volume 1476 of *Lecture Notes in Artificial Intelligence*, pages 132–144, Berlin, 1998. Springer-Verlag.
18. Uwe Egly and Stephan Schmitt. On Intuitionistic Proof Transformations, their Complexity, and Application to Constructive Program Synthesis. *Fundamenta Informaticae*, 39:59–83, 1999.
19. Detlef Fehrer and Helmut Horacek. Exploiting the Addressee's Inferential Capabilities in Presenting Mathematical Proofs. In Pollack [54], pages 959–964.
20. Armin Fiedler. Using a Cognitive Architecture to Plan Dialogs for the Adaptive Explanation of Proofs. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 358–363, Stockholm, SWEDEN, 1999. Morgan Kaufmann.
21. Armin Fiedler. Dialog-driven Adaptation of Explanations of Proofs. In Bernhard Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.
22. Armin Fiedler. P.rex: An Interactive Proof Explainer. In Goré et al. [26], pages 416–420.
23. Armin Fiedler. *User-Adaptive Proof Explanation*. PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, 2001.
24. G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210,405–431, 1935. Translated in [25].

25. G. Gentzen. Investigations into Logical Deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., Amsterdam, 1969.
26. Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors. *Automated Reasoning, First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Artificial Intelligence*, Siena, Italy, 2001. Springer-Verlag.
27. Eric M. Hammer. *Logic and Visual Information*. CSLI Publications & FoLLI, Stanford, California, 1995.
28. Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie, Classe III Sciences Mathematiques et Physiques*, 33, 1930. Translated in [29].
29. Jacques Herbrand. *Logical Writings*. Harvard University Press, 1971. Edited by Warren D. Goldfarb.
30. Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of High-Level Formal Proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and Eleventh Innovative Application of Artificial Intelligence Conference (IAAI-99)*, pages 277–284. AAAI Press, 1999.
31. Helmut Horacek. A Model for Adapting Explanations to the User's Likely Inferences. *User Modeling and User-Adapted Interaction*, 7:1–55, 1997.
32. Helmut Horacek. Presenting Proofs in a Human-Oriented Way. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 142–156, Trento, Italy, 1999. Springer-Verlag.
33. Xiaorong Huang. Proof Transformation Towards Human Reasoning Style. In D. Metzing, editor, *Proceedings of the 13th German Workshop on Artificial Intelligence*, Informatik-Fachberichte 216, pages 37–42. Springer-Verlag, 1989.
34. Xiaorong Huang. Reference Choices in Mathematical Proofs. In Luigia Carlucci Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 720–725. Pitman Publishing, 1990.
35. Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
36. Xiaorong Huang. Proverb: A System Explaining Machine-Found Proofs. In Ashwin Ram and Kurt Eiselt, editors, *Proceedings of Sixteenth Annual Conference of the Cognitive Science Society*, pages 427–432, Atlanta, USA, 1994. Lawrence Erlbaum Associates.
37. Xiaorong Huang. Reconstructing Proofs at the Assertion Level. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 738–752, Nancy, France, 1994. Springer-Verlag.
38. Xiaorong Huang. Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level. In Norman Foo and Randy Goebel, editors, *Proceedings of the Fourth Rim International Conference on Artificial Intelligence (PRICAI-96)*, volume 1114 of *Lecture Notes in Artificial Intelligence*, pages 399–410, Berlin, 1996. Springer-Verlag.
39. Xiaorong Huang and Armin Fiedler. Presenting Machine-Found Proofs. In McRobbie and Slaney [46], pages 221–225.
40. Xiaorong Huang and Armin Fiedler. Proof Verbalization as an Application of NLG. In Pollack [54], pages 965–971.

41. Christoph Kreitz and Stephan Schmitt. A Uniform Procedure for Converting Matrix Proofs into Sequent-Style Systems. *Information and Computation*, 162(1–2):226–254, 2000.
42. Christoph Lingenfelder. Structuring Computer Generated Proofs. In N.S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 378–383, Detroit, Michigan, USA, 1989. IJCAI, Morgan Kaufmann.
43. Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. PhD thesis, University of Kaiserslautern, 1990. 115 pp.
44. David Mcallester, editor. *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, Pittsburgh, PA, USA, 2000. Springer-Verlag.
45. William McCune, editor. *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, Townsville, North Queensland, Australia, 1997. Springer-Verlag.
46. M.A. McRobbie and J.K. Slaney, editors. *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, New Brunswick, NJ, USA, 1996. Springer-Verlag.
47. Andreas Meier. System Description: TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In Mcallester [44], pages 460–464.
48. Dale A. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, 1983. 81 pp.
49. Dale A. Miller. Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs. In Shostak [61], pages 375–393.
50. Dale A. Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.
51. Frank Pfenning. Analytic and Non-Analytic Proofs. In Shostak [61], pages 394–413.
52. Frank Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, 1987. 156 pp.
53. William Pierce. Toward Mechanical Methods for Streamlining Proofs. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 351–365, Kaiserslautern, Germany, 1990. Springer-Verlag.
54. Martha E. Pollack, editor. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, JAPAN, 1997. Morgan Kaufmann.
55. Stephan Schmitt. A Tableau-Like Representation Framework for Efficient Proof Reconstruction. In Roy Dyckhoff, editor, *Theorem Proving with Analytic Tableaux and Related Methods. (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 398–414, St Andrews, Scotland, UK, July 2000. Springer-Verlag.
56. Stephan Schmitt and Christoph Kreitz. On Transforming Intuitionistic Matrix Proofs into Standard-Sequent Proofs. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods. 4th International Workshop. (TABLEAUX '95)*, volume 918 of *Lecture Notes in Artificial Intelligence*, pages 106–121, Schloß Rheinfels, St. Goar, Germany, May 1995. Springer-Verlag.
57. Stephan Schmitt and Christoph Kreitz. Converting Non-Classical Matrix proofs into Sequent-Style Systems. In McRobbie and Slaney [46], pages 418–432.

58. Stephan Schmitt and Christoph Kreitz. Deleting Redundancy in Proof Reconstruction. In Harrie de Swart, editor, *Theorem Proving with Analytic Tableaux and Related Methods. (TABLEAUX '98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 262–276, Oisterwijk, The Netherlands, May 1998. Springer-Verlag.
59. Stephan Schmitt, Lori Lorigo, Christoph Kreitz, and Alexey Nogin. JProver: Integrating Connection-based Theorem Proving into Interactive Proof Assistants. In Goré et al. [26], pages 421–426.
60. Sun-Joo Shin. *The Logical Status of Diagrams*. Cambridge University Press, 1994.
61. R. E. Shostak, editor. *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, Napa, California, USA, 1984. Springer-Verlag.
62. Andreas Wolf. Optimization and Translation of Tableau-Proofs into Resolution. *Journal of Information Processing and Cybernetics (EIK)*, 30(5-6):311–325, 1994. Akademie Verlag Berlin.
63. Andreas Wolf. A Translation of Model Elimination Proofs into a Structured Natural Deduction. In Douglas D. Dankel II, editor, *Proc. of 10th Int. Florida AI Research Society Conference*, pages 11–15, Daytona Beach, FL, USA, 1997. Florida AI Research Society.
64. Andreas Wolf. A Step Towards a Better Understanding of Automatically Generated Model Elimination Proofs. In José Cuena, editor, *Information Technologies and Knowledge Systems (IT&KNOWS'98) – Proceedings of the XV. IFIP World Computer Congress*, pages 415–428. Österreichische Computergesellschaft/International Federation for Information Processing, 1998.
65. Andreas Wolf and Johann Schumann. ILF-SETHEO: Processing Model Elimination Proof for Natural Language Output. In McCune [45], pages 61–64.

Rewrite and Decision Procedure Laboratory: Combining Rewriting, Satisfiability Checking, and Lemma Speculation

Alessandro Armando[1], Luca Compagna[1], and Silvio Ranise[2,⋆]

[1] DIST – Università degli Studi di Genova, Viale Causa 13 – 16145 Genova, Italia
[2] LORIA & INRIA – Université Henri Poincaré-Nancy 2,
615, rue du Jardin Botanique, BP 101, 54602 Villers les Nancy Cedex, France

1 Introduction

The lack of automated support is probably the main obstacle to the application of formal method techniques in the industrial setting. A possible solution to this problem is to combine the expressiveness of general purpose reasoners (such as theorem provers) with the efficiency of specialized ones (such as decision procedures). This is witnessed by the fact that many theorem provers developed for verification purposes (such as Acl2 [11], PVS [17], Simplify [9], STeP [14], and Tecton [10]) have integrated procedures for ubiquitous theories such as the theory of equality, decidable fragments of arithmetics, lists, and arrays. Unfortunately, designing an effective integration is far from being a trivial task and the solutions available in the verification systems listed above are not completely satisfactory for two main reasons. First, the schemes designed to incorporate decision procedure in larger systems are not flexible enough to allow developers to easily incorporate new procedures. Second, only a tiny portion of the proof obligations arising in many practical applications falls exactly into the domain the specialized reasoners are designed to solve. Thus, in many cases, available decision procedures are of little help if they are not combined with mechanisms for widening their scope.

1.1 RDL

RDL [1] is the acronym for **R**ewrite and **D**ecision procedure **L**aboratory. It provides an extension of rewriting to a powerful simplification mechanism exploiting satisfiability procedures for conjunctions of literals (i.e. reasoners specialized to solve the satisfiability problems for certain theories). The interplay between the satisfiability procedure and the other modules of **RDL** is parametric in the theory in which the procedure works. As a consequence, the reasoning activity implemented by the system can be easily extended by plugging-in new satisfiability

⋆ The last author is also partially supported by Università degli Studi di Genova under the program "Finanziamenti a progetti di singoli e/o giovani ricercatori (D.R. 226 del 25.10.2000)".

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 30–45, 2005.

procedures. In turn, incorporated satisfiability procedures can be extended to handle larger classes of formulas by instantiating a generic lemma speculation mechanism. This mechanism is parametric in the satisfiability procedure being extended.

Our main motivation to develop **RDL** is to experiment with different combinations of rewriting and possibly extended satisfiability procedures. The focus of the system is on simplification of quantifier-free clauses and corresponding applications in the area of verification supported by automated theorem proving.

RDL features the following characteristics.

1. It is based on Constraint Contextual Rewriting (CCR) [2, 3]. CCR is a powerful simplification mechanism in which contextual rewriting [20] is complemented by a specialized reasoner, a procedure capable of establishing formula satisfiability w.r.t. a fixed theory of interest. The key feature of CCR is that the context of rewriting (i.e. the literals assumed true during rewriting) are manipulated and checked for consistency by the satisfiability procedure.
2. **RDL** is an open system which can be modularly extended with new satisfiability procedures provided these offer certain interface functionalities. As underlying theories currently available there are the universal theory of equality (UTE), the universal theory of linear arithmetic over integers (ULAI), and the theory obtained as the combination of the previous two (UTELAI).
3. **RDL** implements instances of a *generic extension schema* for decision procedures [4]. The key ingredient of such a schema is a *lemma speculation mechanism* which 'reduces' the satisfiability problem of a given theory to the satisfiability problem of one of its sub-theory for which a satisfiability procedure is already available. The current version of the system provides implementations of this schema which enable the satisfiability procedure for ULAI to handle properties of user-defined functions as well as a fragment of arithmetic with multiplication.

RDL is implemented in (SICStus) Prolog and it is freely available via the *Constraint Contextual Rewriting Project* home page at

http://www.mrg.dist.unige.it/ccr

1.2 Related Systems

In ACL2 [11] (as in its predecessor, NQTHM [6]), a sophisticated schema to incorporate a decision procedure for linear arithmetic in the simplification activity is implemented. Unfortunately, the design of such a schema heavily depends on the particular characteristics of the host system [7]. As a result, it is not easy to incorporate new decision procedures. Both *Simplify* [9] and PVS [17] feature a bunch of cooperating decision procedures, following the paradigm proposed in [16] and in [19], respectively. In both systems, while it is easy to plug-in new procedures, an insufficient degree of automatization is provided for some classes of proof obligations which frequently arise in practical verification efforts such as

some sub-theory of the theory of arithmetic with multiplication. In both systems, the user is forced to supply appropriate lemmas encoding the properties of the interpreted functions (e.g. multiplication). The version of STeP described in [5] implements a rational based version of the Fourier-Motzkin method, extended to handle multiplication by (partial) quantifier elimination and reasoning about the sign of multiplicands. Although STeP offers a high degree of automation for a significant sub-theory of arithmetic with multiplication, it is not flexible enough to provide similar degrees of automation for other theories.

Plan of the Paper

In Section 2, we describe how **RDL** solves a typical verification condition arising in the proof of termination of a function normalizing expressions. This serves the twofold purpose of introducing the concept of (theorem proving) problem solved by **RDL** and of giving a brief overview of the main reasoning activities implemented in the system. In Section 3, we describe how to specify a theorem proving problem to **RDL**. Then, in Section 3, we describe the reasoning activities implemented in the system and their interplay. Finally, in Section 5, we report an excerpt of the experimental results of the system on some typical problems and we compare **RDL** with other state-of-the-art validity checkers.

2 An Example

Consider the problem of showing the termination of the function to normalize conditional expressions in propositional logic as described in [6].

The expressions of the logic are built over propositional constants (denoted in the following with $\mathsf{pl}(N)$, where N is an integer) and the ternary connective "if A then B else C" (denoted in the following with $\mathsf{if}(A, B, C)$, where A, B, and C are variables ranging over the set of expressions of propositional logic). Informally, the function *norm* for normalizing conditional expressions (recursively) remove all the if's occurring as the first argument of another if by pushing them into the other two arguments of the external if.

The argument in the proof of termination of *norm* is based on exhibiting a measure function that decreases (according to a given well-founded ordering) at each function's recursive call. For example, ms (reported in [18]) is one such a function:

$$ms(\mathsf{pl}(N)) = 1$$
$$ms(\mathsf{if}(A, B, C)) = ms(A) + ms(A) * ms(B) + ms(A) * ms(C)$$

where $+$ and $*$ denote addition and multiplication over integers. It is easy to check that ms enjoys the following property:

$$ms(A) > 0 \tag{1}$$

for each expression A of the logic. The definition of ms and property (1) are stated in **RDL** by asserting the following Prolog facts:

```
fact(bm,msbase,[],ms(pl(N))=1).
fact(bm,msstep,[],ms(if(A,B,C))=ms(A)+ms(A)*ms(B)+ms(A)*ms(C)).
fact(bm,msfact,[],ms(A)>0).
```

where `msbase`, `msstep`, and `msfact` are the unique identifiers of the facts in the system and `[]` indicates that the facts are unconditional.

One of the proof obligation expressing the decrease argument is

$$ms(\mathsf{if}(u, \mathsf{if}(v, y, z), \mathsf{if}(w, y, z))) < ms(\mathsf{if}(\mathsf{if}(u, v, w), y, z)), \tag{2}$$

where $<$ denotes the 'less-than' relation over integers and u, v, w, y, and z are expressions of the logic. In **RDL** we can specify the clause (in this case a unit clause) to be checked for validity as follows:

```
input(bm,
     [ms(if(u, if(v,y,z), if(w,y,z))) < ms(if(if(u,v,w), y, z))]).
```

There are still two missing ingredients to complete the specification of our problem to **RDL**. First, we need to provide an informal description of the problem under consideration:

```
description(bm,
'Silvio Ranise',
'Problem taken from the paper
  "Proving Termination of Normalization Functions
   for Conditional Expressions"
 by L C Paulson.').
```

Second, we need to tell **RDL** what satisfiability procedure to use in order to check the validity of the formula:

```
expected_output(bm, aug_aff(eq_la), rpo, [true]).
```

where `aug_aff(eq_la)` tells **RDL** to use the satisfiability procedure for UTELAI extended with lemma speculation techniques which allow to use the definition of ms and its property as well as some properties about multiplication; `rpo` is the recursive path ordering that is going to be used by the system for rewriting[1].

Now, we are in the position to run the system on the specified problem by simply typing

```
run(bm).
```

RDL's output is reported in Figure 1[2]. Lines 1–8 provide the user with a brief summary of the problem that **RDL** is going to simplify. Line 9 gives the formula obtained by the simplification process implemented by the system. Line 22 shows

[1] To simplify the presentation, we omit the precedence over function symbols needed to completely specify the ordering and, in the following, we assume that such a precedence has been defined so that the rewriting steps described below are possible.

[2] The original output of the system has been slightly edited in order to simplify the discussion that follows.

```
 1 Problem: bm
 2 Reasoning Specialist: combination of the theory of ground
 3                       equality and Linear Arithmetic with
 4                       a combination of augmentation and
 5                       affinization enabled.
 6 Ordering: Recursive Path Ordering.
 7 Input Formula: [ms(if(u,if(v,y,z),if(w,y,z)))<ms(if(if(u,v,w),y,z))]
 8 Expected Formula: [true]
 9 Simplified Formula: [true]
10 Status: ok!
11 Reduction:
12 cl_simp:
13   [id,
14     crew>(crew>(crew>(crew>(crew>(crew>(crew>
15     (normal>
16      cxt_entails_true:[
17       augment_affinize:
18        [crew:[
19         augment_affinize,
20          cs_extend]>
21           (augment_affinize>augment_affinize)]]]]]]]]]
22 Time (Elapsed-Theorem Proving): 610-600 msec
```

Fig. 1. Sample output of **RDL**.

the time used by the system to perform the simplification (600 msec) and the total time (610 msec), i.e. the time to perform simplification as well as the other instructions such as input-output. Lines 11 to 21 describe the simplification steps undertook by the system. First of all, **RDL** initializes the simplification of the clause (2) (`cl_simp` at 12). This consists of building up the context of simplification and of selecting a literal to be simplified (`id` at 13); in this case, the simplification context is empty and the literal being simplified is the only literal in the clause. Then, the simplification process can start.

RDL rewrites the l.h.s. and the r.h.s. of (2) with the definition of ms (the sequence of `crew` at 14) and it obtains the following literal:

$$\begin{array}{l} ms(u) + ms(u) * ms(v) + ms(u) * ms(v) * ms(y) + \\ ms(u) * ms(v) * ms(z) + ms(u) * ms(w) + \\ ms(u) * ms(w) * ms(y) + ms(u) * ms(w) * ms(z) < \\ ms(u) + ms(u) * ms(v) + ms(u) * ms(w) + \\ ms(u) * ms(y) + ms(u) * ms(v) * ms(y) + ms(u) * ms(w) * ms(y) + \\ ms(u) * ms(z) + ms(u) * ms(v) * ms(z) + ms(u) * ms(w) * ms(z) \end{array} \tag{3}$$

RDL then performs all the possible cancellations in (3) (`normal` at line 15) and it obtains:

$$ms(u) * ms(y) + ms(u) * ms(z) < 0. \tag{4}$$

In order to prove the validity of (2), **RDL** checks the unsatisfiability of its negation (`cxt_entails_true` at line 16). To do this, it factorizes (4) to $ms(u) * (ms(y) + ms(z)) < 0$ and then it considers the following instance of a trivial property about the sign of multiplicands:

$$(ms(u) > 0 \wedge ms(y) + ms(z) > 0) \Longrightarrow ms(u) * (ms(y) + ms(z)) > 0 \qquad (5)$$

(this is identified by `augment_affinize` at line 17). In order to make the conclusion of (5) available to the system (`cs_extend` at line 20), it is necessary to relieve its hypotheses (`crew` at line 19). This is easy since **RDL** readily instantiates (1) three times, namely to $ms(u) > 0$, $ms(y) > 0$, and to $ms(z) > 0$ (the three `augment_affinize` at lines 19–21). At this point, it is trivial to detect the unsatisfiability of (4) and the conclusion of (5).

3 Specifying a Problem to RDL

The basic concept underlying **RDL**'s user interface is that of *problem*. Intuitively, a problem provides a specification of the clause to be simplified as well as the satisfiability procedure to be used during simplification and the facts that the user wants to assume valid. Formally, a problem determines a (first-order) language and a (first-order) theory. In particular, a problem specifies which predicate and function symbols are interpreted since either they are known to the satisfiability procedures or they are taken into account by the lemma speculation mechanism. The specification of a problem in **RDL** involves a number of information that must be specified by asserting certain facts (in Prolog parlance) to the system.

`description(TagPb, Author, Descr).` The first argument `TagPb` is the unique label of the problem the user wishes to solve. In **RDL**, each problem must be uniquely identified by a Prolog term, e.g. the Prolog constant `bm` in Figure 1. The other two arguments of the predicate are Prolog strings. In particular, `Author` specifies the name of the author of the problem and `Descr` gives an informal description of the problem.

`input(TagPb, Clause).` The first argument `TagPb` is the unique identifier of the problem. The second argument `Clause` specifies the (ground) clause to be simplified. **RDL** represents clauses as Prolog lists of literals. First-order literals are represented by ground Prolog literals.

`fact(TagPb, TagFact, Conds, Concl).` The first argument `TagPb` is the unique label of the problem. The second argument `TagFact` is the unique label of the fact within the name space of the problem. The last two arguments specify the hypotheses and the conclusion of a conditional fact. In particular, `Conds` is a list of literals and `Concl` is a single literal. In this case, **RDL** represents first order literals as Prolog literals. In particular, first order variables are represented by Prolog variables which can occur only in this position of the specification of a problem. As an example, consider the following Prolog fact:

```
fact(pb,label,[g(X)>0, f(Y,c)=g(Z)],h(X,Y)=Z).
```

It encodes the following formula:

$$\forall\, x\, y\, z\, ((g(x) > 0 \wedge f(y, c) = g(z)) \Longrightarrow h(x, y) = z),$$

where x, y, and z are variables (represented by the Prolog variables `X`, `Y`, and `Z` respectively), c is a constant, g is a unary function symbol, f and h are binary function symbol, $=$ is the equality symbol, and $>$ is an infix binary predicate. The formula specified by `fact` is assumed valid during the simplification activity.

`expected_output(TagPb, RS, Ord, Clause)`. The first argument `TagPb` is the unique label of the problem. The second argument `RS` specify the (possibly extended) satisfiability procedure to be used in order to support the simplification activity. In the actual implementation of **RDL**, `RS` can be one of the following Prolog term:

- `eq` identifies a satisfiability procedure for UTE. The implementation of this procedure is based on the congruence closure algorithm described in [19];
- `la` identifies a satisfiability procedure for ULAI based on the version of Fourier-Motzkin algorithm described in [7];
- `eq_la` identifies a combination of the above two satisfiability procedures based on Nelson and Oppen's combination paradigm [16];
- `aug(SatProc)` (where `SatProc` is either `eq`, `la`, or `eq_la`) identifies the extension of the satisfiability procedure `SatProc` by means of the augmentation mechanism [2, 3], i.e. the capability of making available to the satisfiability procedure selected instances of available lemmas (specified by `fact`);
- `aff(SatProc)` (where `SatProc` is either `la` or `eq_la`) identifies the extension of the satisfiability procedure `SatProc` by means of the affinization mechanism [4], i.e. the capability of making available selected properties of multiplication;
- `aug_aff(SatProc)` (where `SatProc` is either `la` or `eq_la`) identifies a combination of the two extension mechanisms outlined above [4].

We notice that each satisfiability (implicitly) declare a set of interpreted predicate symbols by means of the predicate `pred_sym(TagPb, PredSpec)`. The first argument `TagPb` is the unique identifier of the problem. The second argument `PredSpec` is a Prolog term of the form `p(_, _, ..., _)`, where `p` is a predicate symbol together with its arity, namely `(_, _, ..., _)`. For example, the predicate $<$ is automatically declared as interpreted by asserting `pred_sym(_, _<_).`. This is done each time the satisfiability procedure for ULAI is used during the simplification process, i.e. the constant `la` is given as the second argument of `expected_output`.

The third argument `Ord` is the ordering to be used while rewriting. Two possible ordering can be used in **RDL**: the Knuth-Bendix ordering [12] (identified by the Prolog constant `kbo`) and the recursive-path ordering [8] (identified by the Prolog constant `rpo`). For `kbo`, we need to specify the weight of the symbol by means of the predicate `symbol_weight(TagPb, FSym, N)`, where `TagPb` is the unique label of the problem, `Fsym` is a function (or predicate) symbol, and `N` is a positive natural number (i.e. the weight of the symbol). Furthermore, for both ordering we can specify a precedence relation over function (and predicate)

symbols by means of the predicate `ord_gt(TagPb,F1,F2)`, where `TagPb` is the unique label of the problem, `F1` and `F2` are function (or predicate) symbols s.t. `F1` is bigger than `F2` in the precedence relation used to define either `kbo` or `rpo`.

Finally, the last argument `Clause` of `expected_output` specifies the clause which the user thinks **RDL** is going to produce as the result of the simplification activity. This last parameter is not strictly required (it can be left unspecified by using a Prolog variable) and it is mainly used for validating the corpus of problems shipped with the system.

4 The Reasoning Activities of RDL

RDL features a tight integration of three reasoning activities: *contextual rewriting*, *satisfiability checking*, and *lemma speculation*. The sophisticated interactions between these reasoning activities are the key ingredients of the effectiveness of **RDL**'s simplification mechanism. In the following, the various functionalities will be modeled by means of relations whereas the interplay between the various functionalities is specified by an inductive definition by using a set of inference rules.

4.1 Contextual Rewriting

In **RDL**, the rewriter implements (a variant of) contextual rewriting [20]. It manipulates two data structures. The former is the set of literals which can be assumed true during the rewriting activity; the conjunction of these literals is called the (*rewriting*) *context*. The second data structure is a set of conditional rules which are added to **RDL**'s database of valid facts by asserting Prolog facts of the form

```
fact(pb, name, [h1, ..., hn], l=r).
```

where `h1`, ..., `hn`, and `l=r` are **RDL**'s representation of some first-order literals, which are denoted in the following with $h_1, ..., h_n$, and $l = r$ (respectively).

Now, we are in the position to give a high-level description of the rewriting algorithm implemented in **RDL**. In the following, we assume that $\prec$ is a well-founded ordering on ground terms which allows for a suitable mechanization in **RDL**.

Without loss of generality, assume $r\sigma \prec l\sigma$ for a ground substitution σ. Otherwise, swap l with r (if $l\sigma$ is different from $r\sigma$). Given a literal $p[l\sigma]$, **RDL**'s rewriter returns $p[r\sigma]$ if the following two conditions are satisfied: (i) the ground literals $h_1\sigma, ..., h_n\sigma$, and $p[r\sigma]$ are smaller than $p[l\sigma]$ according to $\prec$; and (ii) each $h_i\sigma$ (for $i = 1, ..., n$) can be simplified to *true* by recursively invoking the activity of contextual rewriting. Checking for the entailment of an instantiated condition can be done in three ways. The first is to recursively invoke the **RDL**'s rewriter on the literal under consideration with the aim of rewriting it to *true*. This informal description can be formalized by means of the following inference rule named `crew`:

$$\frac{C :: h_1\sigma \underset{\mathsf{ccr}}{\longrightarrow} true \;\; \cdots \;\; C :: h_n\sigma \underset{\mathsf{ccr}}{\longrightarrow} true}{C :: p[l\sigma]_u \underset{\mathsf{ccr}}{\longrightarrow} p[r\sigma]_u}$$

where $h_1 \wedge \cdots \wedge h_n \implies l = r$ is one of the available (possibly conditional) rewrite rules and $C :: p \underset{\mathsf{ccr}}{\longrightarrow} p'$ is a sequent which denotes the mechanization of the one-step contextual rewriting relation, i.e. it takes a literal p in context C and returns the literal p'.

4.2 Satisfiability Checking

In **RDL**, a satisfiability procedure for a given (first-order) theory T_c works on a data structure (called *constraint store*) representing a conjunction of ground literals to be assumed true during **RDL**'s simplification activity. The constraint store is built by interning the literals in the rewriting context. As it will be discussed below, the particular data structure used to implement the constraint store depends on the theory T_c.

There are four functionalities manipulating the constraint store: $\texttt{cs\_init}(C)$, $\texttt{cs\_unsat}(C)$, $P :: C \underset{\mathsf{cs\_extend}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} C'$, and $C :: p \underset{\mathsf{cs\_normal}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} p'$, where C and C' are constraint stores, p and p' are ground literals, and P is a set of ground literals.

The functionalities `cs_init` and `cs_unsat` manipulate a constraint store and are invoked by **RDL**'s rewriter so to synchronize the content of the rewriting context and of the constraint store. More precisely, $\texttt{cs\_init}(C)$ sets C to the "empty" constraint store and $\texttt{cs\_unsat}(C)$ is a boolean function recognizing a subset of unsatisfiable (in T_c) constraint stores whose unsatisfiability can be checked by means of a computationally inexpensive (syntactic) check.

The remaining two interface functionalities (i.e. $P :: C \underset{\mathsf{cs\_extend}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} C'$ and $C :: p \underset{\mathsf{cs\_normal}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} p'$) must satisfy some requirements in order to allow the "plug-and-play" incorporation of new satisfiability procedures in the system. As said above, the constraint store is the result of interning the literals in the rewriting context by invoking the functionality

$$P :: C \underset{\mathsf{cs\_extend}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} C'$$

which denotes the computation performed in order to intern the literals in the input set P, (possibly) deriving new literals entailed by the conjunction of the literals in P and C, and adding them to the actual state C of the procedure so to obtain the new state C'. The last functionality

$$C :: p \underset{\mathsf{cs\_normal}}{\longrightarrow\!\!\!\!\!\!\longrightarrow} p'$$

provided by the satisfiability procedure computes a normal representation p' of a given literal p w.r.t. the theory T_c and the literals stored in the constraint store C. In order to simplify the integration with the **RDL**'s rewriter, we require that (*i*) the literal p' returned by this functionality is entailed by T_c and the conjunction of literals in C, and that (*ii*) p' is smaller (according to the rewriting ordering $\prec$) than p.

Example 1. The satisfiability procedure for ULAI in RDL. Consider the first-order language consisting of the numerals $..., -2, -1, 0, 1, 2, ...$, variables, the function symbol $+$, the (infix) binary predicate symbols $<$, $\leq$, $=$, $\geq$, and $>$, and the usual logical connectives. The intended structure of this language (whose theory is ULAI) interprets numerals as integers[3], variables range over integers, $+$ is interpreted as addition, $<$, $\leq$, $\geq$, and $>$ are interpreted as the usual ordering relations, and $=$ is interpreted as the identity relation.

Let T_c be the first-order theory containing ULAI and n-ary function symbols (other than $+$) interpreted as arbitrary functions from n-tuples of integers to integers.

The Fourier-Motzkin elimination method [13] is based on the idea of eliminating one variable at a time in the hope of obtaining a 'trivially' unsatisfiable inequality such as, e.g., $0 \leq -1$. It can be straightforwardly adapted to obtain a proof procedure for T_c.

Although the exponential worst-case complexity seems to discourage its usage for checking the unsatisfiability of conjunctions of inequalities, the Fourier-Motzkin method can be made usable in practice (as observed e.g. in [7]) by using the simple trick of choosing the variable to eliminate according to a given ordering.

We assume that $<$, $=$, $\geq$, and $>$ (in the language of ULAI) are preliminary eliminated in favor of $\leq$ (e.g. $x < 0$ can be rewritten to $x \leq -1$ by exploiting the integral property of integers). The inequalities in the constraint store are put into the following (normal) form

$$c_1 \cdot m_1 + \cdots + c_n \cdot m_n \leq c \tag{6}$$

where $n \geq 0$ (if $n = 0$, then (6) stands for $0 \leq c$), $c, c_1, ..., c_n$ are relatively prime integers (called *coefficient*s), $m_1, ..., m_n$ are (first-order) terms (called *multiplicand*s) whose top-most function symbols are different from $+$ s.t. $m_{i+1} \prec m_i$ (where $\prec$ is the ordering used for rewriting which is required to be total over ground terms), and $c_i \cdot m_i$ $(i = 1, ..., n)$ abbreviates the term $m_i + \cdots + m_i$ in which m_i occurs c_i times.

A constraint store is a data base of inequalities of the form (6) indexed by the key multiplicands. More precisely, each key multiplicand points to two lists of inequalities: one contains inequalities where the coefficient of the key multiplicand is positive whereas the other contains inequalities where the coefficient of the key multiplicand is negative. If we derive an inequality of the form $0 \leq c$, where c is a negative integer, we stop the exhaustive elimination of variables and we set a flag signaling the unsatisfiability of the constraint store. The functionality `cs_init`(C) is defined so to set up the empty data base of inequalities and `cs_unsat`(C) returns true when the flag of the unsatisfiability of the constraint store is true.

Let ι and ι' be two inequalities of the form (6) both having m as their heaviest multiplicand, k (k') be the coefficient of m in ι (ι', resp.), k and k' be of opposite

[3] In the following, to simplify the discussion, we will use the term 'integer' in place of 'numeral'.

sign, and $elim(\iota, \iota')$ be the normal form of the linear combination of ι and ι' not containing m. Now, we are in the position to describe an implementation of the functionality $P :: C \xrightarrow[\text{cs\_extend}]{} C'$. First of all, put the literals of P into inequalities of the form (6) and insert them into C at appropriate positions. Then, close the resulting data base under the operation $elim$ so to obtain C', i.e. C' is that for any ι_1, ι_2 in C' we have that $elim(\iota_1, \iota_2)$ is in C' (if $elim$ is defined).

Finally, we notice that it is possible to extend the Fourier-Motzkin algorithm in order to derive equalities entailed by inequalities [13] (stored in the constraint store). A set of entailed equalities is created as soon as an inequality of the form $0 \leq 0$ is derived; all the inequalities which contributed to create this inequality are turned into equalities (see, e.g. [13] for more details). It is natural to exploit these equalities to simplify the literal which is currently rewritten. This observation offers an immediate implementation of $C :: p \xrightarrow[\text{cs\_norm}]{} p'$. In fact, if we extend the data base C of inequalities to store also the entailed equalities, we can use them as rewrite rules since we are always capable of orienting them; the entailed equalities are ground and the ordering $\prec$ is assumed to be total on ground terms.

Another example of implementation of $C :: p \xrightarrow[\text{cs\_norm}]{} p'$ is given by the algebraic manipulation required to perform the cancellation in (3) to derive (4) in Section 2.

4.3 Combining Rewriting and Satisfiability Checking

Now, we are in the position to describe how the functionalities provided by the satisfiability procedure are exploited by the rewriting activity of **RDL**.

A given literal can be rewritten to *true* in a given context if it is entailed by the context. In turn, the check for entailment of a literal l by a conjunction of literals C can be performed by checking the unsatisfiability of $C \wedge \neg l$. In **RDL**, this can be easily done by invoking `cs_unsat` on the constraint store obtained by adding the negation of the literal being rewritten. Similarly, we can rewrite to $false$ a literal if its negation is entailed by the context. This kind of reasoning can be formalized by the following two inference rules, named `cxt_entails_true` and `cxt_entails_false` (read from left to right):

$$\frac{\{\neg p\} :: C \xrightarrow[\text{cs\_extend}]{} C'}{C :: p \xrightarrow[\text{ccr}]{} true} \text{ if } \texttt{cs\_unsat}(C') \qquad \frac{\{p\} :: C \xrightarrow[\text{cs\_extend}]{} C'}{C :: p \xrightarrow[\text{ccr}]{} false} \text{ if } \texttt{cs\_unsat}(C')$$

In **RDL**, there is another mechanism of rewriting realized by simply invoking the functionality for normalization provided by the satisfiability procedure. This can be formalized by the following inference rule, named `normal`:

$$\frac{C :: p \xrightarrow[\text{cs\_normal}]{} p'}{C :: p \xrightarrow[\text{ccr}]{} p'}$$

We notice that the inference rules `cxt_entails_true`, `cxt_entails_false`, and `normal` extends the definition of the relation $C :: p \xrightarrow[\text{ccr}]{} p'$ modeling the activity of contextual rewriting as introduced in Section 4.1.

4.4 Lemma Speculation

Three instances of the lemma speculation mechanism described in [4] are implemented in **RDL**. All the instances share the goal of feeding the satisfiability procedure with new facts about function symbols which are otherwise uninterpreted in the theory in which the satisfiability procedure works. More precisely, they inspect the context C and return a set of ground facts entailed by C. The lemma speculation activity of computing a set S of ground facts given a constraint store C can be modeled by the following relation:

$$C \mapsto \langle C', S \rangle,$$

where C' is a constraint store which differs from C in the fact that some literals are marked as already used (this is useful to avoid infinite looping by reconsidering infinitely often the same literals for deriving new facts).

Augmentation. It extends the information available to the satisfiability procedure with selected instances of lemmas encoding properties of symbols the decision procedure does not know anything about. For example, by devising a suitable set of lemmas about multiplication, it is possible to enable a procedure for ULAI to handle formulas whose satisfiability depends on properties of multiplication (e.g. multiplying two positive integers we obtain a positive integer).

The crucial step for the success of augmentation is the selection of suitable instances of the available formulas. This is an instance of the more general problem of choosing suitable instances of lemmas for guiding a generic prover to a successful proof. Unfortunately, for such a problem no general satisfactory solution does exist. In **RDL**, for our particular instance, we implemented the heuristics of finding instances of the conclusions of the available conditional lemmas promoting further computations (e.g. further Fourier-Motzkin elimination steps in the case of the procedure for ULAI) when added to the current state of the satisfiability procedure.

A further problem is the presence of extra variables in the hypotheses (w.r.t. the conclusion) of lemmas. **RDL** avoids this problem by requiring that the conclusion contains all the variables occurring in the lemma and that all the variables get instantiated by matching the conclusion of the lemma against the largest (according to $\prec$) literal in C.

We notice that augmentation critically depends on the shape of the available lemmas and the algorithm implemented by the satisfiability procedure. If a suitable set of lemmas is defined, then augmentation dramatically widens the scope of a satisfiability procedure. Unfortunately, devising such a suitable set is a time consuming activity. This problem can be solved in some important special cases such as some fragments of arithmetics with multiplication.

Affinization. In the actual version of **RDL**, affinization implements the 'on-the-fly' generation of lemmas about multiplication over integers. We emphasize that the user is no more required to provide suitable lemmas about properties of multiplication since instances of some classes of properties are automatically generated.

To understand how affinization works, consider the non-linear inequality $x * y \leq -1$ (where x and y range over integers). By resorting to its geometrical interpretation, it is easy to verify that (over integers) $x * y \leq -1$ is equivalent to $(x \geq 1 \wedge y \leq -1) \vee (x \leq -1 \wedge y \geq 1)$. To avoid case splitting, we observe that the semi-planes represented by $x \geq 1$ and $x \leq -1$ as those represented by $y \leq -1$ and $y \geq 1$ are non-intersecting. This allows to derive the following four lemmas: $x \geq 1 \Longrightarrow y \leq -1$, $x \leq -1 \Longrightarrow y \geq 1$, $y \geq 1 \Longrightarrow x \leq -1$, and $y \leq -1 \Longrightarrow x \geq 1$. This process can be generalized to non-linear inequalities which can be put in the form $x * y \leq k$ (where k is an integer) by factorization [15]. The generated (conditional) lemmas are used as for augmentation.

A Combination of Augmentation and Affinization. On the one hand affinization can be seen as a significant improvement over augmentation since it does not require any user intervention. On the other hand it fails to apply when inequalities cannot be transformed into a form suitable for affinization. **RDL** combines augmentation and affinization by considering the function symbols occurring in the constraint store C, i.e. the top-most function symbol of the largest (according to $\prec$) literal in C triggers the invocation of either affinization or augmentation.

4.5 Combining Rewriting and Lemma Speculation

The main obstacle to using the facts resulting from the lemma speculation activity is that such facts are conditional. Hence, we should preliminary relieve their hypotheses in order to be entitled to make their conclusions available to the satisfiability procedure. In **RDL**, we solve this problem by rewriting each hypothesis to *true* (if possible) by invoking the rewriter. (Notice that this implies that the rewriter and the satisfiability procedure are mutually recursive.) The above reasoning activity can be formalized by the following inference rule, named either `augment`, `affinize`, or `augment_affinize` depending on which lemma speculation mechanism is specified:

$$\frac{C' :: g_1 \xrightarrow[\text{ccr}]{} true \quad \cdots \quad C' :: g_n \xrightarrow[\text{ccr}]{} true \quad \{c_1, ..., c_m\} :: C \xrightarrow[\text{cs\_extend}]{} C'}{P :: C \xrightarrow[\text{cs\_extend}]{} C'} \text{ if } C \mapsto \langle C', S \rangle$$

where $(g_1 \wedge \cdots \wedge g_n) \Longrightarrow (c_1 \wedge \cdots \wedge c_m)$ is in S and it is ground (for $n \geq 1$ and $m \geq 1$).

5 Experiments

RDL must be judged w.r.t. its effectiveness in simplifying (and possibly checking the validity of) proof obligations arising in practical verification efforts where decision procedures play a crucial role. Although standard benchmarks for theorem provers such as TPTP can be (partially) tackled by **RDL**, we prefer to evaluate **RDL**'s performances on proof obligations extracted from real verification efforts. To do this, we are building a corpus of proof obligations taken from the literature

Table 1. Experimental Results.

#	Problem	**RDL**
1	$f(A) = f(B) \Longrightarrow (r(g(A,B),A) = A) \vdash$ $r(g(y,z),x) = x \vee \neg(g(x,y) = g(y,z)) \vee \neg(y = x)$	26
2	$A * B = B * A, (\neg(C = 0)) \Longrightarrow (rem(C * D, C) = 0) \vdash$ $rem(y * z, x) = 0 \vee \neg(x * y = z * y) \vee x = 0$	109
3	$(A > 0) \Longrightarrow (rem(A * B, A) = 0) \vdash \ rem(x * y, x) = 0 \vee x \leq 0$	12
4	$min(A) \leq max(A) \vdash$ $\neg(k \geq 0) \vee \neg(l \geq 0) \vee \neg(l \leq min(b)) \vee \neg(0 < k) \vee l < max(b) + k$	12
5	$(memb(A,B)) \Longrightarrow (len(del(A,B)) < len(B)) \vdash$ $\neg(w \geq 0) \vee \neg(k \geq 0) \vee \neg(z \geq 0) \vee \neg(v \geq 0) \vee \neg(memb(z,b))$ $\vee\neg(w + len(b) \leq k) \vee w + len(del(z,b)) < k + v$	17
6	$(0 < A) \Longrightarrow (B \leq A * B), 0 < ms(C) \vdash$ $ms(c) + ms(d)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2ms(d)^2 * ms(b) + ms(d)^4$	72
7	$A \geq 4 \Longrightarrow (A^2 \leq 2^A) \vdash \neg(c \geq 4) \vee \neg(b \leq c^2) \vee \neg(2^c < b)$	14
8	$(max(A,B) = A) \Longrightarrow (min(A,B) = B), (p(C)) \Longrightarrow (f(C) \leq g(C)) \vdash$ $\neg(p(x)) \vee \neg(z \leq f(max(x,y))) \vee \neg(0 < min(x,y)) \vee \neg(x \leq max(x,y)) \vee$ $\neg(max(x,y) \leq x) \vee z < g(x) + y$	114
9	$0 < ms(C) \vdash$ $ms(c) + ms(d)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2ms(d)^2 * ms(b) + ms(d)^4$	63
10	$\vdash \ x \geq 0 \Longrightarrow x^2 - x + 1 \neq 0$	40

and from the examples available for similar systems. The problems in our corpus are representative of various verification scenarios and are considered difficult for current state-of-the-art verification systems.

Table 1 reports a selection of the results of our computer experiments. Problem lists the available lemmas[4] (if any) and the formula to be decided. $\vdash$ is the binary relation characterizing the deductive capability of **RDL** (we have that $\vdash$ is contained in $\models_T$, where T is the theory decided by the decision procedure extended with the available facts). The last column records the time (expressed in msec) used by **RDL** to solve a problem[5].

RDL solves problems 1 and 2 with the procedure for UTE. In the former, the procedure is used to derive equalities entailed by the context which are used as rewrite rules and enable the use of the available lemma. The ordered rewriting engine implemented by **RDL** is a key feature to successfully solve problem 2 since this form of rewriting allows to handle usually non-orientable rewrite rules such as $A * B = B * A$. **RDL** solves problem 3 with a satisfiability procedure for ULAI extended with augmentation. Infact, the available lemma is applied once its instantiated condition, namely $x > 0$, is relieved by the decision procedure (it is straightforward to check the inconsistency of $x > 0$ and the literal $x \leq 0$ in the

[4] Capitalized letters denote implicitly universally quantified variables.

[5] Benchmarks run on a 600 MHz Pentium III running Linux. **RDL** is implemented in Prolog and it was compiled using SICStus Prolog, version 3.8.

context). **RDL** solves problems 4, 5, 6, and 7 with a procedure for ULAI extended with the augmentation mechanism. In particular, the formula of problem 6 is a non-linear formula whose validity is successfully established by **RDL** in a similar way of the example in Section 2. **RDL** solves problem 8 with the combination of procedures for ULAI and UTE. **RDL** solves problems 9 and 10 with the procedure for ULAI, extended with both augmentation and affinization. The lemma about multiplication (i.e. $0 < I \Longrightarrow J \leq I * J$) is supplied in problem 6 but it is not in problem 9. Only the combination of augmentation and affinization can solve problem 9. Finally, problem 10 shows the importance of the context in which proof obligations are proved (since **RDL** does not case-split). In fact, without $x \geq 0$ `augment` and `affinize` would not be able to solve problem 10.

As a matter of fact, the online version of STeP fails to solve all of the problems reported in Table 1. However, most of them are successfully solved by the improved version of STeP described in [5]. This version solves problems 9 and 10 by resorting to a partial method for quantifier elimination (see [5] for details). Instead, our affinization mechanism is capable of proving the formula with simpler mathematical techniques. The comparison is somewhat difficult since the method used by STeP works over the rationals and our affinization technique only works over integers. *Simplify* successfully solves problems 1 to 8 thanks to a Nelson-Oppen combination of decision procedures and an incomplete matching algorithm which is capable of instantiating (valid) universally quantified clauses. However, it does not solve problems 9 and 10 since it is unable to handle non-linear facts without user-supplied lemmas (such as $0 < I \Longrightarrow J \leq I * J$ in problem 6). Finally, SVC fails to solve all the problems involving augmentation and affinization since it does not provide a mechanism to take into account facts which partially interpret user-defined function symbols.

References

1. A. Armando, L. Compagna, and S. Ranise. System Description: RDL—Rewrite and Decision procedure Laboratory. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR2001)*, pages 663–669. LNAI 2083, Springer-Verlag, 2001.
2. A. Armando and S. Ranise. Constraint Contextual Rewriting. In *Proc. of the 2nd Intl. Workshop on First Order Theorem Proving (FTP'98)*, 1998.
3. A. Armando and S. Ranise. Termination of Constraint Contextual Rewriting. In *Proc. of the 3rd Intl. W. on Frontiers of Comb. Sys.'s (FroCos'2000)*, volume 1794, pages 47–61. Springer-Verlag, 2000.
4. A. Armando and S. Ranise. A Practical Extension Mechanism for Decision Procedures: the Case Study of Universal Presburger Arithmetic. *Journal of Universal Computer Science*, 7(2):124–140, February 2001. Special Issue on Tools for System Design and Verification (FM-TOOLS'2000).
5. N. S. Bjørner. *Integrating Decision Procedures for Temporal Verification.* PhD thesis, Computer Science Department, Stanford University, 1998.
6. R.S. Boyer and J S. Moore. *A Computational Logic.* Academic Press, 1979.
7. R.S. Boyer and J S. Moore. Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic. *Machine Intelligence*, 11:83–124, 1988.

8. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Hand. of Theoretical Comp. Science*, pages 243–320. 1990.
9. D. L. Detlefs, G. Nelson, and J. Saxe. Simplify: the ESC Theorem Prover. Technical report, DEC, 1996.
10. D. Kapur, D.R. Musser, and X. Nie. An Overview of the Tecton Proof System. *Theoretical Computer Science*, Vol. 133, October 1994.
11. M. Kaufmann and J S. Moore. Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. Soft. Eng.*, 23(4):203–213, 1997.
12. D. E. Knuth and P. E. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297, Oxford, 1970. Pergamon Press.
13. J.-L. Lassez and M.J. Maher. On Fourier's Algorithm's for Linear Arithmetic Constraints. *J. of Automated Reasoning*, 9:373–379, 1992.
14. Z. Manna and the STeP Group. STeP: The Stanford Temporal Prover. Technical Report CS-TR-94-1518, Stanford University, June 1994.
15. V. Maslov and W. Pugh. Simplifying Polynomial Constraints Over Integers to Make Dependence Analysis More Precise. Technical Report CS-TR-3109.1, Dept. of Computer Science, University of Maryland, 1994.
16. G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford Computer Science Department, April 1978.
17. S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNAI*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
18. L. C Paulson. Proving termination of normalization functions for conditional expressions. *J. of Automated Reasoning*, pages 63–74, 1986.
19. R.E. Shostak. Deciding Combination of Theories. *J. of the ACM*, 31(1):1–12, 1984.
20. H. Zhang. Contextual Rewriting in Automated Reasoning. *Fundamenta Informaticae*, 24(1/2):107–123, 1995.

SAT-Based Decision Procedures for Automated Reasoning: A Unifying Perspective

Alessandro Armando[1], Claudio Castellini[1], Enrico Giunchiglia[1], Fausto Giunchiglia[2,3], and Armando Tacchella[1]

[1] DIST, Università di Genova,
viale Causa 13, 16145 Genova – Italy
{armando,drwho,enrico,tac}@dist.unige.it
[2] DICT, Università di Trento,
Povo, 38100 Trento – Italy
fausto@cs.unitn.it
[3] ITC-IRST,
via Sommarive 18, 38050 Trento – Italy

Abstract. Propositional reasoning (SAT) is an essential part of many reasoning tasks. Many problems in computer science can be compiled to SAT and then effectively decided using state-of-the-art solvers. Alternatively, if reduction to SAT is not feasible, the ideas and technology of state-of-the-art SAT solvers can be useful in deciding the propositional component of the reasoning task being considered. This last approach has been used in different contexts by different authors, many times by authors of this paper. Because of the essential role played by the SAT solver, these decision procedures have been called "SAT-based". SAT-based decision procedures have been proposed for various logics, but also in other areas such as planning. In this paper we present a unifying perspective on the various SAT-based approaches to these different reasoning tasks.

1 Introduction

Propositional reasoning (SAT) is an essential part of many reasoning tasks. Many problems in computer science can be compiled to SAT and then effectively solved using state-of-the-art solvers, see, e.g., [Kautz and Selman, 1992, Kautz and Selman, 1996, Biere *et al.*, 1999]. Alternatively, if reduction to SAT is not feasible, the ideas and technology of state-of-the-art SAT solvers can be useful in deciding the propositional component of the reasoning task being considered. This last approach has been used in different contexts by different authors, many times by authors of this paper. Because of the essential role played by the SAT solver, it has been called "SAT-based" in [Giunchiglia and Sebastiani, 1996b]. That paper is about decision procedures for modal logics. The same topic is dealt with in [Giunchiglia and Sebastiani, 1996a, Giunchiglia *et al.*, 1998, Giunchiglia *et al.*, 2000b]. SAT-based decision procedures for decidable fragments of first order logic are presented in [Armando and Giunchiglia, 1989, Armando and

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 46–58, 2005.

Giunchiglia, 1993]. Finally, SAT-based decision procedures have been proposed in temporal reasoning [Armando *et al.*, 1999] and planning [Giunchiglia *et al.*, 2000b, Giunchiglia *et al.*, 2001, Wolfman and Weld, 1999].

In this paper, we present a unifying perspective on the various SAT-based approaches to the different reasoning tasks previously considered. In particular, in Section 2 we present the common ideas of all these various works. Then in Section 3 we show some optimizations to the basic procedures described in Section 2. Finally, we review the cited works on SAT-based decision procedures, presenting them in the context of the unifying framework previously introduced. We conclude the paper in Section 5, with some final remarks.

2 SAT-Based Decision Procedures: A Unifying Perspective

In the following, we consider an arbitrary logic characterized as a pair $\mathcal{L} = \langle L, T \rangle$ where

- L is the *language*, i.e., a set of formulae in some formal language which includes the standard propositional connectives, i.e., the unary connective $\neg$, the binary connective $\supset$, and the k-ary ($k \geq 0$) connectives $\vee, \wedge, \equiv$; and
- T is a theory, i.e. a subset of the language closed under propositional reasoning and the additional rules specific of the logic at hand.

We also assume that the logic is consistent, i.e., that for any formula φ in the language L, it is not the case that both φ and $\neg\varphi$ belong to T; and decidable in the standard sense, see, e.g., [Dreben and Goldfarb, 1979].

In this paper we focus on the following problem:

> Given a logic $\mathcal{L} = \langle L, T \rangle$ and a formula $\varphi \in L$, is the formula $\mathcal{L}$-consistent? That is, does $\neg\varphi$ belong to $L \setminus T$?

The decidability of the logic ensures that the task of deciding the $\mathcal{L}$-consistency of any given formula in L can be accomplished.

In the following, an *atom* of L is a formula whose main symbol is not a propositional connective, i.e., is not in $\{\neg, \supset, \vee, \wedge, \equiv\}$. A *literal* is an atom or the negation of an atom. An *assignment* μ is a finite conjunction of literals such that it is not the case that both ψ and $\neg\psi$ are conjuncts of μ. An assignment μ *satisfies* a formula φ if the formula $\mu \supset \varphi$ can be proved by propositional reasoning. We write $\mu(A) = \top$ as an abbreviation for "A is a conjunct of μ", and $\mu(A) = \bot$ as an abbreviation for "$\neg A$ is a conjunct of μ".

The basic idea behind the SAT-based approach to determine the $\mathcal{L}$-consistency of a formula φ is very simple and consists of the following two steps:

1. *generate* a (possibly partial) assignment which propositionally satisfies the formula, and then
2. *test* that the generated assignment is indeed consistent w.r.t. $\mathcal{L}$.

Given that the generation step involves propositional reasoning only, it is possible to use state-of-the-art SAT solvers for generating assignments. Because of this, we inherit the many optimizations and heuristic strategies (improving the average case behavior) which are implemented in current SAT solvers. Notice that in principle any set S of assignments satisfying φ can be generated and tested. However, in all the SAT-based procedures, the main focus has been on the generation of complete and irredundant sets of assignments. A set S of assignments is

- *complete* for a formula φ, if φ is propositionally logically equivalent to the disjunction of the assignments in S; and
- *irredundant* for a formula φ, if for any assignment $\mu \in S$ we have that $S \setminus \{\mu\}$ is not complete.

Completeness is required to have correct and complete SAT-based procedures. Irredundancy is very important too, as it improves efficiency in many cases.

Several algorithms and techniques have been proposed to solve the satisfiability problem in propositional logic (see, e.g., [Gu *et al.*, 1997]). Among this variety of approaches we have chosen the Davis-Logemann-Loveland (DLL) method [Davis *et al.*, 1962] to develop our decision procedures. The reasons for this choice are manyfold:

- DLL is a simple and elegant algorithm whose implemented variants proved to be very effective in attacking hard SAT instances;
- since most state-of-the-art solvers are based on DLL, there is a lot of knowledge on data structures and algorithms that we can inherit in our setting for free;
- even if DLL is usually tuned to find a single satisfying assignment, it is rather easy to modify it to generate a complete and irredundant set of assignments.

Examples of state-of-the-art SAT solvers based on DLL are BÖHM [Buro and Buning, 1992, Böhm and Speckenmeyer, 1996], SATZ [Li and Anbulagan, 1997], RELSAT v2.0 [Bayardo, Jr. and Schrag, 1997], SATO v3.2 [Zhang, 1997], SIM [Giunchiglia *et al.*, 2001], and – more recently – CHAFF [Moskewicz *et al.*, 2001].

A basic DLL implementation for SAT-based reasoning in outlined in Figure 1. The conventions that we use to present the algorithms are those of [Cormen *et al.*, 1998], described at pages 4 and 5. In particular, variables (e.g. Γ, S, l) are treated as pointers to the data structures representing the corresponding entities. If a pointer does not refer to any object, we give it the special value NIL. Stacks are considered a primitive data type and are accessed with the usual constant-time primitives PUSH and POP, while the function EMPTY builds and returns an empty stack. The primitive ASSIGN(Γ, l) returns the set of clauses Γ minus all the clauses in which literal l occurs, and with all the occurrences of its negation, $\overline{l}$, removed. Finally, we assume that T, F, LA, LB, and HR are five pairwise distinct constants, each one being distinct from NIL. In particular, T and F represent logical truth and falsehood, respectively.

Function DLL-SOLVE takes a formula φ as input and returns T (F) exactly when φ is satisfiable (unsatisfiable, resp.). Function DLL-SOLVE converts φ into

```
DLL-Solve(φ)
 1 Γ ← CNF-Convert(φ)
 2 S ← Empty( )
 3 return DLL-Solve-CNF(Γ, S)

DLL-Solve-CNF(Γ, S)
 1 next ← LA
 2 repeat
 3   case next of
 4     LA : next ← Look-Ahead(Γ, S)
 5     HR : next ← Heuristic(Γ, S)
 6     LB : next ← Look-Back(Γ, S)
 7 until next ∈ {T, F}
 8 return next

Look-Ahead(Γ, S)
 1 for each l deduced from Γ do
 2   S ← Push(S, ⟨Γ, l, LA⟩)
 3   Γ ← Assign(Γ, l)
 4   if an empty clause is in Γ then
 5     return LB
 6 if Γ is not empty then
 7   return HR
 8 else
 9   return Is-Consistent(S)

Is-Consistent(S)
 1 μ ← Assignment-In(S)
 2 if L-Consist(μ) = T then
 3   return T
 4 else
 5   return LB

Heuristic(Γ, S)
 1 Choose a literal l in Γ
 2 S ← Push(S, ⟨Γ, l, HR⟩)
 3 Γ ← Assign(Γ, l)
 4 return LA

Look-Back(Γ, S)
 1 repeat
 2   ⟨Γ, l, r⟩ ← Pop(S)
 3 until r = HR
 4 if length[S] = 0 then
 5   return F
 6 else
 7   S ← Push(S, ⟨Γ, l̄, LB⟩)
 8   Γ ← Assign(Γ, l̄)
 9   return LA
```

Fig. 1. Implementation of the DLL method for SAT-based reasoning.

an equi-satisfiable clausal normal form Γ (line 1) using the function CNF-Convert. We do not discuss here CNF-Convert and the issues related to conversion in clausal normal form. More details can be found in [Giunchiglia *et al.*, 2000b]. Here it is sufficient to say that the conversion can be done in such a way that $|\Gamma|$ is in $O(|\varphi|)$ where $|\varphi|$ is the size, i.e. the number of symbols of φ. Function DLL-Solve also initializes the search stack S (line 2) and then calls DLL-Solve-CNF to determine the satisfiability of Γ. The elements of the search stack are triples of the form $\langle \Gamma, l, flag \rangle$, where Γ is a set of clauses, l a literal, and $flag \in \{\text{T}, \text{F}, \text{LA}, \text{LB}, \text{HR}\}$. Function Assignment-In takes as input the search stack S and returns a conjunction of the literals stored in it, i.e. $\bigwedge_{\langle -, l, - \rangle \in S} l$. Function DLL-Solve-CNF solves Γ by iteratively applying one of the following steps:

Look-Ahead to deduce new truth assignments from Γ. Look-Ahead keeps simplifying Γ (for instance, by exploiting the unit clauses in it) until an inconsistency arises or a fix point is reached. In case of inconsistency (line 4), the return value of Look-Ahead is LB, meaning that the main loop has to call Look-Back. In case of a fix point, we have two possibilities: if there are still clauses in Γ, then the return value is HR; if Γ is empty, then all the clauses have been satisfied and the function Is-Consistent is invoked.

HEURISTIC to decide the next truth assignment and to enforce it; the decision is taken by considering Γ and/or possibly some Γ' obtained from Γ by tentatively assigning truth values to literals.

LOOK-BACK to undo truth assignments, until a point from which the search can continue without loosing solutions. If there is no such a point (i.e., the search tree is complete), then LOOK-BACK concludes that the initial formula cannot be satisfied.

Since Γ and S are pointers, the actions LOOK-AHEAD, LOOK-BACK, and HEURISTIC all update the input formula and stack of DLL-SOLVE-CNF in various ways during the **repeat** ... **until** loop (lines 2-7). Each action modifies Γ and S and returns the next action to be taken which is stored in *next* (lines 4-6 in the program). LA, LB, HR, T, and F are the possible values taken by *next*, meaning that the next action must be LOOK-AHEAD, LOOK-BACK, HEURISTIC, or that of stopping the loop respectively. In the latter case, if *next* is assigned T then Γ is satisfied, otherwise (i.e. if *next* is assigned F) Γ is unsatisfiable. When Γ is satisfiable, the corresponding satisfying truth assignment μ can be extracted from S. This task is accomplished in Figure 1 by IS-CONSISTENT which extracts μ from S (lines 1-3) and then calls the consistency test L-CONSIST specific for the logic at hand. Notice that in the case of mere propositional satisfiability IS-CONSISTENT simply returns T.

3 Optimizations

The simple generate and test strategy implemented by the SAT-based procedure outlined in Section 2 can be improved in several ways. Here we present two optimizations which often lead to dramatic improvements in the performance of the procedure.

3.1 Adding Constraints to the Input Formula

A key feature of the SAT-based procedure presented in Section 2 is that all the consistency checks are carried on-line. An alternative is to preprocess the formula and look for sets of literals in the input formula that are $\mathcal{L}$-inconsistent[1]. If S is one of such sets, the clause $\bigvee_{l \in S} \overline{l}$ can be added to the formula at hand without affecting its $\mathcal{L}$-consistency. This simple optimization can be very effective as shown by the following example.

Let $\mathcal{L}$ be the quantifier-free fragment of first-order logic with equality and let φ be a formula of the form

$$(x = y \wedge \neg y = x) \wedge \ldots \tag{1}$$

All the propositional assignments generated by DLL-SOLVE are then rejected by L-CONSIST. The useless generation of many propositional assignments is

[1] It is worth pointing out that this check can be carried out by any correct, even though not necessarily complete, procedure.

due to the failure of DLL-Solve to recognize that the truth values of $x = y$ and $y = x$ are not independent. The role of the constraints added by the proposed optimization is to rule out such assignments. For instance, by adding the constraint

$$\neg x = y \vee y = x \tag{2}$$

to (1) we obtain a formula which is readily found unsatisfiable by DLL-Solve.

In the context of SAT-based procedures, the idea of adding constraints has been introduced in [Armando *et al.*, 1999]. In that paper, all pairs of mutually inconsistent inequalities (i.e., $x - y \leq 0$ and $x - y \geq 5$) are detected *a priori* at a reasonable cost, and for each such pair a constraint is added. As a result, the search is greatly reduced. The idea can be generalized to n-uples of inconsistent inequalities, but only until the cost of the preprocessing remains sustainable.

The idea of constraints generalizes the pre-processing technique introduced in [Giunchiglia and Sebastiani, 1996a], and since then used in all the subsequent papers on SAT-based procedures for modal logics. In that paper, the input formula is initially pre-processed by taking into account standard properties of the propositional connectives, e.g. associativity and commutativity. Thus, for example, the formula

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_2 \vee \psi_1) \wedge \ldots \tag{3}$$

is translated into

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_1 \vee \psi_2) \wedge \ldots$$

and thus easily recognized as unsatisfiable. In the current approach, we can detect the $\mathcal{L}$-inconsistency of the set of formulae:

$$\{\Box(\psi_1 \vee \psi_2), \neg\Box(\psi_2 \vee \psi_1)\}$$

and this would add the constraint

$$\neg\Box(\psi_1 \vee \psi_2) \vee \Box(\psi_2 \vee \psi_1)$$

to (3) thereby leaving us with a trivially inconsistent formula. As a final remark, it is worth emphasizing that the strategy here presented is more general than pre-processing because it allows us to rule out assignments in cases where pre-processing is of no help. For example, consider the formula

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_1 \vee \psi_2 \vee \psi_3).$$

Using our strategy, by simple syntactic manipulations, we can build and add the following constraint:

$$\neg\Box(\psi_1 \vee \psi_2) \vee \Box(\psi_1 \vee \psi_2 \vee \psi_3)$$

3.2 Introducing CBJ and Learning

Since the basic DLL algorithm of Section 2 relies on simple chronological backtracking, it is not infrequent for DLL to keep exploring a possibly large subtree

```
LOOK-AHEAD(Γ, S)
 1 for each l s.t. l̄ ∧ ASSIGNMENT-IN(S)
       falsifies Γ with reason r do
 2   S ← PUSH(S, ⟨Γ, l, r⟩)
 3   Γ ← ASSIGN(Γ, l)
 4   if a clause r′ ∈ Γ
         has become empty then
 5     S ← PUSH(S, ⟨NIL, NIL, r′⟩)
 6     return LB
 7 if Γ is not empty then
 8   return HR
 9 else
10   r″ ← IS-CONSISTENT(S)
11   if r″ ≠ NIL then
12     S ← PUSH(S, ⟨NIL, NIL, r″⟩)
13     return LB
14   else
15     return T

IS-CONSISTENT(S)
 1 μ ← ASSIGNMENT-IN(S)
 2 if L-CONSIST(μ) = T then
 3   return NIL
 4 else
 5   return L-EXTRACT-REASON(μ)

HEURISTIC(Γ, S)
 1 Choose a literal l in Γ
 2 S ← PUSH(S, ⟨Γ, l, NIL⟩)
 3 Γ ← ASSIGN(Γ, l)
 4 return LA

LOOK-BACK(Γ, S)
 1 ⟨−, −, r⟩ ← POP(S)
 2 wr ← INIT-REASON(r)
 3 repeat
 4   ⟨Γ, l, r⟩ ← POP(S)
 5   wr ← UPDATE-REASON(wr, l, r)
 6 until r = NIL and
         IS-IN-REASON(l, wr)
 7 if length[S] > 0 then
 8   S ← PUSH(S, ⟨Γ, l̄, wr⟩)
 9   Γ ← ASSIGN(Γ, l̄)
10   return LA
11 else
12   return F
```

Fig. 2. Modifying DLL-SOLVE to introduce CBJ.

whose leaves are all dead-ends. This phenomenon occurs also when the formula is satisfiable, but some choice performed way up in the search tree is responsible for the constraints to be violated. A solution borrowed from the constraint satisfaction literature (see, e.g., [Prosser, 1993]) is to jump back over the choices that do not belong to the reason for the failure. Intuitively, if μ is an assignment which falsifies the input formula φ, then a *reason* ν for μ is a subset of the literals in μ such that any assignment extending ν falsifies φ. Reasons are initialized as soon as an inconsistency is detected, and updated while backtracking. The corresponding technique is widely known as (Conflict-Directed) Backjumping (CBJ). In Figure 2 we show how to modify the functions LOOK-AHEAD, IS-CONSISTENT, HEURISTIC, and LOOK-BACK presented in Figure 1 to introduce CBJ. The elements of the search stack are now triples of the form $\langle \Gamma, l, r \rangle$, where Γ is a set of clauses, l a literal, and r a reason.

Looking at Figure 2 we see that each deduction carried out by LOOK-AHEAD is now justified by a *reason* (line 1). For example, if a literal l occurs in a unit clause, then it is assigned the truth value T and the reason for such an assignment is the set of literals that caused the clause to become unit, see, e.g., [Prosser, 1993]. Notice that LOOK-AHEAD records the reasons of each deduction using the search stack S (line 2), but also the reason for a propositional dead-end

(line 5) and the possible failure of the Is-Consistent test (line 12). All such reasons are used by the function Look-Back in order to identify the choices performed by Heuristic that led to the dead-end in the search. Notice that the algorithms shown in Figure 2 smoothly combine CBJ for propositional failures as well as for failures originated by Is-Consistent. In this case, the assignment in S propositionally satisfies Γ, so triggering propositional backjumping would lead to incorrect results. This is why we need an additional function L-Extract-Reason in Is-Consistent to extract the reason for the failure of μ in the specific logic at hand. The function L-Extract-Reason has to return a subset of the literals in the current assignment which is not $\mathcal{L}$-consistent. In principle any such a set can be returned, e.g. the set of literals in μ. However, returning a smaller subset has the advantage of potentially enabling backjumping.

CBJ can be very effective in "shaking" the solver from regions where no solutions can be found. However, since the reasons of the conflict are discarded as soon as it gets mended, the solver may get repeatedly stuck in such regions. To escape this pattern, some sort of global knowledge is needed: the reasons of the conflicts may be turned into additional constraints (i.e., clauses) that have to be satisfied. As long as we have a function that turns reasons into clauses, it is quite easy to implement learning on top of DLL with CBJ. With reference to Figure 2, it is sufficient to add an instruction that converts the working reasons *wr* created inside Look-Back into additional constraints. In this way, we end up adding all the clauses corresponding to the reasons of the discovered conflicts and the same mistake is never repeated. On the other hand, this may cause an exponential blow up of the size of the formula. In practice, it is necessary to introduce some limit to the number of stored clauses, either by dropping some of the clauses that should be learned, or by periodically removing some of the learnt clauses. For more details on learning see, e.g., [Giunchiglia *et al.*, 2001].

CBJ and learning have been proposed and successfully used in SAT-based procedures for planning [Wolfman and Weld, 1999, Castellini *et al.*, 2001]. In particular, [Wolfman and Weld, 1999] proposes a SAT-based procedure for classical planning with resources, while [Giunchiglia, 2000, Castellini *et al.*, 2001] present a SAT-based procedure for conformant planning in nondeterministic domains, see the respective papers for more details. It is worth mentioning that both in [Wolfman and Weld, 1999] and in [Castellini *et al.*, 2001] the function L-Extract-Reason returns a minimal subset of the current assignment whose extensions are bound to fail. By returning such a subset, the hope is to maximise the effects of CBJ and learning.

4 SAT-Based Decision Procedures: Examples

In this Section we briefly review some specific examples of SAT-based decision procedures for quantifier-free decidable fragments of First-Order Logic (Subsection 4.1), temporal reasoning (Subsection 4.2) and various modal logics (Subsection 4.3).

4.1 Quantifier and Function-Free FOL

In [Armando and Giunchiglia, 1989, Armando and Giunchiglia, 1993], a SAT-based decision procedure for the quantifier- and function-free fragment of First-Order Logic (FOL) is presented. The language may thus have individual constants and variables as well as predicate symbols of any arity.

Let φ be formula in this language and let μ be an assignment returned by DLL-SOLVE(φ). If φ does not contain equalities, the existence of such an assignment μ is sufficient for the $\mathcal{L}$-consistency of φ. Thus, in this case, it is sufficient for L-CONSIST to return T. But if φ contains equalities, then L-CONSIST must determine the satisfiability of μ w.r.t. the properties of equality. More in detail, let $\mathcal{C}$ be the set of terms occurring in φ and let $\simeq$ be the smallest equivalence relation over $\mathcal{C}$ such that if $\mu(c_1 = c_2) = \top$ then $c_1 \simeq c_2$. Similarly, if $\mathcal{A}$ is the set of atomic subformulae occurring in φ then let $\cong$ be the smallest equivalence relation over $\mathcal{A}$ such that if $P(r_1, \ldots, r_n) \in \mathcal{A}$, $P(s_1, \ldots, s_n) \in \mathcal{A}$, and $r_i \simeq s_i$ for $i = 1, \ldots, n$, then $P(r_1, \ldots, r_n) \cong P(s_1, \ldots, s_n)$. An assignment μ is *satisfiable* if and only if it is not the case that there are two terms $c_1, c_2 \in \mathcal{C}$ such that $\mu(c_1 = c_2) = \bot$ and $c_1 \simeq c_2$, or there exist atomic formulae $A_1, A_2 \in \mathcal{A}$ such that $\mu(A_1) = \top$, $\mu(A_2) = \bot$, and $A_1 \cong A_2$. With reference to Figure 1, L-CONSIST

1. looks for the equalities $c_1 = c_2$ such that $\mu(c_1 = c_2) = \top$,
2. builds the data structures representing $\simeq$ and $\cong$, and
3. detects inconsistencies by exploiting the strategy suggested above.

The above procedure can be readily generalized to a SAT-based procedure for the quantifier-free fragment of FOL with uninterpreted function symbols by using a standard congruence closure algorithm (see, e.g., [Nelson and Oppen, 1980]) to perform the consistency checks.

4.2 Linear Constraints over the Reals

In [Armando *et al.*, 1999], the logic admits the function constant "$-$" and the domain of interpretation is fixed to the set of the real numbers. Formally, a *temporal constraint* is a linear inequality of the form $x - y \leq r$, where x and y are variables ranging over the real numbers and r is a real constant. A *disjunctive temporal constraint* is a disjunction of the form $c_1 \vee \cdots \vee c_n$ where $c_1, \ldots, c_n$ are temporal constraints and $n \geq 1$. A *disjunctive temporal problem* (DTP) is a finite set of disjunctive temporal constraints to be intended conjunctively. A *temporal assignment* is a function which maps each variable into a real number. A temporal assignment σ *satisfies* an assignment μ if, for each temporal constraint $x - y \leq r$,

- if $\mu(x - y \leq r) = \top$ then it is indeed the case that $\sigma(x) - \sigma(y) \leq r$,
- if $\mu(x - y \leq r) = \bot$ then it is indeed the case that $\sigma(x) - \sigma(y) > r$.

An *assignment is satisfiable* iff there exists a temporal assignment satisfying it. In the literature, the problem of determining whether an assignment is satisfiable or not is called a *Simple Temporal Problem* (STP). There are a number of procedures for checking the satisfiability of an STP, see, e.g., [Chleq,

1995]. The SAT-based decision procedure for checking the satisfiability of DTPs, TSAT, was implemented on top of Böhm's SAT solver [Buro and Buning, 1992, Böhm and Speckenmeyer, 1996]. TSAT proved to be more effective than the other procedures presented in the literature[2]. One of the reasons is that the semantic branching characteristic of DLL-SOLVE is superior (see also [Oddi and Cesta, 2000]) to the syntactic branching performed by tableau-based procedures proposed in [Stergiou and Koubarakis, 1998]. Moreover, since in TSAT consistency checks are performed by an optimized implementation of the simplex method, the system can efficiently handle temporal constraints involving the "−" and the "+" function symbols, multiplication by constants, and any finite number of variables, whereas the procedures proposed in [Stergiou and Koubarakis, 1998] and in [Oddi and Cesta, 2000] can only deal with the temporal constraints as defined above. Finally, we point out that a SAT-based procedure similar to TSAT is at the basis of the planning system described in [Wolfman and Weld, 1999], which is implemented on top of RELSAT [Bayardo, Jr. and Schrag, 1997].

4.3 Modal Logics

SAT-based decision procedures for modal logics have been proposed in [Giunchiglia and Sebastiani, 1996a, Giunchiglia and Sebastiani, 1996b, Giunchiglia *et al.*, 2000b], and have been comparatively evaluated in [Giunchiglia *et al.*, 2000a, Giunchiglia *et al.*, 2000b]. Moreover, in [Giunchiglia and Sebastiani, 1996a, Giunchiglia and Sebastiani, 1996b] the authors clearly pointed out the potentials of the SAT-based approach.

In the modal logics considered in the above cited papers, the language is extended by allowing denumerately many (modal) unary operators $\Box^1, \ldots, \Box^n$. Depending on the specific properties of each operator, different logics are obtained, from the weakest classical modal logic E, to the normal modal logic K [Chellas, 1980]. Decision procedures for 8 modal logics have been proposed in [Giunchiglia *et al.*, 2000b]. Here, for the sake of conciseness, we restrict our attention to the modal logics E and K.

Consider an assignment $\mu = \wedge_i(\wedge_j \Box^i \alpha_{ij}) \wedge \wedge_i \wedge_j \neg\Box^i \beta'_{ij} \wedge \gamma$ where γ is a propositional formula.

- In the modal logic E, μ *is satisfiable* if for each pair $\Box^i \alpha_{ij}$, $\neg\Box^i \beta'_{ik}$ of conjuncts in μ, the formula $\alpha_{ij} \equiv \neg\beta'_{ik}$ is satisfiable.
- In the modal logic K, μ *is satisfiable* if for each conjunct $\neg\Box^i \beta'_{ij}$ in μ, the formula $\wedge_j \alpha_{ij} \wedge \neg\beta'_{ij}$ is satisfiable.

Thus, in E and in K the problem of determining whether an assignment is satisfiable or not boils down to the problem of determining the satisfiability of "simpler" formulae. Simpler, because the number of modal operators gets reduced. Still, modal operators can be nested, as any standard connective. Thus,

[2] The experimental results reported in [Armando *et al.*, 1999] show that TSAT performs up to 2 orders of magnitude less consistency checks than the best procedure presented in [Stergiou and Koubarakis, 1998].

```
L-CONSIST(∧_i □α_i ∧ ∧_j ¬□β_j ∧ γ)
 1 for each conjunct □β_j do
 2   for each conjunct □α_i do
 3     if not DLL-SOLVE(α_i ≡ ¬β_j)
 4       return then F
 5 return T.
```

```
L-CONSIST(∧_i □α_i ∧ ∧_j ¬□β_j ∧ γ)
 1 for each conjunct □β_j do
 2   if not DLL-SOLVE(∧_i α_i ∧ ¬β_j)
 3     then return F
 4 return T
```

Fig. 3. L-CONSIST for the modal logics E (left) and K (right).

in order to determine the satisfiability of these simpler formulae, L-CONSIST calls the DLL-SOLVE procedure. As a result, we have two mutually recursive procedures. The fact that at each call from L-CONSIST to DLL-SOLVE the number of modal operators diminishes guarantees termination of the whole process. Definitions of the L-CONSIST procedure for E and K are sketched in Figure 3, in case there is a single modality $\Box$. The extension to multiple modalities is straightforward.

Notice that the procedures for E and K of Figure 3 are naive and suffer from the fact that consistency checks of the same set of formulae can be repeated many times. A way out of the problem which has been proved very effective is the incorporation of caching mechanisms. For E, we check the consistency of pairs of formulae, and thus caching can be accomplished using a matrix, see [Giunchiglia *et al.*, 2000b]. For K, we check the consistency of sets of formulae, and thus more complex data structures, such as bit matrices, are needed [Giunchiglia and Tacchella, 2001].

5 Conclusions

In this paper we have provided a unifying perspective of a family procedures for automated reasoning based on the common idea of combining state-of-the-art SAT-solvers with reasoning specialists for the theory at hand. To substantiate our claim, we have shown that a variety of SAT-based procedures developed in the last decade (namely decision procedures for quantifier-free decidable fragments of First-Order Logic, for temporal reasoning, and for several modal logics) can be readily recast in our framework.

References

[Armando and Giunchiglia, 1989] A. Armando and F. Giunchiglia. On tautology decision techniques: Complexity and implementation considerations. Technical Report 8911-08, IRST, Trento, Italy, 1989.

[Armando and Giunchiglia, 1993] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[Armando *et al.*, 1999] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Lecture Notes in Computer Science*, volume 1809, pages 97–108, 1999.

[Bayardo, Jr. and Schrag, 1997] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 203–208, Menlo Park, July 27–31 1997. AAAI Press.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 1999.

[Böhm and Speckenmeyer, 1996] M. Böhm and E Speckenmeyer. A fast parallel SAT-solver – efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381–400, 1996.

[Buro and Buning, 1992] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Castellini *et al.*, 2001] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In *ECP*, 2001.

[Chellas, 1980] B. F. Chellas. *Modal Logic – an Introduction*. Cambridge University Press, 1980.

[Chleq, 1995] N. Chleq. Efficient algorithms for networks of quantitative temporal constraints. In *Proceedings of CONSTRAINTS–95*, pages 40–45, April 1995.

[Cormen *et al.*, 1998] Thomas H. Cormen, Charles E. Leiserson, and Ronald R. Rivest. *Introduction to Algorithms*. MIT Press, 1998.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Dreben and Goldfarb, 1979] Burton Dreben and Warren D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas*. Addison-Wesley Publishing Company, Reading, MA, 1979.

[Gent *et al.*, 2000] Ian Gent, Hans Van Maaren, and Toby Walsh, editors. *SAT2000. Highlights of Satisfiability Research in the Year 2000*. IOS Press, 2000.

[Giunchiglia and Sebastiani, 1996a] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE-96*, Lecture Notes in Artificial Intelligence, New Brunswick, NJ, USA, August 1996. Springer Verlag.

[Giunchiglia and Sebastiani, 1996b] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996. Also DIST-Technical Report 9607-08 and IRST-Technical Report 9601-02.

[Giunchiglia and Tacchella, 2001] Enrico Giunchiglia and Armando Tacchella. A subset-matching size-bounded cache for testing satisfiability in modal logics. *Annals of Mathematics and Artificial Intelligence*, 33:39–67, 2001.

[Giunchiglia *et al.*, 1998] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

[Giunchiglia *et al.*, 2000a] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. SAT vs. Translation Based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non Classical Logics*, 10(2):145–172, 2000.

[Giunchiglia *et al.*, 2000b] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT-Based Decision Procedures for Classical Modal Logics. *Journal of Automated Reasoning*, 2000. To appear. Reprinted in [Gent *et al.*, 2000].

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083*, 2001.

[Giunchiglia, 2000] Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 2000.

[Gu *et al.*, 1997] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (sat) problem: A survey. *Satisfiability Problem: Theory and Applications*, pages 19–153, 1997.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[Nelson and Oppen, 1980] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[Oddi and Cesta, 2000] A. Oddi and A. Cesta. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, pages 108–112, Berlin, 2000.

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Stergiou and Koubarakis, 1998] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. In *Proc. AAAI*, 1998.

[Wolfman and Weld, 1999] Steven Wolfman and Daniel Weld. The LPSAT-engine & its application to resource planning. In *Proc. IJCAI-99*, 1999.

[Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, July13–17 1997. Springer.

Temporal Dynamics of Support and Attack Networks: From Argumentation to Zoology

Initial Results

Howard Barringer[1], Dov Gabbay[2], and John Woods[3]

[1] School of Computer Science,
The University of Manchester,
Oxford Rd, Manchester M13 9PL, UK
[2] Department of Computer Science,
King's College London,
Strand, London WC2R 2LS, UK
[3] Department of Philosophy,
University of British Columbia,
1866 Main Mall E370, Vancouver BC Canada V6T 1Z1

1 Introduction

This is the first of a new series of papers on the temporal dynamics of Support and Attack networks. These are graphs with a basic situation described in Figure 5 below. We have nodes $a_1, \ldots, a_n$ connected by arrows to a node b. The nodes have some values attached to them and these values are transmitted by the arrows, and revise the value at b. This series of papers studies the temporal dynamics of such networks. The topic, in this generality, has emerged from our previous research into argumentation frameworks (Gabbay and Woods [14, 13, 10, 12] and Woods [21]). Our starting point is therefore a generalisation of abstract argumentation networks.

Abstract argumentation frameworks were put forward by Dung [5] following the realisation that in real life every argument has a counter argument and no argument is conclusive. An argumentation network has the form (AR, Attack), where AR is a set of arguments and Attack $\subseteq$ AR^2 is an irreflexive binary relation on AR, indicating which argument attacks which arguments. We should emphasise that our approach here is *dialectical* rather than *normative*. When we say that every argument has a counterargument it is not our view that every argument deserves a counter, but rather than every argument, whatever its merits, lies open to a counter, whatever *its* merits. We say that no argument is conclusive, we intend that every argument is susceptible to challenge, again notwithstanding its presumed merits. And when we say (just below) that any argument that attacks an argument is a refutation of it, we intend only that the attacking argument is presented as a refutation and that the attacked argument is, on that argument, put under challenge. Our purpose in emphasising descriptive factors in Support and Attack networks is twofold. We have reservations about the speed with which some argumentation theorists rush to normative judgement; and, in any event, we take description to have an expository priority over normative considerations in theoretical accounts of argumentative practice.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 59–98, 2005.

Example 1. Figures 1 to 3 are three examples of such networks.

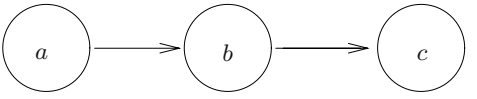


Fig. 1.

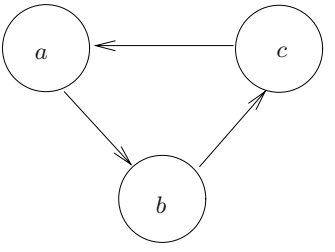


Fig. 2.

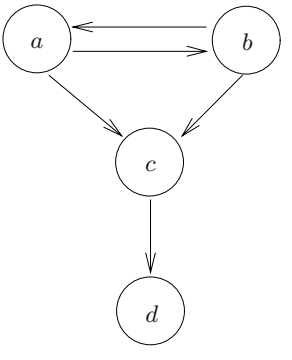


Fig. 3.

(a) The situation in Figure 1 is straightforward. a is not attacked by anything, so it is *in* (or active) as an argument. Since it attacks b, b is *out* (or is a refuted argument) and so c is *in*. So the net result of Figure 1 can be written as $\{+a, -b, +c\}$.

(b) The situation in Figure 2 is a complete loop. No argument can definitely be said to be in or out. We write this as $\{?a, ?b, ?c\}$.

(c) The situation in Figure 3 is more interesting. Here a and b attack each other; so we have $?a, ?b$. Because of that, we can also put it that $?c$ and $?d$. However we can observe that both a and b attack c; so no matter which of a or b are in (i.e. whether we have $\{+a, -b\}$ or $\{-a, +b\}$), we always have $-c$, and so the net result could be taken to be $\{?a, ?b, -c, +d\}$. On the other hand, we might adopt the view that a, b cancel each other, in which case the net result would be $\{-a, -b, +c, -d\}$.

Since circularity, loops and mutual attacks of arguments are very common in real life, it is obvious that much attention is required to resolving loops in argumentation networks. Abstract argumentation networks were generalised by Bench-Capon [4], where a colouring (representing the type of argument) was added to the network. The colours are linearly ordered by strength. A weaker coloured node cannot successfully attack a stronger coloured node. So a network with colours (or with valuations) has the form (AR, Attack, V) where, as before, Attack $\subseteq$ AR $\times$ AR and V is a function giving, say, numbers to nodes: $V : AR \mapsto$ *Numbers*, and the numbers represent strength.

Thus in Figure 2 suppose $V(b) = r$ and $V(a) = V(c) = s$. Clearly if $r < s$, then the net outcome of the network is $\{+b, +c, -a\}$. If $r > s$, then the result is $\{+b, +a, -c\}$. If $r = s$, we get, as before, $\{?a, ?b, ?c\}$.

Note that technically the colouring function V is an instrument for cancelling attacks from some nodes to others. However, it is an instrument that requires restrictions. Not every proposed list of attacks to be cancelled can be implemented by a function V. Consider Figure 2. Suppose that we want to cancel all attacks. To cancel the attacks of a on b and of b on c we must have $V(a) < V(b) < V(c)$. By transitivity $V(a) < V(c)$, so the attack by c on a cannot be cancelled by V.

The main rationale behind the introduction of V is not necessarily the resolution of loops or cancellation of attacks, but the modelling of the intuition that arguments can be divided into kinds, and that some kinds of arguments are more important than others.

This paper generalises argumentation networks in several directions.

1. It allows for nodes in argumentation networks not only to attack other nodes but also for support of other nodes. Moreover, we allow for varying strengths of attack and support. We further generalise the model such that strengths of attacks or support are themselves subject to attack or support. See Figure 4 for example.
2. It allows for the strengths of attack or support to be time dependent. This enables us to model the phenomenon of 'Let's lie low and wait for the argument to blow away'.
3. This paper also examines loop-resolution in argumentation networks, and explores similarities between such loops and predator–prey models in mathematical biology.

The plan of the paper is as follows:

Section 2 will discuss "attack only" networks. There are three problems to be addressed in such networks.

1. The formal definition and motivation of a variety of attack networks.
2. The modes of attack, a discussion of various option as to how to calculate the result of attacks.
3. The resolution of attack loops, such as Figures 2 and 3.

Section 3 is devoted to various methods for the resolution of attack loops. In the course of deciding how to handle loops, we explore formal connections of networks with loops in networks occurring in mathematical biology. In biology

the main emphasis is on a variety of ecology loops. The connection is simple; argument a attacking argument b can also be understood as species a preying on species b. Connections are also explored with the general theory of network modalities.

Section 4 deals with networks that allow for both attack and support arrows. We quickly ascertain the need to redefine the way in which attack and support are (numerically) carried out, and our considerations lead us to a surprising connection with the Dempster–Shafer rule and with the cross-ratio and projective metrics in geometry.

Section 5 deals with time-dependent attack and support of arguments. Here a connection with artificial intelligence time–action models is established, as well as a connection with dynamical systems and general temporal logics.

2 Attack-Only Networks with Strength

We begin this section with an example motivating and explaining the idea of strength of a node and strength of attack on a node.

Example 2. Consider the election for Governor of California and the then candidate, actor Arnold Schwarzenegger. Let

$a =$ The candidate is alleged to have a certain attitude towards women, and to have behaved towards them accordingly.

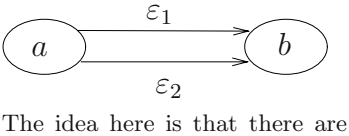
$b =$ The candidate will run California very well.

These arguments may have different strengths based on evidence for case a and training and experience for case b. There is also another argument concerning the question of to what extent can a attack b. Is a relevant at all to b and to what degree? We represent this situation by the network in Figure 4

The node $\varepsilon : (ab)$, where (ab) is the attacking arc from a to b, represents the strength of the argument that a is relevant to b. It therefore can also be attacked, since one can argue against any connection between a and b.[1]

Consider the situation described in Figure 5 where argument a has strength x. It attacks argument b, which initially has strength y.

[1] The model also allows several attacks to emanate from the same argument, as in the figure below.



The idea here is that there are several different kinds of arguments as to why a is an attack upon b. This makes sense especially if a is a fact (see below). Such networks exist in the literature as *transition systems*, and the different arrows from a to b represent different actions, leading from state a to state b.

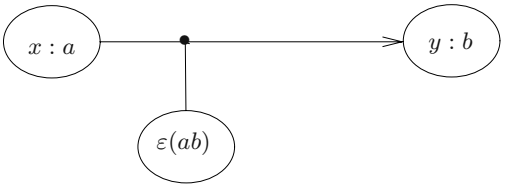


Fig. 4.

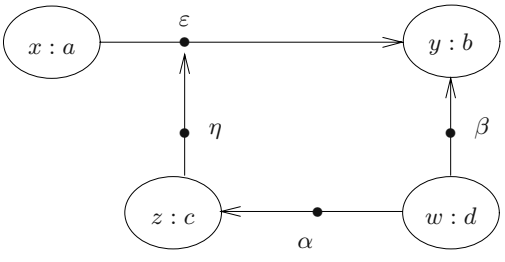


Fig. 5.

ε is the transmission factor, weakening b in a way that takes account of $x : a$. b is also attacked by d with factor β.

However, factor ε is attacked by argument c, which is itself attacked by d, with transmission factor α.

This model has two innovations.

1. The strength of nodes and the transmission factor.
2. The idea that the transmission factor can itself be attacked.

What kind of network does Figure 5 represent? First, note that the strength of nodes is actually a colouring of them. One might expect us to introduce a transmission factor between colours, then in Figure 4 ε could depend only on x and y. We choose to make ε depend on the nodes, taking into consideration that the transmission factor depends on the nature of the argument and not just on their strengths.

The option of attacking transmission factors enables us to delete attacks, one by one, by attacking (lowering) their transmission factor.

Example 3 (Modes of attack). Consider a simple numerical model. Assume all values are between 0 and 1. If a is an argument of strength x which is attacking an argument b of strength y, and the transmission rate is ε, then we get εx as the value transmitted. The question now is how does this value εx reduce the value y of b to a new value y'? We have two options. The first is that the attack reduces the value y of b in proportion, i.e. by εx. Thus the new value of b is $y(1 - \varepsilon x)$. The second option is that the new value of y is $y' = \varepsilon xy$. This second option makes sense if we view the attack of a on b as a pre-emptive protective measure, reducing a possible attack of b on a. If a is strong ($x = 1$) and $\varepsilon = 1$ then $1 - x\varepsilon = 0$ whereupon a destroys b. This is the previous option, being a

genuine attack. However $\varepsilon xy = y$ when $\varepsilon = 1$ and $x = 1$; so b is not affected. But if x is small, then $y' = \varepsilon xy$ is small. So if b attacks a with transmission rate η, the value of this attack would be $1 - \eta y'$ and the attack would not be effective. Hence the second option can be used as a pre-emptive attack.

We now address the problem of combining attacks. In Figure 5, b is also attacked by d and this attack alone will reduce the value of b to $y(1 - \beta w)$. How do we combine them?

Here too there are two options:

1. Perform the operation of reduction consecutively (and commutatively), so that the new value of b after the joint attack is $y(1 - \beta w)(1 - \varepsilon x)$.
2. Add the two reductions, in which case the new value for b is the value $y - y\varepsilon x - y\beta w = y(1 - \varepsilon x - \beta w)$.

The advantages of option 1 are that we are assured that the new value remains between 0 and 1 no matter how many attacks there are, and that the combination is independent of how the attack is calculated. For example, this can give as the new value of b the combination $\varepsilon x(1 - \beta w)$.

Example 3 above has put forward just one mode of attack. There are many other possible modes. Additional possibilities will be examined in Section 4, in conjunction of models with both attack and support.

In general, we have the situation shown in Figure 6. In this case, we require the following function: If b has value y and if $x_1 : a_1, \ldots, x_n : a_n$ attack $y : b$ with strengths $\varepsilon_1, \ldots, \varepsilon_n$ resp., then we need a function $\mathbf{f}$ such that the new value of node b is $y' = \mathbf{f}(y, x_i, \varepsilon_i)$.

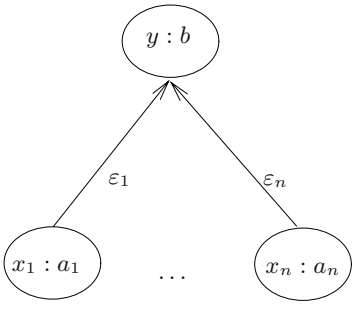


Fig. 6.

This situation is reminiscent of Bayesian networks, where $\mathbf{f}$ is the conditional probability of b on $a_1, \ldots, a_n$.[2]

[2] In Bayesian nets there are no $\varepsilon_1, \ldots, \varepsilon_n$. x_i are the probabilities associated with the nodes a_i and $\mathbf{f}$ is the conditional probability of node b relative to all the a_i. Thus the probability y of b can be calculated.

We adopt option 1 as our mode of attack. So the new value $y' = V(b)$ in Figure 6 is

$$\begin{aligned} y' &= y(1-\varepsilon_1 x_1)\dots(1-\varepsilon_n x_n) \\ &= y\textstyle\prod_i(1-\varepsilon_i x_i). \end{aligned}$$

The magnitude $\Delta^- y$ which y decreases is

$$\Delta^- y = y - y' = y(1-\prod(1-\varepsilon_i x_i)).$$

Example 4. We calculate the transmission of values in Figure 5.

Step 1: The final value V of node d is w, as it is not attacked by anything. Write $V_1(d) = w$. Similarly $V_1(a) = x$. We write V_1 because this is the value obtained as final at Step 1.

Step 2: The new value V_2 of nodes c and b are $V_2(c) = z(1-\alpha w), V_2(b) = y(1-\beta w)$. Of course since nodes a and d have already obtained their final value, we can write: $V_2(a) = V_1(a), V_2(d) = V_2(d)$. Node a cannot transmit because we know from the figure that ε is being attacked, and so we need to wait for its value to change. Only when ϵ gets its final value will a be able to transmit.

Step 3: The new value V_3 of the transmission connection (ab) is

$$\begin{aligned} V_3(ab) &= \varepsilon(1-\eta V_2(c)) \\ &= \varepsilon(1-\eta z(1-\alpha w)). \end{aligned}$$

Of course, $V_3(a) = V_2(a), V_3(d) = V_2(d)$, $V_3(c) = V_2(c)$, and $V_3(b) = V_2(b)$.

Step 4: Now a can transmit to node b. This gives

$$\begin{aligned} V_4(b) &= V_2(b)(1 - V_3(ab)\cdot x) \\ &= y(1-\beta w)(1-\varepsilon x(1-\eta z(1-\alpha w))). \end{aligned}$$

Of course, $V_4(a) = V_3(a), V_4(d) = V_3(d), V_4(c) = V_3(c)$ and $V_4(ab) = V_3(ab)$.

Note that node b has had its value changed in bits and pieces. First, it was changed at Step 1 and then at Step 4. This is all right for the current way of changing values, because it is commutative and cumulative. However, the general definition will now allow for this!

This kind of model contains the traditional one as a special case, where all values are taken to be 1 and there are no attacks on transmissions. Let us see what Figure 5 becomes in this case. Consider Figure 7 and note that it reduces to Figure 8.

We can now give a definition of value propagation for acyclic networks. Cycles will be addressed in the next section.

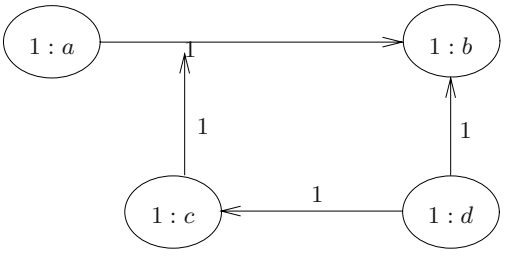


Fig. 7.

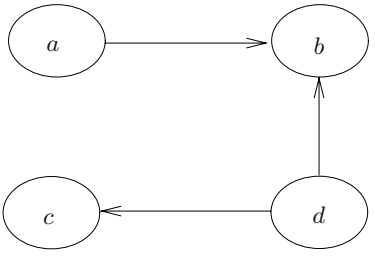


Fig. 8.

To give a definition we need to agree on the representation of the network. Let's do it for the case of Figure 5. We need a set of atomic nodes A. In the case of Figure 5, $A = \{a, b, c, d\}$.

To represent the attack of atomic x on y, i.e. the arrow from x to y, we write the expression $x \looparrowright y$ (called a *torpedo*)[3]. In Figure 5, we have the torpedoes $a \looparrowright b, d \looparrowright c$ and $d \looparrowright b$.

These torpedoes represent the attacks from a to b, d to c and d to b respectively. One of these attacks, namely $a \looparrowright b$, is itself attacked by c. This is represented by the torpedo $c \looparrowright (a \looparrowright b)$.

Note that we *cannot* write an expression of the form $(x \looparrowright y) \looparrowright z$. This would mean that the fact that there is an attack from x to y is in itself an attack on z. We are not saying that such reasoning does not exist. In due course we shall deal with it in the context of fibring networks. In other words, a whole network can be embedded as a node and attack another node.

Figure 5 can be represented by the set of nodes and torpedoes:

$$T = \{a, b, c, d, a \looparrowright b, d \looparrowright b, d \looparrowright c, c \looparrowright (a \looparrowright b)\}.$$

Note that this set T has the property that if $x \looparrowright y \in T$, then $x \in T$ and $y \in T$. What we still need are the numbers (valuations) in the figure. This we can represent by a function $V : T \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers.

We are now ready for a formal definition.

[3] When the arrow is an attack we call it a torpedo. When it is a support (see Section 4) we call it a booster. When it is both we call it an *actor*.

Definition 1. *Let A be a set of atomic nodes.*

1. *Define the notion of a torpedo based on A as follows:*
   - *$a \looparrowright b$ is a torpedo if $a, b \in A$.*
   - *$a \looparrowright x$ is a torpedo if $a \in A$ and x is a torpedo*
2. *Let T be a set of torpedoes and atomic nodes. We say that T is an attack network if the following holds*
   - *$x \looparrowright y \in T$ implies $x \in T$ and $y \in T$.*

   We say that T is finitely branching (in the outgoing direction) if for every $t \in T$ $\{a|(a \looparrowright t) \in T\}$ is finite.
3. *A valuation function on T is a function $V : T \to \mathbb{R}$.*
4. *An attack network with a valuation is a triple $N = (A, T, V)$, where A is a set of atomic nodes, T is an attack network based on A and V is a valuation on T.*
5. *Let $\mathbf{f}$ be a functional giving for each string of real numbers of the form $(y, x_1, \ldots, x_n, \varepsilon_1, \ldots, \varepsilon_n)$ a new real number $y' = \mathbf{f}(y, \bar{x}_i, \bar{\varepsilon}_i)$ (where $\bar{z}_i$ abbreviates $z_1, \ldots, z_n$, for $z = x$ or $z = \varepsilon$). Note that n is arbitrary. We assume $\mathbf{f}$ to be continuous and generally nice[4].*
   This will allow us more freedom in Definition 6 below.
   For example, let $\mathbf{f}(y, \bar{x}_i, \bar{\varepsilon}_i) = y \prod_{i=1}^{n}(1 - \varepsilon_i x_i)$. See Section 4.2 for more options.
6. *An argumentation attack model is a pair $(N, \mathbf{f})$, where N and $\mathbf{f}$ are as above.*

Definition 2. *Let us look at some examples. Consider Figure 9 in which a attacks b but also attacks its own attack. This is a case of a self defeating attack of a on b.*

We have $T = \{a, b, a \looparrowright b, a \looparrowright (a \looparrowright b)\}$ and

$$V(a) = x, V(b) = y,$$
$$V(a \looparrowright b) = \alpha \text{ and}$$
$$V(a \looparrowright (a \looparrowright b)) = \beta$$

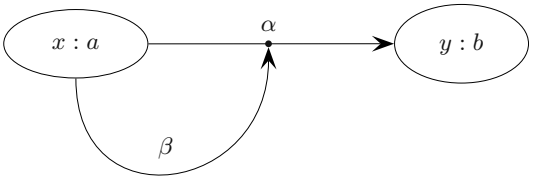


Fig. 9.

[4] By restricting $\mathbf{f}$ to finite sequences, we are forced to impose the condition of finitely branching on T in Definition 4 below. However, $\mathbf{f}$ can be more general, for example, we can take

$$\mathbf{f}'(y, S) = \inf\{\mathbf{f}(y, \bar{x}, \bar{\varepsilon}) \mid (\bar{x}, \bar{\varepsilon}) \in S\},$$

where S can now be an infinite set. This will allow us more freedom in Definition 6 below.

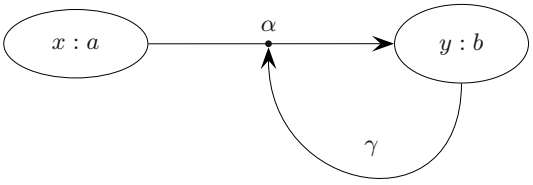


Fig. 10.

We can compare Figure 9 with Figure 10. In Figure 10 we can interpret γ *as a feedback loop, attacking and reducing* α*. The weaker the argument* b *is the less we want to spend effort attacking it.*

Definition 3 (Cycles). *Let* T *be an attack network. Define* $R_T \subseteq A^2$ *as follows:*

$$aR_Tb \text{ iff } a \rightarrowtail b \in T \text{ or for some } x \in A, a \rightarrowtail (x \rightarrowtail b) \in T.$$

Let R_T^* *be the transitive closure of* T*. We say* T *is syntactically acyclic iff there is no* $x \in A$ *such that* xR_T^*x*.*

If $N = (A, T, V)$*, we say* N *is syntactically acyclic if* T *is such.*

Example 5. Figure 11 is cyclic while Figure 12 is acyclic and finitely branching.

Example 6. Figure 13 is cyclic syntactically, but is acyclic semantically using V.

Note that although the network is syntactically cyclic, since $V(\beta) = 0$, it is as if $b \rightarrowtail a$ does not exist in T.

We shall deal with semantic acyclicity later.

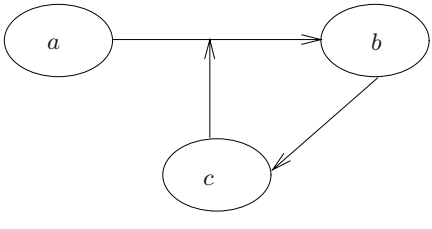


Fig. 11.

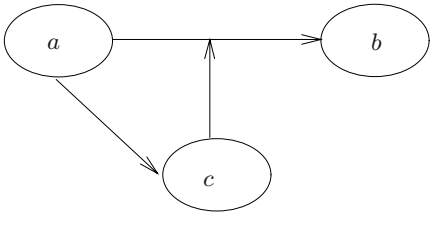
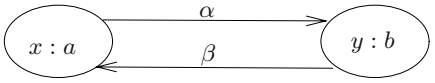


Fig. 12.



$V(\beta) = 0$

Fig. 13.

Definition 4 (Value propagation). *Let $(N, \mathbf{f})$ be a model, where N is acyclic and finitely branching. We shall propagate the values V through the model using* $\mathbf{f}$*. We do this in waves.*

Wave 0

An element $a \in T$ is said to be syntactically free of attack if for every $e \in A$ we have $(e \looparrowright a) \notin T$. Let it be said that the updated elements of Wave 0 are the free of attack elements and let the updated value V_0 be $V_0(a) = V(a)$, for an updated a of wave 0.

Wave $n + 1$

Assume we have defined the updated elements of waves $k \leq n$ and their updated value V_k. Let b be any element and let $a_1, \ldots, a_m$ be all, if any, elements of T such that $(a_i \looparrowright b) \in T$. Assume for each i, that a_i, as well as $a_i \looparrowright b$, were updated at some earlier wave $k_i \leq n$ and $l_i \leq n$ respectively.

Define

$$V_{n+1}(b) = \mathbf{f}(V(b), \bar{V}_{k_i}(a_i), \bar{V}_{l_i}(a_i \looparrowright b)).$$

When the network is finite, the algorithm terminates in quadratic time[5]*.*

Example 7 (Figure 5). Let us examine the network of Figure 5 again. We are listing the updated elements. Compare with Example 3.

Wave 0

$$w : d, x : a, \beta : d \looparrowright b,$$

$$\alpha : d \looparrowright c, \eta : c \looparrowright (a \looparrowright b).$$

Wave 1

$$z(1 - \alpha w) : c$$

Note that the only updated element in this wave is c. b is not updated because not all of its attackers (namely a) have been updated. In our earlier computation we did attack b at this stage, but we cannot do that under our current definition. We will not get a different result because our function $\mathbf{f}$ launches the attacks from separate nodes independently, cumulatively and commutatively.

Wave 2

$$\varepsilon(1 - \eta z(1 - \alpha w)) : a \looparrowright b.$$

Here $a \looparrowright b$ is being updated.

[5] Consider a linear network of n nodes with the following connection structure. Label the nodes from 1 to n. Node i attacks all nodes numbered $> i$. Wave 0 will have to search n nodes. Wave $i < n$ will have to search $n - i$ nodes. The sum of all waves is $n(n-1)/2$.

Wave 3
Now we can update b. We get

$$y(1-\beta w)(1-\varepsilon x(1-\eta z(1-\alpha w))) : b$$

Definition 5. *Let $(N, \mathbf{f})$ be a finite model. Propagate V using $\mathbf{f}$ in waves as defined above. Let the new valuation $V'(a)a \in T$ be the updated value of a. We call V' the result of the waves of attack in the network. Note that the propagation is executed only once.*

3 Handling Loops; Ecologies of Arguments

This section will discuss networks with loops. We have encountered loops in Example 1 (b) and (c). In Figure 2 of (b), we need to resolve the loop $\{?a, ?b\}$ in order to propagate values to c. So technically all we need is some assignment of values to a and to b, and then the algorithm of Definition 4 can be invoked. The values we give to the loop depend on our interpretation of it. Hints for possible interpretations can be obtained from other possible interpretations of the entire network regarded as a mathematical entity. We shall therefore open this section by putting forward several points of view as to the meaning of labelled networks and their internal loops, which will then lead to ways of dealing with their loops.

To begin our discussion, consider the following Figure 14.

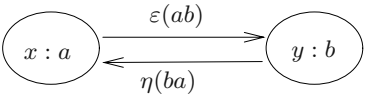


Fig. 14.

Let $\mathbf{f}_1(y, x, \varepsilon), \mathbf{f}_2(x, y, \eta)$ be the two transmission functions. We observe the following:

1. Figure 14 describes a syntactical loop.
2. Depending on the values x, y, ε, η and depending on the functions $\mathbf{f}_1$ and $\mathbf{f}_2$, Figure 14 might not be a loop semantically. For example, if $x : a$ is much stronger than $y : b$ or if $\varepsilon = 0$ then this might not be a loop.

3.1 Interpretation of Loops

There are various interpretations for the situation in Figure 14 besides our argumentation networks interpretation.

The Ecology Interpretation
The figure can be interpreted as an ecology. Species a feeds on species b and species b feeds on species a. The functions $\mathbf{f}_1$ and $\mathbf{f}_2$ give the success rates. This is a predator-prey situation.

Let V_n be the population of some species at generation n. We assume population growth is a discrete process taking place in cycles. Such biological examples are provided by many temperate zone arthropod populations, with one short-lived adult generation each cycle. One possible recurrence equation is the following

$V_{n+1} = V_n(1 + r(1 - \frac{V_n}{K}))$, where r and K are constants.

K is the maximum size for the population and r is a factor measuring depenence on the density of the population. The reader should compare this equation with the equation $V_b = \mathbf{f}_1(V_a, x, \varepsilon)$ arising from Figure 14. See [17, p. 324].

This equation is called *the non-linear logistic equation* which has the standard form

$$U_{n+1} = rU_n(1 - U_n), r > 0$$

This equation can exhibit chaotic behaviour depending on the value r, see [18].

A slightly different pair of equations has to do with parasitic life forms. Here we have, besides the population N_n, a parasitic population V_n. The recursive equations look like the following:

- $V_{n+1} = N_n - N_{n+1}/F$
- $N_{n+1} = FN_n f(N_n, V_n)$.

F is a factor indicating the proportion of those who escape the parasite. The difference between this equation for V_{n+1} and a direct recursion for V_{n+1} is that it is more complex. We get

- $V_{n+1} = N_n(1 - f(N_n, V_n))$
- $N_{n+1} = FN_n f(N_n, V_n)$

See [17, pp. 338].

Let us look at another example from biology. This is a model by M. P. Hasssell (1978) of two parasitoids (P and Q) and one host (N) model. The equations are (see [2, p. 295])

$$N_{t+1} = \lambda N_t f_1(P_t) f_2(Q_t)$$
$$P_{t+1} = N_t[1 - f_1(P_t)]$$
$$Q_{t+1} = N_t f_1(P_t)[1 - f_2(Q_t)]$$

where N, P and Q denote the host and two parasitoid species in generations t and $t + 1$, λ is the finite host rate of increase and the functions f_1 and f_2 are the probabilities of a host not being found by P_t or Q_t parasitoids, respectively. This model applies to two quite distinct types of interaction that are frequently found in real systems. It applies to cases where P acts first, to be followed by Q acting only on the survivors. Such is the case where a host population with discrete generations is parasitized at different developmental stages. In addition, it applies to cases where both P and Q act together on the same host stage, but the larvae of P always out-compete those of Q should multi-parasitism occur.

The functions f_1 and f_2 are:

$$f_1(P_t) = \left[1 + \frac{a_1 P_t}{k_1}\right]^{-k_1}$$
$$f_2(Q_t) = \left[1 + \frac{a_2 Q_t}{k_2}\right]^{-k_2}$$

where k_1 and k_2, a_1 and a_2 are constants.

To compare the biological model with the argumentation model, we put $a_1 = a_2 = 1$, $\lambda = 1$ and $k_1 = k_2 = -1$.
This gives

$$f_1(P_t) = 1 - P_t$$
$$f_2(Q_t) = 1 - Q_t$$

and therefore

$$N_{t+1} = N_t(1 - P_t)(1 - Q_t)$$
$$P_{t+1} = P_t N_t$$
$$P_{t+1} = Q_t N_t(1 - P_t)$$

giving us the appropriate functions for attack and counterattack for the situation in Figure 15:

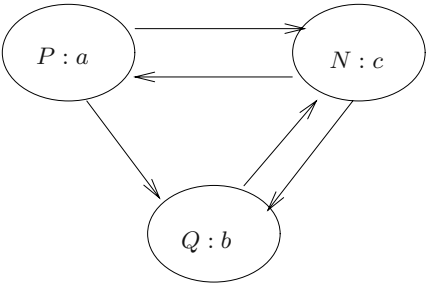


Fig. 15.

In Figure 15, a and b attack c. c counterattacks a and b and a attacks b. The transmission rates are 1. Since c is attacked by a and b, the new value for c is $N(1 - P)(1 - Q)$. Since a is counterattacked by c, the new value for a is PN.[6] Since b is counterattacked by c and attacked by a the new value for b is $QN(1 - P)$.

To give Figure 15 some meaning, think of a, b, c as follows:

$c =$ The US President has a strong case for re-election.
$a =$ A deteriorating situation in Iraq (US soldiers killed) (attacks his chances).
$b =$ Lack of success in combatting Al-Qaeda.

[6] See Example 3 as to why the counterattack value is PN and not $(1 - P)N$.

Clearly, if the value of c is low, less effort is required in attacking it from a and b. This explains the counterattack loop.

a attacks b by the argument that the situation in Iraq has diverted Al Qaeda away from US territory proper.

To sum up, we have shown a connection with biological models. In view of this connection we would like to refer to loops as *ecologies* (of arguments).

Modal Interpretations

We can read the nodes as possible worlds in a Kripke model and read the values as fuzzy truth values. ε is the fuzzy value of the accessibility of a to b (i.e. a *arrow* b means a is a posible world for b (i.e. bRa holds), while x is the fuzzy value of a being a possible world in the first place. So if $V_e(\varphi)$ gives a fuzzy value to the wff φ at world e, then $V_b(\Box\varphi) = \mathbf{f}(V_b(\varphi), \bar{V}_{a_i}(\varphi), \varepsilon_i)$, where a_i are all the nodes leading with an arrow into b.

It is worth giving a formal definition. See [3] for full details.

Definition 6.

1. *Let $\mathbb{L}$ be a propositional language with atoms $\{q_1, q_2, \ldots\}$, a modality $\Box$ and possibly other connectives $\mathbb{C}$. To fix our thoughts, say $\mathbb{C} = \{\Rightarrow, \neg\}$, where $\Rightarrow$ can be thought of as the Łukasiewicz many-valued implication (with truth values in [0,1] and 0 = true and truth table value $(A \Rightarrow B) = max(0, value(B) - value(A))$ and $\neg$ is a negation (with truth table value $(\neg A) = 1 - Value(A)$).*
2. *A modal network model $\mathbf{m}$ is a family of models $\mathbf{m}_q = (A, T, V_q, \mathbf{f}), q$ an atom of $\mathbb{L}$, such that each $\mathbf{m}_q$ is a finitely branching attack network model in the sense of Definiton 1. Thus in $\mathbf{m}$ A, T and $\mathbf{f}$ are fixed and V_q varies with q We assume that $\mathbf{f}, V_q$ give values in $[0, 1]$. We take $\mathbf{f}(y, x_i, \varepsilon_i) = \underset{i}{\mathsf{Sup}}\,(\varepsilon_i \Rightarrow x_i) = \underset{i}{\mathsf{Sup}}\, Max(0, x_i - \varepsilon_i)$.*
3. *For each $t \in T$ and each wff φ we define the value $V^n_\varphi(t)$, (for $n = 0, 1, 2, \ldots$) as follows:*
   (a) *$V^0_q(t) = V_q(t)$, for atomic q, and $t \in T$.*
   (b) *$V^{n+1}_q(t) = \mathbf{f}(V^n_q(t), \bar{V}^n_q(a_i), \bar{V}^n_q(a_i \nrightarrow t))$, where $a_1, \ldots, a_n$ are all the nodes such that $a_i \nrightarrow t \in T$.*
   (c) *$V^n_{A\Rightarrow B}(t) = Max(0, V^n_B(t) - V^n_A(t))$.*
   (d) *$V^n_{\neg A}(t) = 1 - V^n_A(t)$.*
   (e) *$V^n_{\Box A}(t) = V^{n+1}_A(t)$.*[7]
4. *We say $\mathbf{m}$ is stable iff for any wff A and any $t \in T$ there exists an n such that for all $m \geq n$ we have $V^n_A(t) = V^n_A(t)$. For stable models we can let $V^\infty_A(t) = \underset{i}{\mathsf{Lim}}\, V^n_A(t)$.*
5. *We call a stable model $(A, T, V^\infty_A, \mathbf{f})$, a fuzzy modal model for $\mathbb{L}$.*

[7] The reader should carefully note that we have huge scope here for defining a multitude of different modalities by choosing the dependence of $V^n_{\Box A}(t)$ on the set $\{V^{m+n}_A(t), m = 0, 1, \ldots\}$. What we here define is a **K**-type modality. We can also define the hypermodality of [7] by letting:

$$V^n_{\Box A}(t) = \begin{cases} V^{n+1}_A(t), \text{ for } n \text{ odd} \\ \mathrm{Max}(V^{n+1}_A(t), V^n_A(t)), \text{ for } n \text{ even} \end{cases}$$

Example 8 (Ordinary modal logic).

1. Let (S, R, h) be a traditional Kripke model for the language with $\{\rightarrow, \neg, \Box\}$, with S the set of possible worlds, R the accessibility relation and h the assignment to the atoms, (i.e. for each atomic $q, h(q) \subseteq S$). We assume that (S, R) is finitely branching, i.e. for each t the set $S_t = \{s|tRs\}$ is finite. Note that many modal logics are complete for a class of finitely branching models.
2. Let $A = S, T = S \cup \{a \looparrowright b|bRa\}$.
3. Let $V_q(a \looparrowright b) = 1$ for all atomic q and let $V_q(t) = 1$ iff $t \in h(q)$, for $t \in S$.
4. Let $\mathbf{f}(V(t), \bar{V}(a_i), \bar{V}(a_i \looparrowright t)) = 1$, where $a_1, \ldots, a_n$ are all nodes such that tRa_i holds, iff $\bar{V}(a_i) = 1$ for all $1 \leq i \leq n$.
5. We claim this model is stable. This can be proved by induction on the wff φ.
6. Note that we can get a new variety of modal logics by changing $\mathbf{f}$ from point to point, or by making $V^n_{\Box A}(t)$ dependent on $\{V^{n+m}_A(t) \mid m = 0, 1, 2 \ldots\}$ in a variety of ways.

Feedback Interpretation
We can consider the figures as a control-feedback situation. Say node b is a feedback for node a.

3.2 Unfolding Loops

There are various ways of treating loops.

- We can unfold them as done in, say, modal logic.
- We can let node a attack b, calculate the new value and then let b attack a, calculate the new value and then let a attack b and so on. This we call the parasite way of unfolding a loop.
- We can let a and b attack each other simultaneously, calculate the new values and then let them attack again and again. This is the predator-prey way of unfolding a loop.

Let us now turn to Figure 14 and see what are our options for dealing with this loop.

Our first attempt at a solution is to regard (ab) and (ba) as the same channel and read the loop as feedback loops. So a pushes εx towards b and b pushes ηy towards a. The net result is $(\varepsilon x - \eta y)$ in the direction of the positive value. So assuming $\varepsilon x \geq \eta y$ we get that Figure 14 is essentially reduced to Figure 16.

The solution is not satisfactory. It cannot deal with cases like Figure 2 unless we further commit the model to be a proper network flow model with various capacities, as studied in operational research. So let us try another approach. Assume in Figure 14 that we have $x = \eta = \varepsilon = y$.

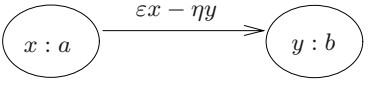


Fig. 16.

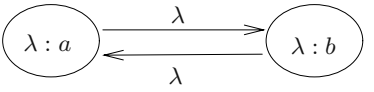


Fig. 17.

Call the common value λ. We now get Figure 17.

Let $V_0(a) = V_0(b) = \lambda$, the initial value, and let us transmit from a to b and back from b to a in cycles and see what presents itself. This is the modal logic approach.

We treat Figure 17 as equivalent to Figure 18 below:

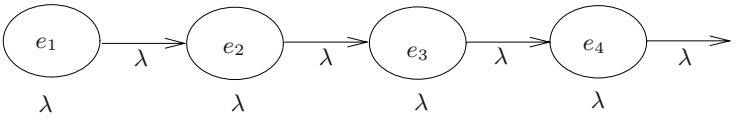


Fig. 18.

In Figure 18, nodes $e_1, e_3 \ldots$ represent node a of Figure 17 and needs $e_2, e_4 \ldots$ represent node b. So we start from $V_1(e_1) = \lambda$ and transmit to the right getting $V_n(e_n), n = 2, 3, \ldots$.

Step 1: Transmit λ^2 to e_2 to get $V_2(e_2) = \lambda(1 - \lambda^2) = \lambda - \lambda^3$.

Step 2: Transmit from e_2 to e_3 the value $\lambda V_2(e_2)$ and get $V_3(e_3) = \lambda(1 - \lambda V_2(e_2)) = \lambda - \lambda^3 + \lambda^5$.

We can continue by induction.

Lemma 1. *Suppose we have a node e with* $V(e_n) = V_{\lambda,n} = \lambda - \lambda^3 + \lambda^5 - \ldots + (-1)^n \lambda^{2n+1}$ *and suppose we are transmitting to a node* $\lambda : e_{n+1}$ *with value* λ *then we get* $V(e_{n+1}) = V_{\lambda,n+1}$.

Proof.

$$\begin{aligned} V(e_{n+1}) &= \lambda(1 - \lambda V(e_n)) \\ &= \lambda - \lambda^3 + \lambda^5 \cdots - \lambda^2(-1)^n \lambda^{2n+1} \\ &= \lambda - \lambda^3 + \lambda^5 \ldots + (-1)^{n+1} \lambda^{2(n+1)+1} \end{aligned}$$

We now observe that when n goes to infinity, we get $V_{\lambda,\infty} = \dfrac{\lambda}{1+\lambda^2}$.[8]

This means that Figure 17 stabilises into Figure 19. Note that the transmission rates in Figure 19 are all 0. This is because the values $V_{\lambda,\infty}$ obtained have already taken into account all recursive transmissions.

[8] Note that if we solve the fixed point recursion equation $V_{\lambda,\infty} = \lambda(1 - \lambda V_{\lambda,\infty})$, we get $V_{\lambda,\infty} = \frac{\lambda}{1+\lambda^2}$.

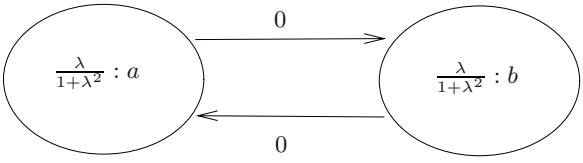


Fig. 19.

Note that Figure 18 represents one way of going through the cycle of Figure 17, i.e. the fuzzy modal logic approach. Another approach is what we called the parasite model, where we apply the transmission on Figure 17 directly, starting from node a to b with $V_1(a) = \lambda$, (corresponding to $V_1(e_2)$) we would get $V_2(b) = \lambda(1 - \lambda^2)$, same as $V_2(e_2)$ and then transmit back to node a and get $V_3(a) = V_1(a)(1-\lambda V_2(b))$ (corresponding to $V_3(e_2)$). So far the values agree, but now there is a difference. Working directly on Figure 17 we transmit $1-\lambda V_3(a)$ to node b whose *last* value is $V_2(b) = \lambda(1-\lambda^2)$ and get $V_4(b) = \lambda(1-\lambda^2)(1-\lambda V_3(a))$. While in Figure 18, the value of node e_4 (which corresponds to b) is λ and so we get in Figure 18 $V_4(e_4) = \lambda(1 - \lambda V_3(e_3))$. So the question is, as we go through the cycle $a \to b \to a \to b \ldots$, do we use the new value or follow Figure 18 and keep the value at λ, the initial value!

Another possibility for dealing with Figure 17 is to adopt the predator-prey model and transmit simultaneously from node a to node b and from node b to node a, and then repeat the cycle. If $V_0(a) = V_0(b) = V_0 = \lambda$ is the initial value, then symmetry is maintained through the cycles and for step $n + 1$ we get

$$V_{n+1}(a) = V_{n+1}(b) = V_{n+1} = V_n(1 - \lambda V_n).$$

So we end up with a recursive equation

- $V_0 = \lambda, 0 \leq \lambda \leq 1$
- $V_{n+1} = V_n(1 - \lambda V_n)$

which for $0 \leq \lambda \leq 1$ gives $V_\infty = 0$, meaning that a and b cancel each other[9].

The above considerations can be applied to other loops. The net result of Figure 2 will be similar to that of Figure 17.

Consider Figure 20. Similar considerations using the Lemma indicate that Figure 20 stabilises as Figure 21

We can make one more move now. To resolve Figure 2, we consider Figures 20 and 21 and let λ approach 1. Thus we get the value $\frac{1}{2}$. Hence the net reults of Figure 2 is Figure 22 below.

A similar net result obtains for Figure 23 below.

Note that now we can resolve the loop in Figure 3. We get $V(a) = V(b) = \frac{1}{2}$ and therefore $V(c) = \frac{3}{4}$ and hence $V(d) = \frac{1}{4}$.

We can also deal with an argument attacking itself. It will get $\frac{1}{2}$.

There is still work to be done on resolving loops. We need to show the following.

[9] The fixed point recursion equation for this case is $V = V(1 - \lambda V)$, yielding $V = 0$.

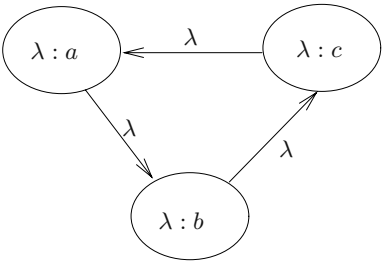


Fig. 20.

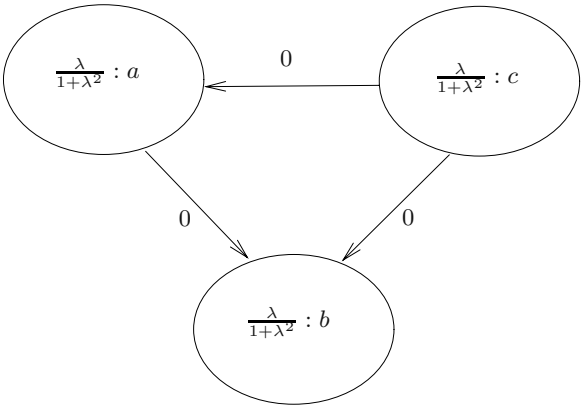


Fig. 21.

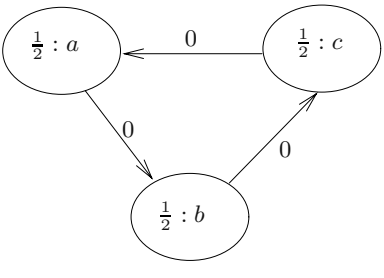


Fig. 22.

1. How the results we get for the loop depend on the choice of numbers we assign to the nodes and for the transmission rates (we gave λ to all!).
2. What happens when loops can be resolved but we use our method anyway, as in Figure 24.
   In Figure 24, the net result is

$$\{+c, -b, +a\}.$$

   What do we get if we assign λ everywhere and get Figure 25?

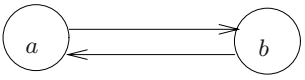


Fig. 23.

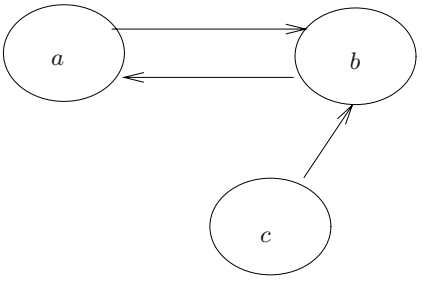


Fig. 24.

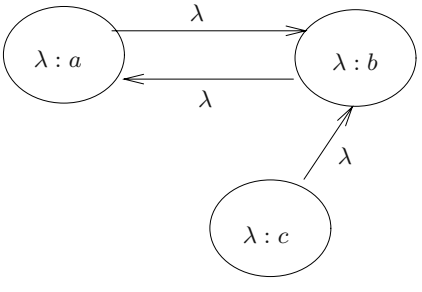


Fig. 25.

Here is the calculation: We start with $V_0(a) = V_0(b) = V_0(c) = \lambda$. Transmit from c to b and get $V_1(b) = \lambda - \lambda^3$. Transmit from b to a and get $V_1(a) = \lambda(1 - \lambda V_1(b)) = \lambda - \lambda^3 + \lambda^5$.

Obviously if we follow the loop we get as before $V_\infty(a) = V_\infty(b) = \frac{\lambda}{1+\lambda^2}$ and the net result is $\{1 : c, \frac{1}{2} : a, \frac{1}{2} : b\}$. This is not satisfactory.

It makes more sense to try to give c value 1 transmitting at rate 1, since c is not in a loop. This will give b value 0 and a value λ. When λ approaches 1 we get the right answer.

Perhaps we might follow the procedure of giving λ only to nodes in a loop?

3. Consider, however, the following loop in Figure 26.

   d is attacked twice and is attacking once, while a is attacking twice and is attacked once. Should we give them λ in the same way?

Example 9 (Resolving Figure 26). Let us try the fixed point approach on Figure 26. We begin with $V_0(a) = V_0(b) = V_0(c) = V_0(d) = y$ and with transmission λ.

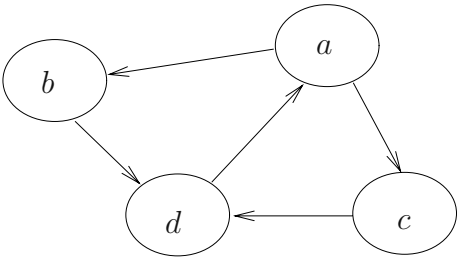


Fig. 26.

(a) We start propagating from node a. We get

$$V_1(b) = V_1(c) = y(1 - \lambda y).$$
$$V_1(d) = y(1 - \lambda y(1 - \lambda y)^2)$$

and therefore

$$V_1(a) = y(1 - \lambda V_1(d)).$$

We need a fixed point solution to $V_1(a) = V_0(a)$. Hence

$$y(1 - \lambda V_1(d)) = y.$$

Excluding $y = 0$, we get

$$1 - \lambda V_1(d) = 1.$$

Hence

$$V_1(d) = 0.$$

This means

$$y(1 - \lambda y(1 - \lambda y)^2) = 0.$$

Hence

$$\lambda y(1 - \lambda y)^2 = 1.$$

Let $x = \lambda y$. We get $x(1-x)^2 = 1$. This has a solution, x_0 of approximate value

$$x_0 \approx 1.755.$$

If we want $0 \leq \lambda \leq 1$ then there is no way $0 \leq y \leq 1$. Hence the only fixed point solution is $y = 0$.

(b) Let us start at node d of Figure 26

$$V_0(d) = y$$
$$V_1(a) = y(1 - \lambda y)$$
$$V_1(b) = V_1(c) = y(1 - \lambda V_1(a))$$
$$V_1(d) = y(1 - \lambda V_1(b))^2$$

and try to solve the fixed point equation:

$$y = y(1 - \lambda V_1(b))^2.$$

Hence if we insist on $y \neq 0$,

$$1 = (1 - \lambda V_1(b))^2.$$

Hence

$$1 - \lambda V_2(b) = \pm 1.$$

So either

i. $V_1(b) = 0$
   or
ii. $V_1(b) = 2\lambda$.

For $V_1(b) = 0$ we get, if $y \neq 0$, that $V_1(a) = 1/\lambda$.
Hence $\lambda y(1 - \lambda y) = 1$. It is clear that this equation has no real solution.
Let us now try the case in which $V_1(b) = 2\lambda$.
Hence

$$y(1 - \lambda y(1 - \lambda y)) = 2\lambda$$

$$y - \lambda y^2(1 - \lambda y) = 2\lambda$$

$$y - \lambda y^2 + \lambda^2 y^3 - 2\lambda = 0$$

Does this have solutions? Remember $0 \leq \lambda \leq 1, 0 \leq y \leq 1$.
If we choose $\lambda = 0.133$ and $y = 0.275$, the value of the polynomial is 0.0006.
Since we are dealing with continuous functions, we can find proper solutions.
Let us now try another way of tackling Figure 26, which can be rewirtten as Figure 27 below, where a_i represent a, b_i represent b, c_i represent c and d_i represent d.

The neural net approach gives us an additional dimension. We can run the cycles in the loop but also transmit to the rest of the network, and possibly stop after so many cycles (say $n = 100$) and examine the values in all nodes of other network. If the time involved in the cycles has meaning in terms of the network itself changing in time (as modelled in Section 4 below), then we have added a new and interesting dimension to loops in these networks.

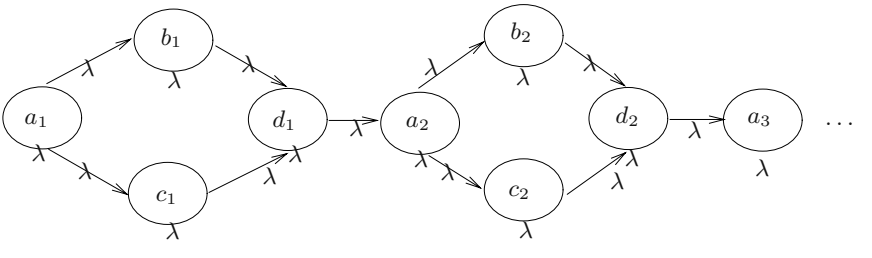


Fig. 27.

In other words, we are saying that attacks take time to be executed, a loop of the form "a attacks b and b attacks a" also takes time to unfold, and meanwhile the network can change.
To give an example of such a loop, think of contradicting witnesses and circumstantial evidence, one supporting a and one supporting $b = \neg a$. So the loop is as in Figure 28

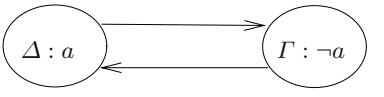


Fig. 28.

where Δ, Γ are themselves argument structures which are time dependent. This loop certainly takes time to unfold! There may be some facts in Δ or Λ that take time to verify or refute!

The general treatment of loops should be done in the context of neural networks (see [8]), not because of a conceptual connection, but because these nets can technically reach equilibrium and resolve loops of the kind that arise there.

Note that every graph can be presented as an acyclic graph of nodes which are themselves maximally connected cycles. So when we are dealing with cycles we can make use of that.

4 Attack and Support Networks

This section discusses the addition of support arrows to argumentation networks. We will see that in order to have equal attack and support cancel each other, we need to reconsider the way we calculate the values of attacks (and supports). We offer a new definition and establish a connection between the new definition, the Dempster–Shafer rule, and surprisingly, the Cross-Ratio and projective metric distance from geometry.

4.1 Discussion of Support

Consider a connection from a to b in Figure 29.

The double arrow indicates support. The simplest way to do it is to attack $(1 - y)$ which is the distance of b from 1.[10] Thus the new value of b is

$$\begin{aligned} &1 - (1 - y)(1 - \lambda x) = \\ &1 - [1 - \lambda x - y + \lambda xy] \\ &= \lambda x + y - \lambda xy = y + (1 - y)\lambda x \end{aligned}$$

If we have several supports, then $(1 - y)$ shrinks to

$$(1 - y)(1 - \lambda_1 x_1)(1 - \lambda_2 x_2) \ldots (1 - \lambda_k x_k)$$

[10] This is Bernouli's rule of combination, see [19, pp. 75–76].

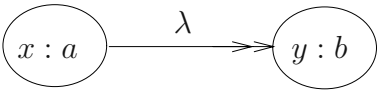


Fig. 29.

and the new value y' becomes $1-(1-y)\prod_i(1-\lambda_i x_i)$. The difference $y'-y$ becomes

$$\begin{aligned}\Delta^+ y &= 1-(1-y)\textstyle\prod_i(1-\lambda_i x_i)-y\\ &= (1-y)(1-\textstyle\prod_i(1-\lambda_i x_i))\end{aligned}$$

and we have

$$y' = y + \Delta^+ y.$$

How do we deal with both attack and support? Consider Figure 30. In this figure $x:a$ attacks $y:b$ and $z:c$ supports it. So the new value for b is

$$y - \lambda xy + \mu z(1-y).$$

It is not clear what to do with several simultaneous attacks and supports. The model must be commutative in the order of application.

Our solution is simple. b is at a distance y from 0 and distance $1-y$ from 1. Let the attackers attack y to get it nearer to 0 and let the supporters attack $(1-y)$ to get b nearer to 1. Thus if $x_i:a_i$ attack $y:b$ with transmission λ_i and $z_i:c_i$ support $y:b$ with transmission μ_i we get y' as the new value at b, where

$$\begin{aligned}y' &= y - \Delta^- y + \Delta^+ y\\ &= y - y(1-\textstyle\prod_i(1-\lambda_i x_i))\\ &\quad +(1-y)(1-\textstyle\prod_i(1-\mu_i z_i))\\ &= y\textstyle\prod_i(1-\lambda_i x_i) + (1-y)(1-\prod_i(1-\mu_i z_i))\end{aligned}$$

Note that there is something numerically wrong with our proposal. In Figure 29, if we let $z=x$ and $\mu=\lambda$, i.e. the attack and support have the same values, then, we would have expected that they cancel each other. However, this is not the case. The new value is $y' = y - 2\lambda xy + \lambda x$.

This should not surprise us. The closer y is to 1, the less is the numerical value of an attack on $1-y$, and the more numerical value we get for an attack on y. So, for example, assume $y=0.9$ in value. Then a support of 50% of y will

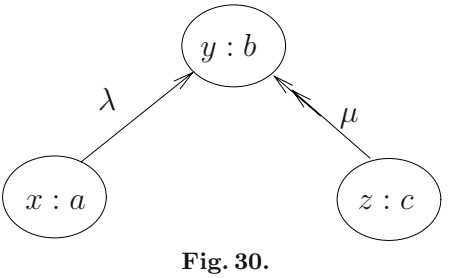


Fig. 30.

half the distance of y from 1, i.e. will yield $\Delta^+ = 0.005$ in numerical value, while in comparison, a 50% attack on y will half the distance of y from 0 and will yield $\Delta^- = 0.45$. The net result of simultaneous attack and support will yield the new value $0.9 - 0.45 + 0.05 = 0.50$.

Can we remedy the situation? Perhaps we should attack by changing the ratio $r(y)$ of y to $1 - y$, (i.e. $r(y) = y/(1 - y)$, and then calculate the new y' which will give the new ratio. So suppose the transmitted value (of attack or support) is $0 \leq \theta \leq 1$.

If θ is an attack we want to reduce $r(y)$ and so we let $r'(y) = \theta r(y)$. If θ is a support, we want to increase y, so the new ratio is $r'(y) = r(y)/\theta$.

We now solve the equation

$$\frac{y'}{1 - y'} = r'(y)$$

and therefore we get

$$y' = \frac{r'(y)}{1 + r'(y)}.$$

We must now decide on what value θ to use. Let us use the same value we used before, as agreed in Example 3. In Figure 29, we have $x : a$ attacking $y : b$ with transmission rate λ and we therefore have $\theta = (1 - \lambda x)$.

Let us calculate the values of attack and support with θ.

Case of Attack

$$\begin{aligned} r'(y) &= \frac{y(1 - \lambda x)}{1 - y} \\ y' &= \frac{y(1 - \lambda x)}{(1 - y)(1 + \frac{y(1 - \lambda x)}{1 - y})} \\ &= \frac{y(1 - \lambda x)}{(1 - y + y - \lambda xy)} \\ &= \frac{y(1 - \lambda x)}{(1 - \lambda xy)} \end{aligned}$$

Case of Support

$$\begin{aligned} r'(y) &= \frac{y}{(1 - y)(1 - \lambda x)}. \\ y' &= \frac{y}{(1 - y)(1 - \lambda x)(1 + y/(1 - y)(1 - \lambda x))} \\ y' &= \frac{y}{(1 - y)(1 - \lambda x) + y} \\ &= \frac{y}{(1 - \lambda x + y\lambda x)} \\ &= \frac{y}{1 - \lambda x + y\lambda x} \\ &= \frac{y}{1 - \lambda x(1 - y)} \end{aligned}$$

Let us now assume as before that the attack is 50%, e.g. $x = 0.5, \lambda = 1$.

We get $\theta = 0.5$. Assume as before $y = 0.9$. Hence $r'(y) = \frac{0.9}{0.1}.0.6 = 4.5$ and $y' = \frac{4.5}{1+4.5} = \frac{4.5}{5.5} = \frac{9}{11}$.

This should be compared with the previously attained value $0.45 = \frac{9}{20}$.

For the support we get

$$r'(y) = \frac{0.9}{0.1.0.5} = 18$$

So

$$y' = \frac{18}{1+18} = \frac{18}{19}$$

This should be compared with the value 0.05 we got previously.

How do we handle simultaneous attacks and supports? We follow the same principle as before. If $\theta_1, \ldots, \theta_n$ are attacking values and $\theta'_1, \ldots, \theta'_m$ are the supporting values then the new $r'(y)$ is

$$r'(y) = r(y)\frac{\prod_i \theta_i}{\prod_i \theta'_i}.$$

It is worthwhile comparing the recursion results we obtained with the kind of recursion one gets in mathematical biology. We use the table (Table 3.1) on [20, p. 53].

1. Old attack formula
$$y_{n+1} = y_n(1 - \lambda x)$$
This can be compared with exponential population growth.
2. New attack formula
$$y_{n+1} = \frac{y_n(1 - \lambda x)}{1 - \lambda x y_n}$$
This can be compared with the Beverton–Hort formula in the table of [20, p. 53].

Let us also examine what happens in case of loops. Consider Figures 17 and 18 again. We have $v_1(e_1) = \lambda$ and the recursion equation, according to Figure 18 is

$$V_{n+1}(e_{n+1}) = \frac{\lambda(1 - \lambda V_n(e_n))}{1 - \lambda^2 V_n(e_n)}.$$

The recursion fixed point equation for this case is

$$V = \frac{\lambda(1 - \lambda V)}{1 - \lambda^2 V}$$

or

$$V - \lambda^2 V^2 = \lambda - \lambda^2 V$$

$$\lambda^2 V^2 - \lambda^2 V - V + \lambda = 0$$

$$V^2 - \frac{(1+\lambda^2)}{\lambda^2} V + \frac{1}{\lambda} = 0$$

Let λ approach 1, we get

$$V^2 - 2V + 1 = 0$$

and so $V_\infty = 1$.
If we do the recursion proper, as in Figure 17, we get

$$V_{n+1} = \frac{V_n(1 - \lambda V_n)}{1 - \lambda V_n^2}$$

The fix point equation becomes

$$V(1 - \lambda V^2) = V(1 - \lambda V).$$

If we discard the solution $v = 0$, we get

$$1 - \lambda V^2 = 1 - \lambda V$$

hence

$$V^2 = V$$

and hence $V = 1$.

4.2 Connection with Metric Projective Geometry and the Dempster–Shafer Rule

In the previous subsection, we agreed that in the situation of Figure 30, node a attacks node b by attacking the ratio:

$$r(y) = \frac{y}{1 - y}$$

We proposed that the attack value θ be $\theta = 1 - \lambda x$. We want in this subsection to re-examine our decision and see whether we want to use a different attack value. First to simplify our qualitative consideration, assume $\lambda = 1$ and $\mu = 1$. Second, let us focus on node c, which is supporting node b, with value z. Assume that z is very small, almost 0. One may feel that in many real applications, a very limited support is worse than nothing. It implies an attack on argument b, the hidden implication is that if b were any good why isn't c's support of it a bit stronger? This way of thinking would integrate the support and attack together. So if a node supports another node with value z then it simultaneously attacks it with value $1 - z$. If $z = 1$, then the support is complete. If $z \approx 0$ then the support is insulting and really accomplishes an attack to the value of $1 - z$.

Let us look at Figure 30 again. There are two ways to look at this figure (with $\lambda = \mu = 1$). One way is that we have two nodes, $x : a$ and $z : c$, the first attacking the node $y : b$ and the second supporting it.

The other way is that there is a single node $z : c$ supporting the node $y : b$, but simultaneously attacking it to the value $1 - z$, as discussed above. Figure 30,

with $x = 1-z$ is a representation of this new point of view through the additional node $x = (1-z) : a$.

Of course it is nicer to represent this new point of view directly, and indeed, Figure 32 represents this new point of view of support/attack mode by a double arrow.

Let us now calculate the new value y' of the attack and support configuration of Figure 30. We have:

$$r'(y) = \frac{y(1-x)}{(1-y)(1-z)}$$

Hence

$$\begin{aligned} y &= \frac{r'(y)}{1+r'(y)} \\ &= \frac{y(1-x)}{(1-y)(1-z)+y(1-x)} \\ &= \frac{y(1-x)}{1-y-z+y(z+1-x)} \end{aligned}$$

In order to compare with a later formula, let us rename the values. Let $z_2 = 1-x$ and let $z_1 = z$. We get the equation (DS1) below:

$$(DS1) \qquad y' = \frac{yz_2}{1-y-z_1+y(z_1+z_2)}$$

This equation means that a node $y : b$ is simultaneously supported by $z_1 : c$ and attacked by $(1-z_2) : a$. Alternatively, we can say that the node is being [Support, Attacked] by the pair $[z_1, z_2]$. If $z_1 \leq z_2$ (i.e. $z + x \leq 1$), we can say we have a [Support, Attack] interval $[z_1, z_2], 0 \leq z_1 \leq z_2 \leq 1$.[11]

We adopt this terminology in preparation for the Dempster–Shafer point of view, yet to come. See item 3 of Example 10.

Let us now examine the case where $x = 1-z$, i.e. $z = 1-x$. We can view this as a [Support, Attack] pair $[z_1, z_2] = [z, z]$.[12]

We can view Figure 30 again and see that we are getting a situation of support value z from node c and attack value $1-z$ from node a.

We have already calculated the new ratio $r'(y)$ for node b, it is

$$r'(y) = \frac{y(1-(1-z))}{(1-y)(1-z)} = \frac{yz}{(1-y)(1-z)}$$

[11] Actually the intervals involved are $[0, z_1], [1-z_2, 1]$.

[12] Beware some possible confusion in notation. In Figure 30, the attack of a node is with value $x = 1-z$ and the support is with value z. If we regard Figure 30 as representing the [Support, Attack] double arrow of Figure 32, we write it as $[z, 1-x] = [z, z]$ and *not as* $[z, x]$. This is because $z_2 = (1-x)$ appears in (DS1).

Let us write this equation as

$$(*) \qquad r'(y) = \frac{y/(1-y)}{(1-z)/z}$$

We now calculate the new value y', it is

$$(**) \qquad \begin{aligned} y' &= \frac{r'(y)}{1+r'(y)} \\ &= \frac{yz}{(1-y)(1-z)+yz} \end{aligned}$$

We thus get that a node $z : c$ supporting a node $y : b$ yields the new value $y' : b$, where:

$$(DS2) \qquad y' = \frac{yz}{1-y-z+2yz}$$

provided, of course, that $y + z - 2yz \neq 1$.

Let us say that $(*)$ and (DS2) represent a combined [Support, Attack] result of a node to a value $[z, z]$, attacking a node with value y.

We now connect (DS2) to the Dempster–Shafer rule (see [19, 15]), and to the Cross-Ratio and projective metric from geometry (see [1, 6]).

Example 10 (Dempster–Shafer rule). The range of values we are dealing with is the set of all subintervals of the unit interval [0,1]. The Dempster–Shafer addition on these intervals is defined by

$$[a, b] \oplus [c, d] = [\frac{a \cdot d + b \cdot c - a \cdot c}{1-k}, \quad \frac{b \cdot d}{1-k}]$$

where $k = a \cdot (1-d) + c \cdot (1-b)$, where '$\cdot$', '$+$', '$-$' are the usual arithmetical operations. The compatibility condition required on a, b, c, d is

$$\varphi([a, b], [c, d]) \equiv k \neq 1.$$

The operation $\oplus$ is commutative and associative. Let $\mathbf{e} = [0, 1]$.

The following also holds:

- $[a, b] \oplus \mathbf{e} = [a, b]$
- For $[a, b] \neq [1, 1]$ we have $[a, b] \oplus [0, 0] = [0, 0]$
- For $[a, b] \neq [0, 0]$ we have $[a, b] \oplus [1, 1] = [1, 1]$
- $[a, b] \oplus [c, d] = \varnothing$ iff either $[a, b] = [0, 0]$ and $[c, d] = [1, 1]$ or $[a, b] = [1, 1]$ and $[c, d] = [0, 0]$.

In this algebra, we understand the transmission value $[a, b]$ as saying that the actual transmission value lies in the interval $[a, b]$.

Let us make three comments:

1. Let x denote $[x, x]$. We get for $0 \leq a \leq 1$ and $0 \leq c \leq 1$ the following

$$a \oplus c = \left[\frac{ac + ac - ac}{1 - a(1-c) - c(1-a}\,,\ \frac{ac}{1 - a(1-c) - c(1-a)}\right]$$

$$= \left[\frac{ac}{1 - a - c + 2ac}, \frac{ac}{1 - a - c + 2ac}\right]$$

$$= \frac{ac}{1 - a - c + 2ac}$$

provided $(a + c - 2ac) \neq 1$.
We note immediately that (DS2) is $y \oplus z$. This is also the propagation method used by the MYCIN expert system. See [16].

2. Let us check for what values of a, c can we have equality, i.e. when can we have $a + c - 1 = 2ac$?
Assume $a \leq c$.
We claim the only solution to the equation $a + c - 2ac = 1$ is $a = 0, c = 1$ for $a \leq c$ and $a = 1, c = 0$ for the case $c \leq a$. There is no solution for $c = a$.
To show this, let $c = a + \varepsilon, 0 \leq \varepsilon \leq c - a$.
Then assume

$$\begin{aligned} a + a + \varepsilon &= 1 + 2a(a + \varepsilon) \\ 2a + \varepsilon &= 1 + 2a^2 + 2\varepsilon a \\ \varepsilon - 2\varepsilon a &= 1 + 2a^2 = 2a \\ \varepsilon(\tfrac{1}{2} - a) &= a^2 - a + \tfrac{1}{2} \\ &= (a - \tfrac{1}{2})^2 - (\tfrac{1}{2})^2 + \tfrac{1}{2} \\ &= (a - \tfrac{1}{2})^2 + (\tfrac{1}{2})^2 \end{aligned}$$

Hence

$$\begin{aligned} &(a - \tfrac{1}{2})^2 + \varepsilon(a - \tfrac{1}{2}) + (\tfrac{1}{2})^2 = 0 \\ &[(a - \tfrac{1}{2}) + \tfrac{\varepsilon}{2}]^2 - (\tfrac{\varepsilon}{2})^2 + (\tfrac{1}{2})^2 = 0 \\ &(a - \tfrac{1}{2} + \tfrac{\varepsilon}{2})^2 = (\tfrac{\varepsilon}{2})^2 - (\tfrac{1}{2})^2 \\ &\quad = ((\tfrac{\varepsilon}{2} - \tfrac{1}{2})(\tfrac{\varepsilon}{2} + \tfrac{1}{2}) \end{aligned}$$

Hence $\varepsilon = 1$ and since $0 \leq c = a + \varepsilon \leq 1$ we must have $a = 0$ and $c = 1$.
In particular, we get that for $a = c = x$, $x \oplus x$ is always defined and we have

$$x \oplus x = \frac{2x^2}{(x - \frac{1}{2})^2 + (\frac{1}{2})^2}$$

For example, we have

$$0 \oplus 0 = 0$$

$$1 \oplus 1 = 1$$

$$\tfrac{1}{2} \oplus \tfrac{1}{2} = 1$$

3. Let us check what happens when $c = d$.
   We get

$$\begin{aligned}[a,b] \oplus c &= \frac{bc}{1 - a(1-c) - c(1-b)} \\ &= \frac{bc}{1 - a + ac - c + bc} \\ &= \frac{bc}{1 - a - c + c(a+b)}\end{aligned}$$

   The reader should compare this equation with the formula (DS1) obtained before.

Example 11 (Cross-Ratio). Consider the interval $[0, 1]$ and two points y and $1-z$ in this interval. Let $A = 0, B = 1, C = y$ and $D = 1-z$. Taking AC, CB, AD, DB as directed intervals, we have it that $AC = y, CB = 1 - y, AD = 1 - z$ and $DB = z$.

The projective Cross-Ratio between these points, denoted traditionally by $(A, B; C, D)$ is calcualted as the ratio of ratios of the directed intervals.

$$(A, B; C, D) = \frac{AC/CB}{AD/DB} = \frac{y/(1-y)}{(1-z)/z} = \frac{yz}{(1-y)(1-z)}$$

Note that this is formula $(*)$.

Note that this measures distance. In the Cayley–Klein metric, $\log(AB; CD)$ is used to describe the distance between points C and D. Figure 31 shows how it is done.

C and D are inside the unit circle. The chord connecting them meets the circle at A and B. See [1, Sections 4.10 and 11.7] and [6, Chapter 6].

Returning to Figure 30, we have

$$(*) \qquad r'(y) = (0, 1; y, 1 - z)$$

We can now define a new kind of support/attack arrow (with value $z/1-z$) in a network, as displayed in Figure 32 by double arrow

We have for $0 \leq y, z \leq 1$

(♯1) $r(y) = \dfrac{y}{1-y}$

(♯2) $r'(y) = (0, 1; y, 1 - z)$

(♯3) $y' = \dfrac{yz}{1 - y - z + 2yz} = y \oplus z$

(♯4) Furthermore, a formula (DS1) for a combined support to value z_1 and attack to value z_2, as in Figure 30 gives the result $y' = y \oplus [z_1, z_2]$.

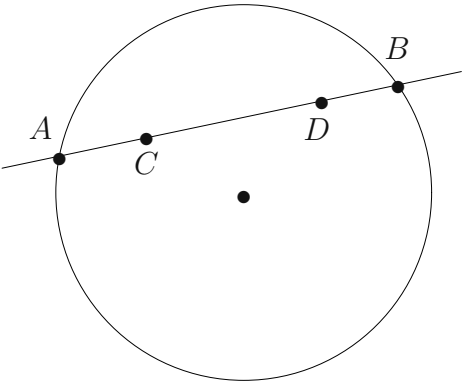
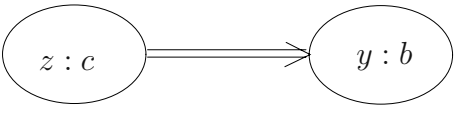


Fig. 31.



provided $y + z - 2yz \neq 1$.

Fig. 32.

Equations (♯2) and (♯3) and (♯4) open new opportunities for us.

1. Allow for values to be intervals because of the connection with Dempster–Shafer.
2. Allow for a connection with a more general non-Euclidean metric, using complex numbers.
3. Attack and support values need not be in $[0, 1]$.

We shall investigate these further.

Example 12 (Cross-Ratio for intervals). This example will try to extend the notion of Cross-Ratio for intervals, i.e. we look for Cross-Ratio for $(0, 1; [y_1, y_2], 1 - z), 0 \leq y_1 \leq y_2; 0 \leq z \leq 1$.

We saw that the situation in Figure 33 can be described as follows:

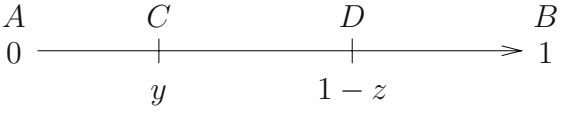


Fig. 33.

1. $$r(y,z) = (0,1;y,1-z)$$
$$= \frac{yz}{(1-y)(1-z)}$$

2. We also know that the Dempster–Shafer rule for the case of $y \oplus z = [y,y] \oplus [z,z]$ gives the value
$$y \oplus z = \frac{r}{1+r} = \frac{yz}{1-y-z+2yz}$$

3. Our aim is to define Cross-Ratio $(0,1;[y_1,y_2],1-z)$. We use (2): Consider
$$[y_1,y_2] \oplus z = \frac{y_2 z}{1-y_1-z+z(y_1+y_2)}$$

4. Define by analogy with (2):
$$(*1) \qquad [y_1,y_2] \oplus z = \frac{r([y_1,y_2],z)}{1+r([y_1,y_2],z)}$$
we do not know what $r^* = r([y_1,y_2],z)$ means. However, using $(*1)$ and solving for r^* we get:
$$r^* = \frac{[y_1,y_2] \oplus z}{1-[y_1,y_2] \oplus z}$$
Fortunately, the expressions in the right-hand side are all numbers: Hence we get
$$r^* = \frac{y_2 z}{1-y_1-z+z(y_1+y_2)(1-\frac{y_2 z}{1-y_1-z+z(y_1+y_2)})}$$
$$= \frac{y_2 z}{1-y_1-z+zy_1+zy_2-y_2 z}$$
$$= \frac{y_2 z}{1-y_1-z+zy_1}$$
$$= \frac{y_2 z}{(1-y_1)(1-z)}$$
$$= \frac{y_2}{y_1} \cdot \frac{y_1 z}{(1-y_1)(1-z)} = \frac{y_1}{y_1} r(y_1,z)$$
We therefore have
$$(*2) \qquad r([y_1,y_2],z) = \frac{y_2}{y_1} r(y_1,z).$$

We can therefore define

$$(*3) \qquad (0,1;[y_1,y_2],1-z) =_{\text{def}} \frac{y_1}{y_1}\,(0,1;y_1,1-z)$$

or more generally:

$$(\sharp) \qquad (A,B;[C_1,C_2],D) =_{\text{def}} \frac{AC_2}{AC_1}\,(A,B;C_1,D).$$

Let us check whether $(\sharp)$ is invariant under some projective transformations. Let us consider $\frac{y_2}{y_1}$. Think of it as a cross ratio as in the figure below

$$\frac{y_2-0}{y_1-0}\Big/\frac{(y_2-\frac{y_1+y_2}{2})}{(y_1-\frac{y_1+y_2}{2})} = \frac{y_1}{y_1}\Big/\frac{y_2-y_1}{y_1-y_2} = \frac{-y_2}{y_1}$$

Thus

$$\frac{y_2}{y_1} = -(0,\frac{y_1+y_2}{2},y_1,y_2).$$

This Cross Ratio uses the midpoint between y_1 and y_2. Midpoints E between points A and B can be characterised as the Harmonic conjugate of the point at infinity relative to A and B.

So any transformation of the line leaving the point at infinity fixed will also preserve midpoints, i.e. if A goes to A', B to B' and E to E' and ∞ to ∞, then if E is the midpoint of AB then E' is the midpoint of $A'B'$.

5. Since $r(y,z)$ is commutative it stands to reason to define

$$r^{**} = r([y_1,y_2],[z_1,z_2]) = \text{def}\frac{y_2}{y_1}\cdot\frac{z_2}{z_1}\,r(y_1,z_1).$$

We now have a candidate definition for a Cross-Ratio for intervals.

$$r^{**} = \frac{y_2}{y_1}\frac{z_2}{z_1}\frac{y_1z_1}{(1-y_1)(1-z_1)}$$

Hence

$$(*3) \qquad r^{**} = \frac{y_2z_2}{(1-y_1)(1-z_1)}$$

Let $\bar{y} = [y_1, y_2], \bar{z} = [z_1, z_2]$. Therefore we can define a new $\boxplus$ using a similar connection as $(*2)$:

$$\begin{aligned} \bar{y} \boxplus \bar{z} &= \frac{r^{**}}{1 + r^{**}} \\ &= \frac{y_2 z_2}{(1 - y_1)(1 - z_1) + y_2 z_2} \end{aligned}$$

Hence we summarise:

$$(*4) \qquad \bar{y} \boxplus \bar{z} = \frac{y_2 z_2}{1 - y_1 - z_1 + z_1 y_1 + z_2 y_2}$$

Let us compare $\boxplus$ with $\oplus$
We have

$$\begin{aligned} \bar{y} \oplus \bar{z} &= \left[\frac{y_1 z_2 + y_2 z_1 - y_1 z_1}{1 - y_1 + y_1 z_2 - z_1 + y_2 z_1}, \frac{y_2 z_2}{1 - y_1 + y_1 z_2 - z_1 + y_2 z_1} \right] \\ &= \left[\frac{y_1 z_2 + y_2 z_1 - y_1 z_1}{1 - y_1 - z_1 + y_1 z_2 + y_2 z_1}, \frac{y_2 z_2}{1 - y_1 - z_1 + y_1 z_2 + y_2 z_1} \right] \end{aligned}$$

They are not the same, unless $z_1 = z_2$ or $y_1 = y_2$.
To see this let us ask when do we get a point interval? We equate the numerators of the interval endpoint and we get

$$y_1 z_2 + y_2 z_1 - y_1 z_1 = y_2 z_2$$

hence

$$z_1(y_2 - y_1) = z_2(y_2 - y_1)$$

i.e. either $y_1 = y_2$ or $z_1 = z_2$ i.e. one has to be a point

Summary
We have extended the Cross Ratio to a case of one interval, and it agrees with the Dempster–Shaver $\oplus$. We can also extend the Cross-Ratio to the case with two intervals, giving it the value

$$r^{**}(\bar{y}, \bar{z}) = \frac{y_2 z_2}{(1 - y_1)(1 - z_1)}$$

but it does not agree with the Dempster–Shaver $\bar{y} \oplus \bar{z}$.

We note, however, that since

$$r^{**}(\bar{y}, \bar{z}) = \frac{y_1}{y_1}, \frac{z_1}{z_1} r(y_1, z_1),$$

if we assume $y_1 = 1 - y_2, z_1 = 1 - z_2$ we get

$$r^*(\bar{y}, \bar{z}) = r(y_2, z_2) r(y_1 z_1)$$

We need to check what benefit this gives us!

Example 13 (Using Dempster–Shafer for attack and support). Consider again the basic situations depicted in Figures 30 and 32, or perhaps consider the more fundamental situation of Figure 4. Let us focus on the following Figure 34.

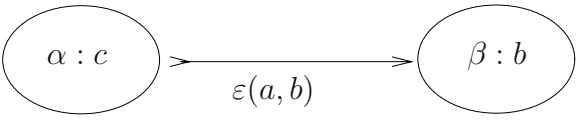


Fig. 34.

The new kind of arrow can stand in for attack, support or any combination transmitted from node c to node b. Our aim in this example is to review our options for the kind of values $\alpha, \beta, \varepsilon$ can take and the options available for the mathematical formulas for their combination and transmission.

Our previous discussion allows for the following Dempster–Shafer option

1. $\varepsilon = 1, \alpha = [z_1, z_2], \beta = y, 0 \leq y \leq 1, 0 \leq z_1 \leq z_2 \leq 1$ and $y' = y \oplus [z_1, z_2]$ and the arrow is interpreted as [Support, Attack] connection as in formula (DS1). We saw the connection with the Cross Ratio as well.
2. To maintain symmetry, we must also allow β to be of the form $[y_1, y_2], 0 \leq y_1 \leq y_2 \leq 1$ and we must write a formula for the [support, attack] on $\beta : b$. The obvious answer is to let

$$\beta' = \alpha \oplus \beta = [z_1, z_2] \oplus [y_1, y_2].$$

3. Another possibility is to take $\boxplus$, i.e. let $\beta'' = \alpha \boxplus \beta$ (as in $(*4)$ of the previous example) but then β'' is a number not a proper interval.
4. Next let us ask what values can we give to ε? Again the simplest and most general value can be $\varepsilon = [u_1, u_2] 0 \leq u_1 \leq u_2 \leq 1$. We need to say how to combine it with α to get a value transmitted? Again in analogy with expert systems in AI we can let the transmitted value to be $\alpha \oplus \varepsilon$. Thus the new value β' would be

$$\beta' = \alpha \oplus \varepsilon \oplus \beta.$$

5 Temporal Dynamics (in Outline)

We devote this section to briefly outline some intuitive motivation for temporal dynamics. We assume that our model has attack arrows only. The reader should be aware that the temporal dynamic aspect of networks is central to the subject and will receive extensive study in our projected series of papers.

Consider the simple network of Figure 35.

In this figure t is a time parameter. So the strength and transmission parameters of the net from a to b depends on time t.

The value of b is $y'(t) = y(t)(1 - \lambda(t)x(t))$.

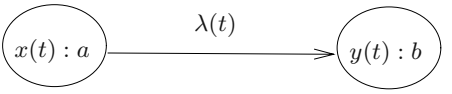


Fig. 35.

Assume that at time $t = 0$ we have $x(0) = 1, \lambda(0) = 1$. In this case $y'(0) = 0$. However if $x(t)$ and $\lambda(t)$ decrease quickly, while $y(t)$ changes slowly, then at time ε we get

$$y'(\varepsilon) \approx y(0)(1 - \varepsilon^2 \cdot \dot{\lambda}(0)\dot{x}(0))$$

where $\dot{x}$ is $\dfrac{dx}{dt}$, i.e. the speed (time derivative) of x and similarly $\dot{\lambda} = \dfrac{d\lambda}{dt}$.

So if we are anxious to keep argument b, we might choose to wait a little (wait ε) for argument a and its transmission to weaken considerably.

Consider that we have

a = sex scandal

b = Governor to resign.

The chances are that public opinion will change quickly.

These time changes should be studied in the context of a time-action model. Suppose we have action **e** with precondition b and postcondition c. We want to take action **e** but if b is successfully attacked, we cannot do so. So we wait a bit. Conversely, suppose that we have d attacks a. Since d attacks a, b is available as $+b$ and so action **e** can be taken. But if d is weakening with time, we may choose to take action **e** immediately, while a is still 'saving' b by attacking a.

So a more sophisticated time–action–argument model will look at the speed of changes and will give values for actions to be taken.

We need to say more about what actions do in the model. We need to define the notion of a fact. We agree that syntactical facts e (as opposed to arguments), can be identified in our model by two properties:

1. $V(e) = 1$
2. e is not attacked by anything.

Of course there may be some arguments that have properties 1 and 2 above, but then for all practical purposes they are like facts.

There may be examples where it looks like some facts can be attacked by other facts. The fact that data is available on the computer may be attacked by the fact that a password was irretrievably lost. However, we can also look at the attack as focussing on the transmission rate of the fact and not the fact itself. We further accept that a node e is considered a semantical fact if $V(e) = 1$ and no attack arrows with positive transmission rate go into e. In a temporal dynamics model, these properties must hold at all times. If they hold only at some of the time, then e is not a fact but a commonly accepted truth which may sometime be attacked or doubted.

What do actions do? Actions create or destroy facts (see Gabbay and Woods [11]). So if at time t an action **e** is fired then the result is that some facts get

deleted from the network and some new facts are added. We can also assume that all values V change as the result of the action.

For simplicity, let us assume that an action adds only one fact or deletes only one fact. Since we can formally delete by attacking we will only allow adding facts. By adding a fact we mean either a new fact or turning an existing argument into a fact. So an action has the form $\mathbf{e}$ = (preconditions, post conditions), where the precondition is a sequence of arguments $((x_i : a_i))$ and the postcondition is a sequence $((a \to y_i \to b_i))$. This means that we add the fact a and let it attack b_i with transmission rate $y_i, i = 1, \ldots, n$. In a given network if a is not a node then we add it as a node with value 1 and let it attack any node b_i which is in the network.

If a is already in the network, then "disconnect" all attacks on a by giving them value 0. Give a the value 1 and let a attack all existing b_i in the network. If b_i is already attacked by a with transmission rate u_i, then let the new transmission rate be $\max(u_i, y_i)$.

Note that $\mathbf{e}$ is stated independently of the network. To be activated we need the net final value of a_i to be at least x_i and then the postconditions act on the available b_i.

Example 14. Consider the network of Figure 5. Consider the action $\mathbf{e}$ with precondition $((x : a), (w : d))$ and postcondition $((b \to u \to c), (b \to y \to g))$. This action can be applied to the network of Figure 5.

The result is Figure 36 below. Note that since there is no g in the network, $b \to y \to g$ is not implemented.

This is equivalent to Figure 37. The next question for us to answer in a temporal network is the following. If action $\mathbf{e}$ is activated at time t, when do we see the result? If the network operates in discrete time, then the result is at time $t+1$. Otherwise we have to treat the action like an *impulse* in a physical system, as when a ball hits another ball and gets it moving, and assume the result of the action $\mathbf{e}$ at t manifests itself immediately at all times s such that $t < s$. We have to give a reasonable definition of how the result of the action manifests itself. A good example for initial consideration is that if a new argument e is created by an action at time t then it shows up at all times $s > t$ and its strength at

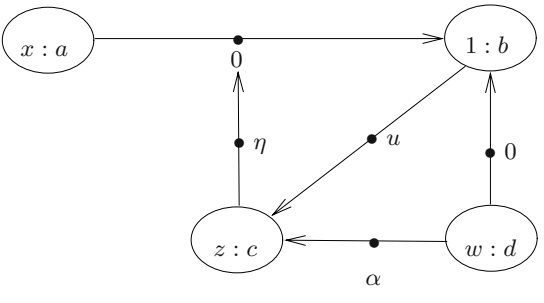


Fig. 36.

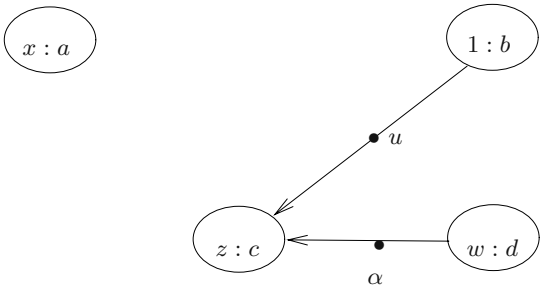


Fig. 37.

time $s > t$ decays slowly as s increases, say it has the form $V_s(e) = \dfrac{k}{1+s-t}$, k a constant ≤ 1. Similarly we can ask for a decay of the transmission rates.

References

1. C. Adler. *Modern Geometry*. Second edition, McGraw Hill, 1967.
2. R. M. Anderson, B. D. Turner and L. R. Taylor, eds. *Population Dynamics*. Blackwell, 1979.
3. H. Barringer, D. Gabbay and J. Woods. Network modalities. In preparation (paper 260).
4. T. Bench-Capon. Persuasion in practical argument using value based argumentation framework. *Journal of Logic and Computation*, **13**, 429–448, 2003.
5. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and N-person games. *Artificial Intelligence*, **77**, 321–357, 1995.
6. T. Faulkner. *Projective Geometry*, Oliver and Boyd, 1949..
7. D. M. Gabbay. Theory of hypermodal logics, *Journal of Philosophical Logic*, **31**, 211–243, 2002.
8. A. S. d'Avila Garcez, D. M. Gabbay and L. C. Lamb. Argumentation Neural Networks. In *Proceedings of 11th International Conference on Neural Information Processing (ICONIP'04)*, Calcutta, India, Lecture Notes in Computer Science LNCS, Springer-Verlag, (to appear) November 2004.
9. D. Gabbay and G. Metcalfe. Cross Ratio uni-norms. In preparation.
10. D. M. Gabbay and J. Woods. Non-cooperation in dialogue logic. *Synthese*, **127**, 161–180, 2001.
11. D. M. Gabbay and J. Woods. *Ad baculum* is not a fallacy. In *Proceedings of the Fourth Internatioal Conference of the International Society for the Study of Arguementatin*, F. H. van Eemeren,R. Grootendorst, J. A. Blair and C.A. Willard, eds. p. 221–224, SicSat, Amsterdam, 1998.
12. D. M. Gabbay and J. Woods. More on non-cooperation in dialogue logic. *Logic Journal of the IGPL*, **9**, 305–324, 2001.
13. D. M. Gabbay and J. Woods. Formal approaches to practical reasoning. In *Handbook of the Logic of Argument and Inference: The Turn Towards the Practical*, D. M. Gabbay, R. H. Johnson, H. J. Ohlbach and J. Woods, eds. pp. 449–481. North-Holland, Amsterdam, 2002.

14. D. M. Gabbay and J. Woods. The law of evidence and labelled deductive systems. *Phi-News*, **4**, 5–47, 2003.
15. P. Hajek, T Havranek, R. Jirousek. *Uncertain Information Processing in Expert Systems*. CRC Press, 1992.
16. P. Hajek and J. Valdes. An analysis if MYCIN-like expert systems. *Mathware and Soft Computing*, **1**, 45–68, 1994.
17. S. A. Levin, ed. *Studies in Mathematical Biology*, Part II, Populations and Communities. Mathematical Association of America, 1978.
18. J. D. Murray. *Mathematical Biology*, Volume 1, Springer-Verlag, 2001.
19. G. Shafer. *Mathematical Theory of Evidence*. Princeton University Press, 1976.
20. P. Turchin. *Complex Population Dynamics*. Princeton University Press, 2003.
21. J. Woods. *The Death of Argument: Fallacies in Agent-Based Reasoning*. Kluwer, Dordrecht and Boston, to appear in 2004.

Footprints of Conditionals*

Christoph Beierle and Gabriele Kern-Isberner

Praktische Informatik VIII – Wissensbasierte Systeme,
Fachbereich Informatik, FernUniversität Hagen,
D-58084 Hagen, Germany
{christoph.beierle,gabriele.kern-isberner}@fernuni-hagen.de

Abstract. Probabilistic conditionals are a powerful means of representing commonsense and expert knowledge. By viewing probabilistic conditionals as an institution, we obtain a formalization of probabilistic conditionals as a logical system. Using the framework of institutions, we phrase a general representation problem that is closely related to the selection of preferred models. The problem of discovering probabilistic conditionals from data can be seen as an instance of the inverse representation problem, thereby considering knowledge discovery as an operation inverse to inductive knowledge representation. These concepts are illustrated using the well-known probabilistic principle of maximum entropy for which we sketch an approach to solve the inverse representation problem.

1 Introduction

Commonsense and expert knowledge is most generally expressed by rules, connecting a precondition and a conclusion by an *if-then*-construction. If-then-rules are more formally denoted as *conditionals*, and often they occur in the form of *probabilistic conditionals*. For instance, such conditionals may describe the knowledge available to a physician when he has to make a diagnosis. Or they may express commonsense knowledge like "*Students are young with a probability of (about) 80 %*" and "*Singles (i.e. unmarried people) are young with a probability of (about) 70 %*", the latter knowledge being formally expressed by $\{(young|student)[0.8], (young|single)[0.7]\}$. The crucial point with conditionals is that they carry generic knowledge which can be applied to different situations. This makes them most interesting objects in Artificial Intelligence, in theoretical as well as in practical respect (see e.g. [CFH95]).

In this paper, we address several problems and questions which usually arise when dealing with (probabilistic) conditionals. First, how can we formalize a general logic of probabilistic conditionals? Probabilistic conditionals are non-classical in various respects, so how does this logic fit the frameworks of other logics like e.g. classical logic? We show that the concept of *institutions* deceived by Goguen and Burstall [GB92] as a general framework for logical systems may also be applied to probabilistic conditionals. By answering this question in this

* The research reported here was partially supported by the DFG – Deutsche Forschungsgemeinschaft within the CONDOR-project under grant BE 1700/5-1.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 99–119, 2005.

way, we might remove some of the suspicions that probabilistic modelling and reasoning is sometimes looked upon from a formal logical point of view.

Second, how to use probabilistic rule bases for modelling and inference? How to combine knowledge expressed by conditionals so as to yield expressive answers to queries? We propose a model-based solution to this problem here by formalizing a general *representation problem* within the framework of institutions: Given a specification of a theory, select a set of models as its desired representation. The well-known probabilistic principle of maximum entropy (ME-principle) is easily seen to solve this representation problem for probabilistic conditionals in a most satisfying way [Par94].

Now that a formal framework for probabilistic conditionals and their (inductive) representation has been outlined, we raise the third question: Where do the probabilistic conditionals apt to represent knowledge appropriately come from? Actually, this question should be considered in the first place, but usually, one tends to assume that some omniscient expert is able express his knowledge as (probabilistic) rules. In practice, however, statistical data are often used to (at least) support the building of knowledge bases. Statistical data may be summarized as a frequency distribution which constitutes a probabilistic model for the rules it represents. From our point of view, the important question how to extract rules from statistical data may thus be viewed as inverting the above mentioned representation problem, in that now a set of probabilistic conditionals (i.e. a probabilistic specification) has to be selected, given a model. So, the task of discovering rules from data can be considered as an instance of this abstract *inverse representation problem*. For the inductive representation of probabilistic conditionals via the ME-principle, we will sketch an approach to solve this inverse representation problem, i.e. to compute a concise set of rules which are most relevant with respect to the ME-method. Since entropy measures the amount of indeterminateness which is dual to information, this approach can be considered as aiming to find the most informative rules in data.

The formal framework we present in this paper to deal with practical problems in probabilistic reasoning is part of the even more general framework on which the CONDOR project is based. CONDOR (Conditionals – discovery and revison) is aimed to provide a complete theory and a computational tool for inductively representing and revising conditional knowledge, on the one hand, and to discover rules in data which are relevant with respect to some underlying inductive representation method, on the other hand.

The rest of this paper is organized as follows: In Section 2, we further motivate our approach and review some of the related work in the three problem areas this paper is concerned with. Based on the institution framework and our formalization of probabilistic conditional logic as an institution carried out in Section 3, we introduce the general notions of both the representation problem and the inverse representation problem in Section 4 and illustrate them in various application areas. Section 5 gives an overview of an algorithm coping with the inverse representation problem in the ME-framework. Section 6 gives some conclusions and points out further work.

2 Motivation and Related Work

2.1 Towards a Formal Logic for Probabilistic Conditionals

As a general framework for logical systems, Goguen and Burstall introduced the notion of an institution [GB92]. An institution formalizes the informal notion of a logical system, including syntax, semantics, and the relation of satisfaction between them. The latter poses the major requirement for an institution: that the satisfaction relation is consistent under the change of notation.

Institutions have been used for the general study of logics. For instance, there are widely applicable results about building up larger theories from smaller components. Institution morphisms support the comparison of different logics, are used for glueing together several logics within one system, and may permit a theorem prover for one institution to be used on theories from another one. Additionally, institutions have also been used as a basis for specification and development languages; in fact, institutions arose in the context of designing the specification language Clear [BG80,GB92]. For some of the work using institutions see e.g. [GT00,ST97,BV87,Tar96,GR02].

Whereas in [GB92] the examples for an institution are mostly based on classical logics, we will show here that also the logic of probabilistic conditionals fits nicely into the institution framework. Viewing probabilistic conditionals as an institution supports the study of structural properties of both syntax and semantics of this logic, and it will allow us to immediately apply various general results for institutions to our probabilistic logic, giving us for free e.g. the presentation lemma, various closure properties, or the notions and results about theories and their morphisms.

2.2 Modelling Based on the Principle of Maximum Entropy

Usually, probabilistic rule bases represent incomplete knowledge, in that there are a lot of probability distributions apt to represent them. So learning, or inductively representing, respectively, the rules means to take them as a set of conditional constraints and to select a unique probability distribution as a "best" model which can be used for queries and further inferences. Paris [Par94] investigates several inductive representation techniques and proves that the *principle of maximum entropy (ME-principle)* yields the only method to represent incomplete knowledge in an unbiased way, satisfying a set of postulates describing sound commonsense reasoning.

Therefore, the ME-principle provides a most convenient and founded method to represent incomplete probabilistic knowledge. Unlike Bayesian networks [CDLS99], no external (and often unjustified) independence assumptions have to be made, and only relevant conditional dependencies are part of the knowledge base. Bayesian networks need a lot of probabilities being specified. If one has to model the dependencies, for instance, between two diseases, D_1, D_2, and two symptoms, S_1, S_2, one has to quantify all probabilities $P(s_j|d_i)$, where s_j and d_i, respectively, is any one of $S_j, \neg S_j$ and $D_i, \neg D_i$, for $i, j = 1, 2$. But not only

the large amount of probabilities necessary to build up Bayesian networks are a problem. Although a physician will usually be capable to quantify $P(S_j|D_i)$ from his expert knowledge, he will hardly be able to say something informed about $P(S_j|\neg D_i)$ – what is the probability of a symptom given that the disease is *not* present? In an ME-environment, the expert has only to list whatever relevant conditional probabilities he is aware of. Moreover, the two basic ingredients for Bayesian networks, namely the set of conditional probabilities and the independence assumptions, specify *complete probabilistic knowledge*, thereby detracting from the flexible power of generic conditional information. ME-modelling, on the other hand, preserves the generic nature of conditionals by minimizing the amount of information being added.

Nevertheless, modelling ME-rule bases has to be done carefully so as to ensure that *all* relevant dependencies are taken into account. This task can be difficult and troublesome. Usually, the modelling rules are based somehow on statistical data. To date, however, no tool is known that helps to find ME-optimal set of rules. In [KI00], a general approach to solve the inverse representation problem is presented which we will describe here for the ME-framework.

2.3 Searching for Structures of Knowledge

The most usual approach to discover "interesting" rules from data is to look for rules with a significantly high (conditional) probability and a concise antecedent [KIR96,AMS+96,MS98]. Basing relevance on frequencies, however, is sometimes unsatisfactory and inadequate, in particular in complex domains like medicine. Furthermore, the rules are considered as isolated pieces of knowledge, no interaction between rules can be taken into account.

In order to obtain more structured information, one often searches for causal relationships by investigating conditional independencies and thus non-interactivity between sets of variables [CH92,SGS93,Hec96,Bun96]. Some of these algorithms also make use of optimization criteria which are based on entropy [HC90,Gei92]. Although causality is undoubtedly most important for human understanding, it seems to be too rigid a concept to represent human knowledge in an exhaustive way. For instance, a person suffering from a flu is certainly sick ($P(sick|flu) = 1$), and he often will complain about headache ($P(headache|flu) = 0.9$). Then we have $P(headache|flu) = P(headache|flu \wedge sick)$, but we would surely expect $P(headache|\neg flu) \neq P(headache|\neg flu \wedge sick)$! Although the first equality suggests a conditional independence between *sick* and *headache*, due to the causal dependency between *headache* and *flu*, the second inequality shows this to be (of course) false. Furthermore, a physician might also state some conditional probability involving sick and headache, so that we obtain a complex network of rules. Each of these rules will be considered relevant by the expert, but none will be found when searching for conditional independencies! So, what actually are the "structures of knowledge" by which conditional dependencies (not independencies!) manifest themselves in data? What are the "footprints" conditionals leave on probabilities when being learned inductively?

3 Viewing Probabilistic Conditional Logic as an Institution

After recalling the definition of an institution and fixing some basic notation, we first present propositional logic in the institution framework. We then formalize the logic of probabilistic conditionals as an institution, give some examples, and demonstrate how well-known concepts like marginal distributions occur within the institution context.

3.1 Preliminaries: Basic Definitions and Notations

If C is a category, $|C|$ denotes the objects of C and $/C/$ its morphisms; for both objects $c \in |C|$ and morphisms $\varphi \in /C/$, we also write just $c \in C$ and $\varphi \in C$, respectively. C^{op} is the opposite category of C, with the direction of all morphisms reversed. $\mathcal{SET}$ and $\mathcal{CAT}$ denote the categories of sets and of categories, respectively. (For more information about categories, see e.g. [HS73] or [Mac72].) The central definition of an institution [GB92] is the following (cf. Figure 1 that visualizes the relationships within an institution):

Definition 1. *An institution is a quadruple* $Inst = \langle Sig, Mod, Sen, \models \rangle$ *with a category* Sig *of signatures as objects, a functor* $Mod : Sig \to \mathcal{CAT}^{op}$ *yielding the category of* Σ*-models for each signature* Σ*, a functor* $Sen : Sig \to \mathcal{SET}$ *yielding the sentences over a signature, and a* $|Sig|$*-indexed relation* $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$ *such that for each signature morphism* $\varphi : \Sigma \to \Sigma' \in /Sig/$*, for each* $m' \in |Mod(\Sigma')|$*, and for each* $f \in Sen(\Sigma)$ *the following* satisfaction condition *holds:*

$$m' \models_{\Sigma'} Sen(\varphi)(f) \quad \textit{iff} \quad Mod(\varphi)(m') \models_\Sigma f$$

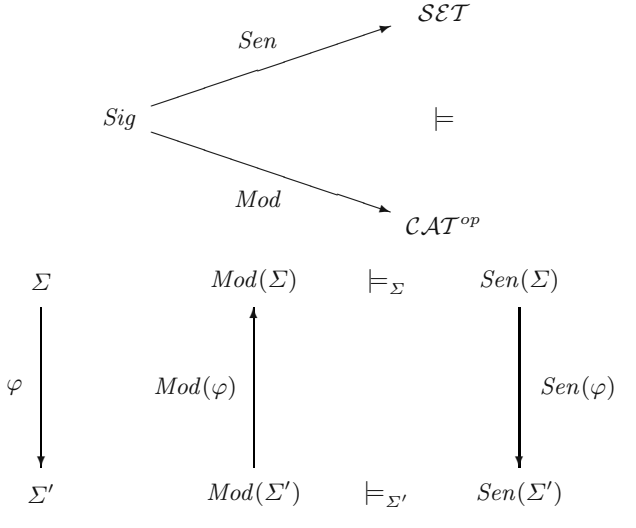


Fig. 1. Relationships within an institution $Inst = \langle Sig, Mod, Sen, \models \rangle$ [GB92].

For sets F, G of Σ-sentences and a Σ-model m we write $m \models_\Sigma F$ iff $m \models_\Sigma f$ for all $f \in F$. The satisfaction relation is lifted to semantical entailment $\models_\Sigma$ between sentences by defining $F \models_\Sigma G$ iff for all Σ-models m with $m \models_\Sigma F$ we have $m \models_\Sigma G$. $F^\bullet = \{f \in Sen(\Sigma) \mid F \models_\Sigma f\}$ is called the *closure* of F, and F is *closed* if $F = F^\bullet$. The closure operator fulfils the *closure lemma* $\varphi(F^\bullet) \subseteq \varphi(F)^\bullet$ and various other nice properties like $\varphi(F^\bullet)^\bullet = \varphi(F)^\bullet$ or $(F^\bullet \cup G)^\bullet = (F \cup G)^\bullet$. A consequence of the closure lemma is that entailment is preserved under change of notation carried out by a signature morphism, i.e. $F \models_\Sigma G$ implies $\varphi(F) \models_{\varphi(\Sigma)} \varphi(G)$ (but not vice versa).

3.2 The Institution of Propositional Logic

In all circumstances, propositional logic seems to be the most basic logic. The components of its institution $Inst_\mathcal{B} = \langle Sig_\mathcal{B}, Mod_\mathcal{B}, Sen_\mathcal{B}, \models_\mathcal{B} \rangle$ will be defined in the following.

Signatures: $Sig_\mathcal{B}$ is the category of propositional signatures. A propositional signature $\Sigma \in |Sig_\mathcal{B}|$ is a set of propositional variables, $\Sigma = \{a_1, a_2, \ldots\}$. A propositional signature morphism $\varphi : \Sigma \to \Sigma' \in /Sig_\mathcal{B}/$ is a function mapping propositional variables to propositional variables.

Models: For each signature $\Sigma \in Sig_\mathcal{B}$, $Mod_\mathcal{B}(\Sigma)$ contains the set of all propositional interpretations for Σ, i.e.

$$|Mod_\mathcal{B}(\Sigma)| = \{I \mid I : \Sigma \to Bool\}$$

where $Bool = \{true, false\}$. Due to its simple structure, the only morphisms in $Mod_\mathcal{B}(\Sigma)$ are the identity morphisms. For each signature morphism $\varphi : \Sigma \to \Sigma' \in Sig_\mathcal{B}$, we define the functor $Mod_\mathcal{B}(\varphi) : Mod_\mathcal{B}(\Sigma') \to Mod_\mathcal{B}(\Sigma)$ by $(Mod_\mathcal{B}(\varphi)(I'))(a_i) := I'(\varphi(a_i))$ where $I' \in Mod_\mathcal{B}(\Sigma')$ and $a_i \in \Sigma$.

Sentences: For each signature $\Sigma \in Sig_\mathcal{B}$, the set $Sen_\mathcal{B}(\Sigma)$ contains the usual propositional formulas constructed from the propositional variables in Σ and the logical connectives $\wedge$ (and), $\vee$ (or), and $\neg$ (not). Additionally, the classical (material) implication $A \Rightarrow B$ is used as a syntactic variant for $\neg A \vee B$. The symbols $\top$ and $\bot$ denote a tautology (like $a \vee \neg a$) and a contradiction (like $a \wedge \neg a$), respectively.

For each signature morphism $\varphi : \Sigma \to \Sigma' \in Sig_\mathcal{B}$, the function $Sen_\mathcal{B}(\varphi) : Sen_\mathcal{B}(\Sigma) \to Sen_\mathcal{B}(\Sigma')$ is defined by straightforward inductive extension on the structure of the formulas; e.g. $Sen_\mathcal{B}(\varphi)(a_i) = \varphi(a_i)$ and $Sen_\mathcal{B}(\varphi)(A \wedge B) = Sen_\mathcal{B}(\varphi)(A) \wedge Sen_\mathcal{B}(\varphi)(B)$. In the following, we will abbreviate $Sen_\mathcal{B}(\varphi)(A)$ by just writing $\varphi(A)$.

In order to simplify notations, we will often replace conjunction by juxtaposition and indicate negation of a formula by barring it, i.e. $AB = A \wedge B$ and $\overline{A} = \neg A$. As usual, an *atom* is a formula consisting of just a propositional variable, a *literal* is a positive or a negated atom, an *elementary conjunction* is a conjunction of literals, and a *complete conjunction* is an elementary conjunction containing each atom either in positive or in negated form. Ω_Σ denotes the set

of all complete conjunctions over a signature Σ; if Σ is clear from the context, we may drop the index Σ.

Note that there is an obvious bijection between $|\mathit{Mod}_{\mathcal{B}}(\Sigma)|$ and Ω_Σ, associating with $I \in |\mathit{Mod}_{\mathcal{B}}(\Sigma)|$ the complete conjunction $\omega_I \in \Omega_\Sigma$ in which an atom $a_i \in \Sigma$ occurs in positive form iff $I(a_i) = \mathit{true}$.

Satisfaction Relation: For any $\Sigma \in |\mathit{Sig}_{\mathcal{B}}|$, the satisfaction relation

$$\models_{\mathcal{B},\Sigma} \quad \subseteq \quad |\mathit{Mod}_{\mathcal{B}}(\Sigma)| \times \mathit{Sen}_{\mathcal{B}}(\Sigma)$$

is defined as expected for propositional logic, e.g. $I \models_{\mathcal{B},\Sigma} a_i$ iff $I(a_i) = \mathit{true}$ and $I \models_{\mathcal{B},\Sigma} A \wedge B$ iff $I \models_{\mathcal{B},\Sigma} A$ and $I \models_{\mathcal{B},\Sigma} B$ for $a_i \in \Sigma$ and $A, B \in \mathit{Sen}_{\mathcal{B}}(\Sigma)$.

Proposition 1. *$\mathit{Inst}_{\mathcal{B}} = \langle \mathit{Sig}_{\mathcal{B}}, \mathit{Mod}_{\mathcal{B}}, \mathit{Sen}_{\mathcal{B}}, \models_{\mathcal{B}} \rangle$ is an institution.*

It is easy to prove this proposition since the satisfaction condition

$$I' \models_{\mathcal{B},\Sigma'} \varphi(A) \quad \text{iff} \quad \mathit{Mod}_{\mathcal{B}}(\varphi)(I') \models_{\mathcal{B},\Sigma} A$$

holds by straightforward induction on the structure of A. E.g., for a propositional variable a_i, we have $I' \models_{\mathcal{B},\Sigma'} \varphi(a_i)$ iff $I'(\varphi(a_i)) = \mathit{true}$ iff $(\mathit{Mod}_{\mathcal{B}}(\varphi)(I'))(a_i) = \mathit{true}$ iff $\mathit{Mod}_{\mathcal{B}}(\varphi)(I') \models_{\mathcal{B},\Sigma} a_i$.

Example 1. Let $\Sigma = \{s, t, u\}$ and $\Sigma' = \{a, b, c\}$ be two propositional signatures with the atomic propositions s – *being a scholar*, t – *being not married*, u – *being single* and a – *being a student*, b – *being young*, c – *being unmarried*. Let I' be the Σ'-model with $I'(a) = \mathit{true}$, $I'(b) = \mathit{true}$, $I'(c) = \mathit{false}$. Let $\varphi : \Sigma \to \Sigma' \in \mathit{Sig}_{\mathcal{B}}$ be the signature morphism with $\varphi(s) = a$, $\varphi(t) = c$, $\varphi(u) = c$. The functor $\mathit{Mod}_{\mathcal{B}}(\varphi)$ takes I' to the Σ-model $I := \mathit{Mod}_{\mathcal{B}}(\varphi)(I')$, yielding $I(s) = I'(a) = \mathit{true}$, $I(t) = I'(c) = \mathit{false}$, $I(u) = I'(c) = \mathit{false}$.

Note that in the example, φ is neither surjective nor injective. φ not being surjective makes the functor $\mathit{Mod}_{\mathcal{B}}(\varphi)$ a forgetful functor – any information about b (being young) in I' is forgotten in I. φ not being injective implies that any two propositional variables from Σ mapped to the same element in Σ' are both being identified with the same proposition; thus, under the forgetful functor $\mathit{Mod}_{\mathcal{B}}(\varphi)$, the interpretation of t (*being not married*) and u (*being single*) will always be equivalent since $\varphi(t) = \varphi(u)$.

3.3 The Institution of Probabilistic Conditional Logic

Based on $\mathit{Inst}_{\mathcal{B}}$, we can now define the institution of probabilistic conditional logic $\mathit{Inst}_{\mathcal{C}} = \langle \mathit{Sig}_{\mathcal{C}}, \mathit{Mod}_{\mathcal{C}}, \mathit{Sen}_{\mathcal{C}}, \models_{\mathcal{C}} \rangle$. We will first give a very short introduction to probabilistics as far as it is needed here.

Let $\Sigma \in |\mathit{Sig}_{\mathcal{B}}|$ be a propositional signature. A *probability distribution* (or *probability function*) *over* Σ is a function $P : \mathit{Sen}_{\mathcal{B}}(\Sigma) \to [0, 1]$ such that $P(\top) = 1$, $P(\bot) = 0$, and $P(A \vee B) = P(A) + P(B)$ for any formulas $A, B \in \mathit{Sen}_{\mathcal{B}}(\Sigma)$ with $AB = \bot$. Each probability distribution P is determined uniquely by its

values on the complete conjunctions $\omega \in \Omega_\Sigma$, since $P(A) = \sum_{\omega \in \Omega_\Sigma, \omega \models_{\mathcal{B},\Sigma} A} P(\omega)$. For two propositional formulas $A, B \in Sen_\mathcal{B}(\Sigma)$ with $P(A) > 0$, the *conditional probability of B given A* is $P(B|A) := \frac{P(AB)}{P(A)}$. Any subset $\Sigma_1 \subseteq \Sigma$ gives rise to a distribution $P_{\Sigma_1} : Sen_\mathcal{B}(\Sigma_1) \to [0,1]$ by virtue of defining $P_{\Sigma_1}(\omega_1) = \sum_{\omega \in \Omega_\Sigma, \omega \models_{\mathcal{B},\Sigma} \omega_1} P(\omega)$ for all $\omega_1 \in \Omega_{\Sigma_1}$; P_{Σ_1} is called the *marginal distribution of P on Σ_1*.

Signatures: $Sig_\mathcal{C}$ is identical to the category of propositional signatures, i.e. $Sig_\mathcal{C} = Sig_\mathcal{B}$.

Models: For each signature Σ, the objects of $Mod_\mathcal{C}(\Sigma)$ are probability distributions over the propositional variables, i.e.

$$|\, Mod_\mathcal{C}(\Sigma)\,| \;=\; \{P \mid P \text{ is a probability distribution over } \Sigma\}$$

As for $Mod_\mathcal{B}(\Sigma)$, we assume in this paper that the only morphisms in $Mod_\mathcal{C}(\Sigma)$ are the identity morphisms.

For each signature morphism $\varphi : \Sigma \to \Sigma'$, we define a functor $Mod_\mathcal{C}(\varphi) : Mod_\mathcal{C}(\Sigma') \to Mod_\mathcal{C}(\Sigma)$ by mapping each distribution P' over Σ' to a distribution $Mod_\mathcal{C}(\varphi)(P')$ over Σ. $Mod_\mathcal{C}(\varphi)(P')$ is defined by giving its value for all complete conjunctions over Σ:

$$(Mod_\mathcal{C}(\varphi)(P'))(\omega) \;:=\; P'(\varphi(\omega)) = \sum_{\omega' : \omega' \models_{\mathcal{B},\Sigma'} \varphi(\omega)} P'(\omega')$$

where ω and ω' are complete conjunctions over Σ and Σ', respectively. We still have to show:

Proposition 2. *$Mod_\mathcal{C}(\varphi)(P')$ is a Σ-model.*

It is straightforward to see that the equation used to define $Mod_\mathcal{C}(\varphi)(P')$ easily generalizes to propositional formulas $A \in Sen_\mathcal{B}(\Sigma)$, that is to say it holds that $(Mod_\mathcal{C}(\varphi)(P'))(A) = P'(\varphi(A))$.

Another immediate consequence is the following:

Proposition 3. *Let $\varphi : \Sigma \to \Sigma'$ and $P := Mod_\mathcal{C}(\varphi)(P')$ as above. If φ is injective, then the marginal distribution $P'_{\varphi(\Sigma)}$ of P' over $\varphi(\Sigma) := \{\varphi(x) \mid x \in \Sigma\} \subseteq \Sigma'$ is identical to P with variables $\varphi(a_i)$ in P' renamed to a_i in P, i.e. $P'_{\varphi(\Sigma)}(\omega') = P(\varphi^{-1}(\omega'))$ for all complete conjunctions ω' over $\varphi(\Sigma)$.*

This proposition shows that the well-known (semantical) concept of a marginal distribution coincides with the forgetful functor induced by an injective signature morphism, forgetting the propositional variables not reached by a non-surjective φ. If φ is non-injective, two propositional variables mapped to the same element in Σ' are identified. Therefore, any conjunction containing a negated and a non-negated propositional variable both being identified under φ gets the probability 0. Thus, in the general case for non-surjective and non-injective φ, $Mod_\mathcal{C}(\varphi)(P')$ is defined by 'collapsing' all propositional variables identified under φ and taking the marginal distribution over the remaining variables reached by φ.

Example 2. Let Σ, Σ' and φ be as in Example 1. We define a Σ'-model P' by assigning a probability $P'(\omega')$ to every complete conjunction over Σ':

ω'	$P'(\omega')$	ω'	$P'(\omega')$	ω'	$P'(\omega')$	ω'	$P'(\omega')$
abc	0.1950	$ab\overline{c}$	0.1758	$a\overline{b}c$	0.0408	$a\overline{b}\overline{c}$	0.0519
$\overline{a}bc$	0.1528	$\overline{a}b\overline{c}$	0.1378	$\overline{a}\overline{b}c$	0.1081	$\overline{a}\overline{b}\overline{c}$	0.1378

Thus, for instance, the probability $P'(abc)$ of *being a student*, *being young*, and *being unmarried* is 0.1950, and the probability $P'(a\overline{b}c)$ of *being a student*, *not being young*, and *being unmarried* is 0.0408.

The functor $Mod_{\mathcal{C}}(\varphi)$ transforms P' into the following Σ-model P:

ω	$P(\omega)$	ω	$P(\omega)$	ω	$P(\omega)$	ω	$P(\omega)$
stu	0.2358	$st\overline{u}$	0.0000	$s\overline{t}u$	0.0000	$s\overline{t}\overline{u}$	0.2277
$\overline{s}tu$	0.2609	$\overline{s}t\overline{u}$	0.0000	$\overline{s}\overline{t}u$	0.0000	$\overline{s}\overline{t}\overline{u}$	0.2756

For instance, the probability $P(stu)$ of *being a scholar*, *being not married*, and *being single* is 0.2358.

Sentences: For each signature Σ, the set $Sen_{\mathcal{C}}(\Sigma)$ contains *probabilistic conditionals* (sometimes also called *probabilistic rules*) of the form

$$(B|A)[x]$$

where $A, B \in Sen_{\mathcal{B}}(\Sigma)$ are propositional formulas from $Inst_{\mathcal{B}}$. $x \in [0,1]$ is a probability value indicating the degree of certainty for the occurrence of B under the condition A.

Note that a probabilistic fact of the form $B[x]$ can easily be expressed as a conditional $(B|\top)[x]$ with a tautology as trivial antecedent.

For each signature morphism $\varphi : \Sigma \to \Sigma'$, the extension $Sen_{\mathcal{C}}(\varphi) : Sen_{\mathcal{C}}(\Sigma) \to Sen_{\mathcal{C}}(\Sigma')$ is defined by straightforward inductive extension on the structure of the formulas: $Sen_{\mathcal{C}}(\varphi)((B|A)[x]) = (\varphi(B)|\varphi(A))[x]$.

Satisfaction Relation: The satisfaction relation $\models_{\mathcal{C},\Sigma} \subseteq |\,Mod_{\mathcal{C}}(\Sigma)\,| \times Sen_{\mathcal{C}}(\Sigma)$ is defined, for any $\Sigma \in |\,Sig_{\mathcal{C}}\,|$, by

$$P \models_{\mathcal{C},\Sigma} (B|A)[x] \quad \text{iff} \quad P(A) > 0 \text{ and } P(B \mid A) = \frac{P(AB)}{P(A)} = x$$

Note that for probabilistic facts we have $P \models_{\mathcal{C},\Sigma} (B|\top)[x]$ iff $P(B) = x$ from the definition of the satisfaction relation since $P(\top) = 1$.

Proposition 4. *$Inst_{\mathcal{C}} = \langle\, Sig_{\mathcal{C}},\, Mod_{\mathcal{C}},\, Sen_{\mathcal{C}},\, \models_{\mathcal{C}} \,\rangle$ is an institution.*

Example 3. Let Σ, Σ', P, P' and φ be as in Example 2. Then $P' \models_{\mathcal{C},\Sigma'} (b|\top)[0.6614]$ since the probability of *being young* is $P'(b) = 0.6614$, and $P' \models_{\mathcal{C},\Sigma'} (b|a)[0.8]$ since the probability of *being young* under the condition of *being a student* is $P'(b \mid a) = 0.8$.

Similarly, $P \models_{\mathcal{C},\Sigma} (u|\top)[0.4967]$ since under P, the probability of *being single* is $P(u) = 0.4967$. This immediately implies $P' \models_{\mathcal{C},\Sigma'} (c|\top)[0.4967]$ (i.e. under P', the probability of *being unmarried* is 0.4967) due to the satisfaction condition since $\varphi(u) = c$.

In [BKI02], we extend our institutional view of probabilistic conditional logic. In that paper, the institution $Inst_{\mathcal{P}}$ of probabilistic propositional logic, having probabilistic facts of the form $A[x]$ as sentences, is defined additionally and the relationships between the three institutions $Inst_{\mathcal{B}}$, $Inst_{\mathcal{P}}$, and $Inst_{\mathcal{C}}$ are investigated in detail. This is done by using institution morphisms [GB92,GR02] and institution embeddings, telling us, e.g., precisely the possibilities of interpreting probabilistic conditionals as probabilistic or propositional formulas and vice versa.

4 Conditional Theories and the Inverse Representation Problem

The general institution framework provides a rich set of results about theories as well as about their presentations and interrelationships. After reviewing briefly the basic concepts and results, we develop our notions of the representation problem and the inverse representation problem based on theory presentations and illustrate them in various application areas.

4.1 Theories and Their Presentations

The results in this subsection are taken from [GB92] and hold for any institution, thus in particular also for the institution of probabilistic conditional logic:

A *Σ-presentation* is a pair $\langle \Sigma, F\rangle$ with $F \subseteq Sen(\Sigma)$, and a *Σ-theory* is a presentation $\langle \Sigma, F\rangle$ such that F is closed, i.e. $F = F^{\bullet}$ (cf. Section 3.1). Given two theories $T = \langle \Sigma, F\rangle$ and $T' = \langle \Sigma', F'\rangle$, a *theory morphism* φ from T to T', written $\varphi : T \to T'$, is a signature morphism $\varphi : \Sigma \to \Sigma'$ such that $\varphi(F) \subseteq F'$. Σ-theories and their morphisms form a category. $Mod(\langle \Sigma, F\rangle)$ denotes the full subcategory of $Mod(\Sigma)$ of all Σ-models that satisfy F. For a theory morphism $\varphi : T \to T'$, the restriction and corestriction of the functor $Mod(\varphi)$ yields a functor $Mod(\varphi) : Mod(T') \to Mod(T)$.

The *presentation lemma* says that in order to check that a signature morphism is a theory morphism, it suffices to check only the sentences of a presentation for entailment: For presentations $\langle \Sigma, F\rangle$ and $\langle \Sigma', F'\rangle$, $\varphi : \langle \Sigma, F^{\bullet}\rangle \to \langle \Sigma', F'^{\bullet}\rangle$ is a theory morphism iff $\varphi(F) \subseteq F'^{\bullet}$.

From other general properties and theorems for institutions given in [GB92], we can also conclude e.g. that the category of probabilistic conditional logical theories is cocomplete since $Sig_{\mathcal{C}}$ is cocomplete, allowing us putting theories together from subtheories.

4.2 The Representation Problem and Preferred Models

The original motivation for institutions was the definition of the semantics of the specification language Clear [BG80], and institutions have been used for

various aproaches to modularized specification and programming development, often involving the notion of an abtract data type (ADT). When trying to use formal methods in software development, one quickly comes accross the need for a rigorous method for specifying, refining, and implementing data types at levels that are independent from a specific representation used in e.g. traditional programming languages. Using institutions, a *specification* is a (theory) presentation $\langle \Sigma, F \rangle$. This certainly meets the requirement of abstractness; but what does $\langle \Sigma, F \rangle$ represent? The general institution framework provides $Mod(\langle \Sigma, F \rangle)$ as a semantics for $\langle \Sigma, F \rangle$, but in many cases we are interested only in specific models. This is what we call the *representation problem*:

> Given a specification $\langle \Sigma, F \rangle$, select a class of (preferred) models $M \subseteq Mod(\langle \Sigma, F \rangle)$ as its desired representation.

In every specification approach based on logic, an answer to the representation problem must be given. In the initial approach to ADT specifications one is interested in models that are initial in the category $Mod(\langle \Sigma, F \rangle)$; other approaches take the terminal models, the finitely generated models, or even all models as in so-called loose ADT specifications (cf. e.g. [BG80, EM85, ST97, BV91, Wir90, GB92]).]

When it comes to reasoning, the answer given to the representation problem must be taken into account. Whereas classical logical reasoning is done with respect to *all* models, reasoning with respect to the models selected according to the representation problem requires tailored inference techniques. For instance, when reasoning with respect to equationally defined initial ADTs, induction should be used since the initial models are finitely generated.

While for ADT specifications the selection of models $M \subseteq Mod(\langle \Sigma, F \rangle)$ requires special reasoning techniques, the selection itself is not motivated by these reasoning techniques. On the other hand, this is indeed the case for approaches in defeasible reasoning. Here, the motivation to focus on preferred models is to select models which are most appropriate for yielding plausible conclusions. The set of formulas F in a specification $\langle \Sigma, F \rangle$ is taken to specify incomplete knowledge, and basing entailment upon a relatively small set of models (the most plausible ones) means to extend the knowledge expressed by F, so as to derive more (tentative) conclusions than can be obtained by classical deduction. If there is only one most plausible model, then this model completes the available knowledge, and hence inductively represents F. Thereby, in order to get stronger ('better') inference capabilities, one deliberately accepts leaving the framework of classical logical reasoning and choosing e.g. the *preferential models* approach of nonmonotonic logics (cf. [Sho87,Mak89,KLM90]) as appropriate paradigm.

In a probabilistic environment, the problem of plausible inference is even more difficult to be dealt with, since $Mod_{\mathcal{C}}(\langle \Sigma, R \rangle)$ typically contains a huge number of very different distributions, all reflecting the conditional knowledge given by R. Entailment based on all models is quite weak, e.g. it is not possible to derive the probability of each conjunct from the probability of a conjunction. In general, mostly intervals of possible probability values can be obtained which are often inexpressively large.

Example 4. Let R be a set containing two probabilistic conditionals both having C as its conclusion, one under the condition A and the other one under the condition B, i.e. $R = \{(C|A)[x], (C|B)[y]\}$ for some given probabilities $x, y \in (0, 1)$. What does this mean for the occurrence of C under the condition of both A and B? We can show that for *any* probability $z \in (0, 1)$ the set R is compatible with $(C|AB)[z]$, that is, for all (non-trivial) $x, y, z \in (0, 1)$, there is a probability distribution P such that both $P \models_{\mathcal{C}} R$ and $P \models_{\mathcal{C}} (C|AB)[z]$. So, actually nothing can be derived about the probability of $(C|AB)$ from R.

The problem of yielding plausible inferences from a set R of probabilistic conditionals can be solved by the *principle of maximum entropy (ME-principle)* in the following way: This information-theoretical principle selects a distribution P^* from $Mod_{\mathcal{C}}(\langle \Sigma, R \rangle)$ whose entropy

$$H(P^*) = -\sum_{\omega \in \Omega} P^*(\omega) \log_2 P^*(\omega)$$

is maximal. (For some strong arguments supporting this approach see e.g. [SJ80, PV90, Par94, RKI97, KI01b]). Similar as for initial ADT specifications (initial objects are unique up to isomorphisms), this selects a *unique* model (cf. [Csi75]), denoted by $P^* = ME(\langle \Sigma, R \rangle)$, as the desired representation for the specification $\langle \Sigma, R \rangle$ which can be used for inferences. As shown in [KI98], $ME(\langle \Sigma, R \rangle)$ can be written in the form

$$ME(\langle \Sigma, R \rangle)(\omega) = \alpha_0 \prod_{\substack{1 \leqslant i \leqslant n \\ \omega \models A_i B_i}} \alpha_i^{1-x_i} \prod_{\substack{1 \leqslant i \leqslant n \\ \omega \models A_i \overline{B_i}}} \alpha_i^{-x_i} \tag{1}$$

with the α_i's being chosen appropriately so as to satisfy all of the conditional constraints in R.

Thus, also in the ME-approach to probabilistic logic, the preferred model selection for a specification $\langle \Sigma, R \rangle$ is motivated by improved reasoning facilities. Based on the single model $ME(\langle \Sigma, R \rangle)$ selected, the (incomplete) knowledge given by R is completed in an information-theoretically optimal way: Maximizing entropy in $Mod_{\mathcal{C}}(\langle \Sigma, R \rangle)$ means to permit as much indeterminateness as possible, so that R be represented most faithfully, without external knowledge being added.

4.3 The Inverse Representation Problem

In system and software development, one generally starts with a set of requirements that have to be specified, further refined, revised, implemented, etc., until one arrives at a model that (hopefully) meets all the requirements. Using specifications along this way, various instances of the representation problem will arise.

For probabilistic conditionals, there is also another line of development. Given (possibly large) sets of data, statistical information can be generated from it, giving us a probability distribution and thus a model. What one is interested in the

area of knowledge discovery in data (KDD) and data mining is to find a set of peculiarly interesting or *relevant* rules that hold in the given data and may thus be taken as a representation for it. In classical KDD tasks, sentences or rules are often considered to be relevant if they have a certain statistical significance. In mathematical contexts, one often seeks for a minimal set of sentences (or *axioms*) describing a (set of) preferred model(s). When axiomatizing the essential properties of a concrete data type in an ADT specification, one might aim at a set of equations that is confluent and terminating when interpreted as a set of rewrite rules, while minimality of the set of equations is not important. A further characterization of relevant sentences is to require syntactical simplicity: a single-headed conditional with only one literal in its conclusion is likely to be more interesting than a conditional containing a complex formula in its conclusion. So, in general, the notion of relevance depends heavily on the corresponding application and is – at least in KDD – often frequency-based.

From a more abstract point of view, however, the problem of discovering relevant relationships (in data or in models) can be seen as the problem to compute a set of formulas which represents a given model (or a given set of models) according to some (inductive) representation method. Therefore, we propose to call this the *inverse representation problem*:

> Given a set of Σ-models $M \subseteq Mod(\langle \Sigma, F \rangle)$, find a set of (relevant) sentences F such that the specification $\langle \Sigma, F \rangle$ has M as its desired representation.

As before, M may be a singleton or an arbitrary subset of $Mod(\langle \Sigma, F \rangle)$.

Note that by viewing knowledge discovery as an inverse representation problem, the vague notion of *relevance* is given a more precise meaning: relevance here means relevance with respect to a particular representation method. It can be sharpened by combining it with a demand for minimality, in order to find a kind of a *base* for the given model (or given models, respectively), as in mathematical contexts. Alternatively, one can focus on computing rules with a simple syntax to make the discovered knowledge most expressive and clear. So, although the inverse representation problem provides a clear formal frame for knowledge discovering, context-dependent aspects can also be taken into regard.

Let us consider again the logic $Inst_{\mathcal{C}}$ of probabilistic conditionals. We can rephrase the inverse representation problem within the ME-framework as follows:

> Given a probability distribution $P \in Mod_{\mathcal{C}}(\Sigma)$, find a set of rules R such that $P \in Mod_{\mathcal{C}}(\langle \Sigma, R \rangle)$ and such that the entropy of P is maximal in $Mod_{\mathcal{C}}(\langle \Sigma, R \rangle)$, i.e. $P = ME(\langle \Sigma, R \rangle)$.

Whereas previously no tool had been known that helps one to find such an ME-optimal set of rules, in [KI00] a general approach to solve the inverse representation problem was presented which works for ME-representation and related methods. In the following, an overview of this approach for the ME-framework (see also [KI01c]) will be given.

5 Discovery of Knowledge Structures

In this section, we briefly describe and exemplify a method which can be used to compute a concise set of probabilistic conditionals R from a given distribution P over a signature Σ such that $P = ME(\langle \Sigma, R \rangle)$. The basic idea is to exploit numerical relationships as manifestations of interactions of underlying conditional knowledge. Our approach differs from usual knowledge discovery and data mining methods in that it takes explicitly inductive representation, or inference, respectively, into consideration. It is not based on observing conditional independencies, but aims at learning relevant conditional dependencies in a non-heuristic way. As a further novelty, our method does not compute single, isolated rules, but yields as a result a set of rules in taking into account highly complex interactions of rules. Since *single-elementary conditionals*, i.e. conditionals whose antecedents are conjunctions of literals, and whose consequents consist of a single literal, are often found to be particularly interesting and informative, we focus on computing sets R containing only single-elementary conditionals in the sequel.

To simplify notation, we will mostly omit the signature subscript Σ as well as the $\mathcal{C}$ subscript, since both are clear from the context. We will consider also *structural conditionals*, i.e. conditionals $(B|A)$ with $A, B \in Sen_{\mathcal{B}}(\Sigma)$, without attached probabilities. So, in order to make differences clear, we will denote sets of probabilistic conditionals with R^*, and the corresponding sets of structural conditionals with R, and vice versa. In particular, the ME-optimal distribution appertaining to a set R^* of probabilistic conditionals will be denoted by $ME(R^*)$, and we will say that $ME(R^*)$ is *generated* by R^*.

5.1 The Footprints of Conditionals

In Section 2.3, we argued that, in particular in complex domains, basing relevance of rules on frequencies is sometimes unsatisfactory and inadequate. Instead, we asked what the "footprints" are conditionals leave on probabilities when being learned inductively. To answer this question, we first take a structural look on conditionals, bare of numerical values, that is, we focus on the set $R = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ of measure-free conditionals. Conditionals are quite extraordinary pieces of knowledge, flexible and dynamic yet difficult to handle. First, they are objects of a non-boolean nature, demanding for non-classical representation techniques. Second, they are different from material implication in classical logic, in general not describing certain knowledge but plausible conclusions. An adequate way to model its non-classical uncertainty is to represent a conditional $(B|A)$ as a *generalized indicator function* on worlds $\omega \in \Omega$, setting

$$(B|A)(\omega) = \begin{cases} 1 & : \quad \omega \models AB \\ 0 & : \quad \omega \models A\overline{B} \\ u & : \quad \omega \models \overline{A} \end{cases} \tag{2}$$

where u stands for *unknown* (cf. [DeF74,Cal91]). When we consider (finite) sets of conditionals $R = \{(B_1|A_1), \ldots, (B_n|A_n)\}$, we have to modify (2) appropriately

to identify the impact of each conditional in R on worlds ω in Ω. So to each conditional $(B_i|A_i)$ in R, we associate two abstract symbols $\mathbf{a}_i^+, \mathbf{a}_i^-$, symbolizing a (possibly) positive effect on verifying worlds and a (possibly) negative effect on falsifying worlds:

$$\sigma_i(\omega) = \begin{cases} \mathbf{a}_i^+ & \text{if} \quad (B_i|A_i)(\omega) = 1 \\ \mathbf{a}_i^- & \text{if} \quad (B_i|A_i)(\omega) = 0 \\ 1 & \text{if} \quad (B_i|A_i)(\omega) = u \end{cases} \tag{3}$$

$\mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^-$ are taken as generators of a (free abelian) group $\mathcal{F}_R = \langle \mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^- \rangle$, so that the neutral group element, 1, corresponds to the neutral behavior of the conditionals on worlds (i.e. $(B_i|A_i)(\omega) = u$). Since all $\sigma_i(\omega)$ are group elements, we can form (arbitrary) products. The function $\sigma_R : \Omega \to \mathcal{F}_R$, defined by

$$\sigma_R(\omega) = \prod_{1 \leqslant i \leqslant n} \sigma_i(\omega) = \prod_{\substack{1 \leqslant i \leqslant n \\ \omega \models A_i B_i}} \mathbf{a}_i^+ \prod_{\substack{1 \leqslant i \leqslant n \\ \omega \models A_i \overline{B}_i}} \mathbf{a}_i^- \tag{4}$$

describes the all-over effect of R on ω. $\sigma_R(\omega)$ is called *(a representation of) the conditional structure of ω with respect to R.*

Example 5. Let $R = \{(c|a), (c|b)\}$, where a, b, c are atoms. We associate $\mathbf{a}_1^+, \mathbf{a}_1^-$ with the first conditional, $(c|a)$, and $\mathbf{a}_2^+, \mathbf{a}_2^-$ with the second one, $(c|b)$. Since $\omega = abc$ verifies both conditionals, we obtain $\sigma_R(abc) = \mathbf{a}_1^+ \mathbf{a}_2^+$. In the same way, e.g., $\sigma_R(ab\overline{c}) = \mathbf{a}_1^- \mathbf{a}_2^-$, $\sigma_R(a\overline{b}c) = \mathbf{a}_1^+$ and $\sigma_R(\overline{a}b\overline{c}) = \mathbf{a}_2^-$.

Notice the striking similarity between (4) and (1) – in (1), the abstract symbols $\mathbf{a}_i^+, \mathbf{a}_i^-$ of (4) have been replaced by the numerical values $\alpha_i^{1-x_i}$ and $\alpha_i^{-x_i}$, respectively (α_0 is simply a normalizing factor). Therefore, the ME-distribution $ME(R^*)$ follows the conditional structure of worlds with respect to the conditionals in R^* and is thus most adequate to represent probabilistic conditional knowledge. The α_i's bear the crucial conditional information, and $\alpha_i^{1-x_i}, \alpha_i^{-x_i}$ are the "footprints" left on the probabilities when ME-learning R^* (also cf. [KI98]).

In the following, we will make use of these ideas and prepare the theoretical ground for the data mining techniques to be presented in the next two sections.

5.2 Conditional Indifference

To comply with the group theoretical structure of $\mathcal{F}_R$, we introduce the free abelian group $\widehat{\Omega} := \langle \omega \mid \omega \in \Omega \rangle$, generated by all $\omega \in \Omega$, and consisting of all words $\widehat{\omega} = {\omega_1}^{r_1} \ldots {\omega_m}^{r_m}$ with $\omega_1, \ldots, \omega_m \in \Omega$ and integers $r_1, \ldots r_m$. We will often use fractional representations for the elements of $\widehat{\Omega}$, that is, for instance, we will write $\frac{\omega_1}{\omega_2}$ instead of $\omega_1 \omega_2^{-1}$. Now σ_R may be extended to $\widehat{\Omega}$ in a straightforward manner by setting $\sigma_R(\widehat{\omega}) = \sigma_R(\omega_1)^{r_1} \ldots \sigma_R(\omega_m)^{r_m}$, yielding a *homomorphism of groups* $\sigma_R : \widehat{\Omega} \to \mathcal{F}_R$. Similarly, we extend probability

distributions P to homomorphisms $P : \widehat{\Omega} \to (\mathbb{R}^+, \cdot)$ from $\widehat{\Omega}$ into the non-negative real numbers by setting $P(\omega_1{}^{r_1} \ldots \omega_m{}^{r_m}) = P(\omega_1)^{r_1} \cdot \ldots \cdot P(\omega_m)^{r_m}$. Having the same conditional structure defines an equivalence relation $\equiv_R$ on $\widehat{\Omega}$: $\widehat{\omega}_1 \equiv_R \widehat{\omega}_2$ iff $\sigma_R(\widehat{\omega}_1) = \sigma_R(\widehat{\omega}_2)$, i.e. iff $\widehat{\omega}_1 \widehat{\omega}_2^{-1} \in ker\, \sigma_R := \{\widehat{\omega} \in \widehat{\Omega} \mid \sigma_R(\widehat{\omega}) = 1\}$. Thus, the kernel of σ_R plays an important part in identifying the conditional structure of elements $\widehat{\omega} \in \widehat{\Omega}$.

Besides the explicit representation of knowledge by R, also the implicit normalizing constraint $P(\top|\top) = 1$ has to be taken into account. It is easy to check that $ker\, \sigma_{(\top|\top)} = \widehat{\Omega}_0 := \{\widehat{\omega} = \omega_1{}^{r_1} \cdot \ldots \cdot \omega_m{}^{r_m} \in \widehat{\Omega} \mid \sum_{j=1}^{m} r_j = 0\}$. Two elements $\widehat{\omega}_1 = \omega_1^{r_1} \ldots \omega_m^{r_m}$, $\widehat{\omega}_2 = \nu_1^{s_1} \ldots \nu_p^{s_p} \in \widehat{\Omega}$ are equivalent modulo $\widehat{\Omega}_0$, $\widehat{\omega}_1 \equiv_\top \widehat{\omega}_2$, iff $\widehat{\omega}_1 \widehat{\Omega}_0 = \widehat{\omega}_2 \widehat{\Omega}_0$, i.e. iff $\sum_{1 \leqslant j \leqslant m} r_j = \sum_{1 \leqslant k \leqslant p} s_k$. This means that $\widehat{\omega}_1$ and $\widehat{\omega}_2$ are equivalent modulo $\widehat{\Omega}_0$ iff they both are a (cancelled) product of the same number of generators, each generator being counted with its corresponding exponent. Therefore, we set $ker_0\, \sigma_R := ker\, \sigma_R \cap \widehat{\Omega}_0 = ker\, \sigma_{R \cup \{(\top|\top)\}}$.

Now, this formalization allows us to make clear what it means that a distribution P "follows the conditional structures" imposed by R in the next definition. To make it a very concise one, we focus on the principal case of positive distributions (for the general definition, cf. [KI01b]).

Definition 2. *Suppose P is a (positive) probability distribution, and let $R = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ be a set of structural conditionals. P is* (conditionally) indifferent with respect to *R iff $P(\widehat{\omega}_1) = P(\widehat{\omega}_2)$, whenever $\widehat{\omega}_1 \equiv_R \widehat{\omega}_2$ and $\widehat{\omega}_1 \equiv_\top \widehat{\omega}_2$, for all $\widehat{\omega}_1, \widehat{\omega}_2 \in \widehat{\Omega}$.*

If P is indifferent with respect to R, then it does not distinguish between elements $\widehat{\omega}_1 \equiv_\top \widehat{\omega}_2$ with the same conditional structure with respect to R. Conversely, any deviation $P(\widehat{\omega}) \neq 1$ can be explained by the conditionals in R acting on $\widehat{\omega}$ in a non-balanced way. Note that the notion of indifference only aims at observing conditional structures, without making use of any probabilities associated with the conditionals. The following proposition shows, that conditional indifference establishes a connection between the kernels $ker_0\, \sigma_R$ and $ker_0\, P := \{\widehat{\omega} \in \widehat{\Omega}_0 \mid P(\widehat{\omega}) = 1\}$ which will be crucial to elaborate conditional structures:

Proposition 5. *A probability distribution P is indifferent with respect to a (finite) set R of structural conditionals iff $ker_0\, \sigma_R \subseteq ker_0\, P$.*

Any ME-distribution is indifferent with respect to its generating set of conditionals:

Proposition 6. *Let R^* be a (finite) set of probabilistic conditionals with structural (i.e. measure-free) counterpart R, and let $P^* = ME(R^*)$ the ME-distribution generated by R^*. Then P^* is indifferent with respect to R.*

The proof of this proposition is straightforward by observing (1).

Example 6. We continue Example 5. Here we observe

$$\sigma_R\left(\frac{abc \cdot \overline{a}\overline{b}\overline{c}}{a\overline{b}c \cdot \overline{a}bc}\right) = \frac{\sigma_R(abc) \cdot \sigma_R(\overline{a}\overline{b}\overline{c})}{\sigma_R(a\overline{b}c) \cdot \sigma_R(\overline{a}bc)} = \frac{\mathbf{a}_1^+ \mathbf{a}_2^+ \cdot 1}{\mathbf{a}_1^+ \cdot \mathbf{a}_2^+} = 1,$$

that is, $\frac{abc \cdot \overline{a}\overline{b}\overline{c}}{\overline{a}\overline{b}c \cdot \overline{a}bc} \in ker_0\ \sigma_R$. Then any ME-representation $P^* = ME(\{(c|a)[x], (c|b)[y]\})$ with $x, y \in [0,1]$ will fulfill $P^*\left(\frac{abc \cdot \overline{a}\overline{b}\overline{c}}{\overline{a}\overline{b}c \cdot \overline{a}bc}\right) = 1$, i.e. $P^*(abc)P^*(\overline{a}\overline{b}\overline{c}) = P^*(a\overline{b}c)P^*(\overline{a}bc)$.

5.3 An Algorithm for the Discovery of Rules

The concept of conditional structures, however, is not only an algebraic means to judge well-behavedness with respect to conditionals [KI01a]. As group elements, they make conditional effects on worlds computable and thereby allow us to study interactions between different conditionals in R^*. On (multis)sets of worlds (i.e. elements of $\widehat{\Omega}$), we may observe cancellations or accumulations of conditional impacts which are reflected by the corresponding ME-representation (see Example 6 above). Conversely, finding a set of rules which is able to represent a given probability distribution P via ME-methods can be done by elaborating numerical relationships in P, interpreting them as manifestations of underlying conditional dependencies. The procedure to discover appropriate sets of rules is sketched in the following and will be illustrated by an example application in the next section (for full details of the algorithm, we refer to [KI01b]):

- Start with a set B of single-elementary rules the length of which is considered to be large enough to capture all relevant dependencies. Ideally, B would consist of rules whose antecedents have maximal length (i.e. number of variables -1).
- Search for numerical relationships in P by investigating which products of probabilities match, in order to calculate $ker_0\ P$.
- Compute the corresponding conditional structures with respect to B, yielding equations of group elements in $\mathcal{F}_B$.
- Solve these equations by forming appropriate factor groups of $\mathcal{F}_B$.
- Building these factor groups correspond to eliminating and joining the basic conditionals in B to make their information more concise, in accordance with the numerical structure of P. Actually, the antecedents of the conditionals in B are shortened so as to comply with the numerical relationships in P.

As strange as this connection between knowledge discovery and group theory might appear at first sight, it is obvious from an abstract and methodological point of view: Considering knowledge discovery as an operation inverse to inductive knowledge representation, the use of group theoretical means to realize invertability is bold, but straightforward. Moreover, the joint impact of conditionals and their interactions can be symbolized by products and quotients, respectively. Their handling in a group theoretical structure allows a systematic disentangling of highly complex conditional interaction, thereby presenting a completely new approach to discover "structures of knowledge".

5.4 An Application Example

We will now illustrate the method described in the previous section by an example. Given some positive probability distribution P', we will show how to calcu-

late efficiently a set S^* of (probabilistic) conditionals such that $P' = ME(S^*)$. P' is indifferent with respect to each such set S^*, so we have $ker_0\ \sigma_S \subseteq ker_0\ P'$, thereby relating numerical relationships to structural information about the relevant conditionals.

Example 7. Let $\Sigma' = \{a, b, c\}$ be the propositional signature introduced in Example 1, i.e. with the three propositional variables a – *being a student*, b – *being young*, and c – *being unmarried*. Further, let $P' \in Mod_{\mathcal{C}}(\Sigma')$ be the distribution given in Example 2 which we repeat here for convenience:

ω'	$P'(\omega')$	ω'	$P'(\omega')$	ω'	$P'(\omega')$	ω'	$P'(\omega')$
abc	0.1950	$ab\overline{c}$	0.1758	$a\overline{b}c$	0.0408	$a\overline{b}\overline{c}$	0.0519
$\overline{a}bc$	0.1528	$\overline{a}b\overline{c}$	0.1378	$\overline{a}\overline{b}c$	0.1081	$\overline{a}\overline{b}\overline{c}$	0.1378

Important relationships between probabilities are revealed by

$$P'(\overline{a}b\overline{c}) = P'(\overline{a}\overline{b}\overline{c}), \quad \frac{P'(abc)}{P'(ab\overline{c})} = \frac{P'(\overline{a}bc)}{P'(\overline{a}b\overline{c})}, \quad \frac{P'(a\overline{b}c)}{P'(a\overline{b}\overline{c})} = \frac{P'(\overline{a}\overline{b}c)}{P'(\overline{a}\overline{b}\overline{c})}.$$

We list the twelve basic single-elementary conditionals of B which we start with:

$$\begin{array}{lll} \psi_{a,0}{=}(a|\overline{b}\overline{c}) & \psi_{b,0}{=}(b|\overline{a}\,\overline{c}) & \psi_{c,0}{=}(c|\overline{a}\overline{b}) \\ \psi_{a,1}{=}(a|\overline{b}c) & \psi_{b,1}{=}(b|\overline{a}c) & \psi_{c,1}{=}(c|\overline{a}b) \\ \psi_{a,2}{=}(a|b\overline{c}) & \psi_{b,2}{=}(b|a\overline{c}) & \psi_{c,2}{=}(c|a\overline{b}) \\ \psi_{a,3}{=}(a|bc) & \psi_{b,3}{=}(b|ac) & \psi_{c,3}{=}(c|ab) \end{array}$$

with corresponding generators $\mathbf{b}^+_{v,l}, \mathbf{b}^-_{v,l} (v \in \Sigma, 1 \leqslant l \leqslant 3)$ of $\mathcal{F}_B$. The numerical relationships yield the following group equations :

$$\begin{array}{lll} \mathbf{b}^-_{a,2}\mathbf{b}^+_{b,0}\mathbf{b}^-_{c,1} & = \sigma_B(\overline{a}b\overline{c}) & \\ & = \sigma_B(\overline{a}\overline{b}\overline{c}) & = \mathbf{b}^-_{a,0}\mathbf{b}^-_{b,0}\mathbf{b}^-_{c,0} \\ \mathbf{b}^+_{a,3}\mathbf{b}^+_{b,3}\mathbf{b}^+_{c,3} \cdot \mathbf{b}^-_{a,2}\mathbf{b}^+_{b,0}\mathbf{b}^-_{c,1} & = \sigma_B(abc)\sigma_B(\overline{a}b\overline{c}) & \\ & = \sigma_B(ab\overline{c})\sigma_B(\overline{a}bc) & = \mathbf{b}^+_{a,2}\mathbf{b}^+_{b,2}\mathbf{b}^-_{c,3} \cdot \mathbf{b}^-_{a,3}\mathbf{b}^+_{b,1}\mathbf{b}^+_{c,1} \\ \mathbf{b}^+_{a,1}\mathbf{b}^-_{b,3}\mathbf{b}^+_{c,2} \cdot \mathbf{b}^-_{a,0}\mathbf{b}^-_{b,0}\mathbf{b}^-_{c,0} & = \sigma_B(a\overline{b}c)\sigma_B(\overline{a}\overline{b}\overline{c}) & \\ & = \sigma_B(a\overline{b}\overline{c})\sigma_B(\overline{a}\overline{b}c) & = \mathbf{b}^+_{a,0}\mathbf{b}^-_{b,2}\mathbf{b}^-_{c,2} \cdot \mathbf{b}^-_{a,1}\mathbf{b}^-_{b,1}\mathbf{b}^+_{c,0} \end{array}$$

Considering these equations for each atom a, b, c separately and omitting the $\{+,-\}$-signs, we obtain the following equivalences modulo an appropriate subgroup of $\mathcal{F}_B$:

$$\mathbf{b}_{a,0} \equiv \mathbf{b}_{a,1} \equiv \mathbf{b}_{a,2} \equiv \mathbf{b}_{a,3},\ \mathbf{b}_{c,0} \equiv \mathbf{b}_{c,1} \equiv \mathbf{b}_{c,2} \equiv \mathbf{b}_{c,3},\ \mathbf{b}_{b,0} \equiv 1,\ \mathbf{b}_{b,3} \equiv \mathbf{b}_{b,1}\mathbf{b}_{b,2}.$$

Eliminating $\psi_{b,0}$ (due to $\mathbf{b}_{b,0} \equiv 1$) and joining conditionals according to these equations results in the set $S = \{(a|\top), (c|\top), (b|a), (b|c)\}$ of structural conditionals. Associating the proper probabilities (which are directly computable from P') with these structural conditionals, we obtain

$$S^* = \{(a|\top)[0.4635],\ (c|\top)[0.4967],\ (b|a)[0.8],\ (b|c)[0.7]\}$$

as an ME-generating set for P', i.e. $P' = ME(S^*)$. That means, that these four probabilistic conditionals represent P' with respect to the *ME*-method. The twelve conditionals of the set B which we started with have been modified, and their number has been reduced considerably, so as to obtain a much more concise set as *ME*-representation of P'.

6 Conclusions and Further Work

In this paper, we choose the abstract logical concept of institutions as a theoretical framework for probabilistic conditionals. We described how inductive representation of and reasoning with conditional knowledge can be looked upon as an instance of quite a general representation problem. Moreover, we showed that the crucial problem of discovering relevant conditional relationships in statistical data can also be addressed in this formal framework, namely, by considering knowledge discovery as an operation which is inverse to inductive knowledge representation. This gave rise to phrasing the inverse representation problem. In order to exemplify our ideas, we made use of the information-theoretical principle of maximum entropy as a most appropriate inductive representation method in probabilistics. We briefly described an approach to compute sets of conditionals from statistical data, which are optimal with respect to this principle. Here, the notion of conditional structures proved to be most helpful to search for the "footprints" giving evidence for conditional relationships in statistical information. This connection between formal logical work and practical uncertain reasoning is part of the CONDOR project and will be continued there. In particular, we will elaborate these ideas also in ordinal frameworks.

References

[AMS+96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in knowledge discovery and data mining*, pages 307–328. MIT Press, Cambridge, Mass., 1996.

[BG80] R. Burstall and J. Goguen. The semantics of Clear, a specification language. In *Proceedings of the 1979 Copenhagen Winterschool on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332, Berlin, 1980. Springer-Verlag.

[BHP+92] C. Beierle, U. Hedtstück, U. Pletat, P. H. Schmitt, and J. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55(2–3):149–191, 1992.

[BKI02] C. Beierle and G. Kern-Isberner. Looking at probabilistic conditionals from an institutional point of view. In *Workshop Conditionals, Information, and Inference*. Hagen, 2002.

[Bun96] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):195–210, 1996.

[BV87] C. Beierle and A. Voss. Viewing implementations as an institution. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *Lecture Notes in Computer Science*, Berlin, 1987. Springer-Verlag.

[BV91] C. Beierle and A. Voss. Stepwise software development: Combining axiomatic and algorithmic approaches in algebraic specifications. *Technology and Science of Informatics*, 10(1):35–51, January 1991.

[Cal91] P.G. Calabrese. Deduction and inference using conditional logic and probability. In I.R. Goodman, M.M. Gupta, H.T. Nguyen, and G.S. Rogers, editors, *Conditional Logic in Expert Systems*, pages 71–100. Elsevier, North Holland, 1991.

[CDLS99] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic networks and expert systems*. Springer, New York Berlin Heidelberg, 1999.

[CFH95] G. Crocco, L. Fariñas del Cerro, and A. Herzig, editors. *Conditionals: From Philosophy to Computer Science*. Studies in Logic and Computation. Oxford University Press, 1995.

[CH92] G.F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9:309–347, 1992.

[Csi75] I. Csiszár. I-divergence geometry of probability distributions and minimization problems. *Ann. Prob.*, 3:146–158, 1975.

[DeF74] B. DeFinetti. *Theory of Probability*, volume 1,2. John Wiley and Sons, New York, 1974.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1 – Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science. Volume 6, Springer-Verlag, Berlin, 1985.

[GB92] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, January 1992.

[Gei92] D. Geiger. An entropy-based learning algorithm of bayesian conditional trees. In *Proceedings Eighth Conference on Uncertainty in Artificial Intelligence*, pages 92–97, 1992.

[GR02] J. A. Goguen and G. Rosu. Institution morphisms. In D. Sannella, editor, *Festschrift for Rod Burstall*. 2002. (to appear).

[GT00] J. Goguen and W. Tracz. An implementation-oriented semantics for module composition. In G. Leavens and M. Sitaraman, editors, *Foundations of Component-based Systems*, pages 231–263. Cambridge, 2000.

[HC90] E. Herskovits and G. Cooper. Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. Technical Report KSL-90-22, Knowledge Systems Laboratory, 1990.

[Hec96] D. Heckerman. Bayesian networks for knowledge discovery. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in knowledge discovery and data mining*. MIT Press, Cambridge, Mass., 1996.

[HS73] H. Herrlich and G. E. Strecker. *Category theory*. Allyn and Bacon, Boston, 1973.

[KI98] G. Kern-Isberner. Characterizing the principle of minimum cross-entropy within a conditional-logical framework. *Artificial Intelligence*, 98:169–208, 1998.

[KI00] G. Kern-Isberner. Solving the inverse representation problem. In *Proceedings 14th European Conference on Artificial Intelligence, ECAI'2000*, pages 581–585, Berlin, 2000. IOS Press.

[KI01a] G. Kern-Isberner. Conditional preservation and conditional indifference. *Journal of Applied Non-Classical Logics*, 11(1-2):85–106, 2001.

[KI01b] G. Kern-Isberner. *Conditionals in nonmonotonic reasoning and belief revision.* Springer, Lecture Notes in Artificial Intelligence LNAI 2087, 2001.

[KI01c] G. Kern-Isberner. Discovering most informative rules from data. In *Proceedings International Conference on Intelligent Agents, Web Technologies and Internet Commerce, IAWTIC'2001*, 2001.

[KIR96] G. Kern-Isberner and H.P. Reidmacher. Interpreting a contingency table by rules. *International Journal of Intelligent Systems*, 11(6), 1996.

[KLM90] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

[Mac72] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1972.

[Mak89] D. Makinson. General theory of cumulative inference. In M. Reinfrank et al., editors, *Non-monotonic Reasoning*, pages 1–18. Springer Lecture Notes on Artificial Intelligence 346, Berlin, 1989.

[MS98] N. Megiddo and R. Srikant. Discovering predictive association rules. In *Proceedings of the 4th International Conference on Knowledge Discovery in Databases and Data Mining*, 1998.

[Par94] J.B. Paris. *The uncertain reasoner's companion – A mathematical perspective.* Cambridge University Press, 1994.

[PV90] J.B. Paris and A. Vencovská. A note on the inevitability of maximum entropy. *International Journal of Approximate Reasoning*, 14:183–223, 1990.

[RKI97] W. Rödder and G. Kern-Isberner. Representation and extraction of information by probabilistic logic. *Information Systems*, 21(8):637–652, 1997.

[SGS93] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search.* Number 81 in Lecture Notes in Statistics. Springer, New York Berlin Heidelberg, 1993.

[Sho87] Y. Shoham. A semantical approach to non-monotonic logics. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI'87*, 1987.

[SJ80] J.E. Shore and R.W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, IT-26:26–37, 1980.

[ST97] D. Sannella and A. Tarlecki. Essential comcepts for algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.

[Tar96] A. Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specifications*, volume 1130 of *Lecture Notes in Computer Science*, pages 478–502, Berlin, 1996. Springer-Verlag.

[Wir90] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier Science Publishers B.V., 1990.

Time for Thinking Big in AI

Wolfgang Bibel

TU Darmstadt, Computer Science, Intellectics Group,
Alexanderstr. 10, D-64283 Darmstadt, Germany
bibel@informatik.tu-darmstadt.de

1 Introduction

Today it is commonplace to talk about the revolution going on in our society, driven by information technology. Through talking about the ongoing changes people seem to compensate their uneasiness with the restlessness in this world. Few of those heralding the future developments are actually looking further ahead than perhaps five or ten years. And even fewer still are pondering about the consequences the coming technological changes will have for society and prepare for actions to be planned for coping with these changes.

In this paper written in honor of Jörg Siekmann's sixtieth birthday I want to alert the community for some of the grand tasks which for our field lie not so far ahead on our road into the future. The main message is that our field carries a great responsibility for attacking fundamental questions which have been on the minds of people throughout thousands of years. Only now do we have the methodology and technology to make real progress towards satisfactory answers.

The article starts in the next section with a brief summary of the vision behind Intellectics, the field of Artificial Intelligence (AI) and Cognitive Science, and of the fundamental scientific questions which have come under our responsibility through the success of the research paradigm underlying Intellectics. In Section 3 some of these successes are briefly described which indicate a great maturity of the theoretical and methodological basis of the field reached during its brief history. If after so many successes we would now become more courageous and pursue grander goals than hitherto, a major technological issue is the direction we should follow: do we need just smart ideas to build really intelligent systems or will any such system be necessarily huge. In Section 4 we give arguments for a combination of both directions.

On the basis of all these preparations we finally describe a number of challenge projects with enormous importance for mankind as a whole. Due to the general relevance of intelligence and hence of AI-technology these projects affect basically all other disciplines, in the spirit of the great intellectician Woody Bledsoe who said: "In the end AI will be the only science". Which does not at all mean that our field wants to overtake all these other disciplines. Rather it means that the Intellectics paradigm is the one which finally has emerged as the succeeding one.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 120–131, 2005.

2 Intellectics

Scientific research today produces knowledge at an enormous scale. The volumes of thousands of journals are filled each year with the achieved results. No single scientist is able to oversee the resulting body of science.

However if one looks closer at this confusing variety one recognizes that the scientific enterprise as a whole actually poses only a very few really deep questions. All the other explorations may be seen as sidetracks from the road to answering these basic questions.

What is matter? is one of these fundamental questions. *What are the forces?* and *What is space and time*? are further ones. In fact these fundamental questions have still no really satisfactory answers. And because of that physicists are continuing with their observations of nature and with their experiments. In the pursuit of searching for these answers myriads of results have been found and applied successfully – sometimes to the benefit of mankind. In this sense the fundamental questions proved to be extremely fruitful although perhaps too hard to be ever solved completely. Besides physics other disciplines were created along the way to account for special aspects of matter and forces, such as chemistry, astronomy, geology and so forth. These "physical" disciplines altogether provide insight into many phenomena in this world, but not into all.

What is life? is another fundamental question and one of those which (so far) has stayed out of reach for any physical theory. There must be something particular beyond matter and forces which characterizes species such as man. Of course, a dog consists of matter which is subject to the physical forces. But why do dogs have offspring while stones don't? Biology is the scientific discipline competent for these sorts of questions. And again there are several related ("biological") disciplines such as botany, zoology, physiology and so forth. They together with the physical disciplines provide insight into even more phenomena in this world, but again not into all.

What are the mind and the psyche? is the most prominent fundamental question which has stayed completely out of reach for any biological and physical theory, and the same is true for all aspects related to it. This question as well as all those mentioned so far have occupied the thinking for thousands of years. The oldest extensive treatments which are still accessible to us are those from Aristoteles, Platon and others from that time. They may be regarded as the founders of the discipline regarding itself as competent in all of these questions, namely philosophy. In the course of time again other disciplines grew out of it, classified together as the humanities.

The answers of the old philosophers are completely outdated today as far as the physical and biological questions are concerned. Philosophy therefore retracted its focus to the mental and psychological sphere in which Aristotle's and Platon's answers are still discussed in ernest and cited along with modern publications. This shows that the progress in the pursuit of studying mind, psyche and the phenomena related with them such as human relations, society, history and so forth has been less spectacular to put it mildly.

The advent of the modern computer immediately caused a revolution in the paradigm with which these issues were attacked, now by scientists trained in disciplines not related with philosophy and the humanities. Alan Turing, John von Neumann, John McCarthy, Marvin Minsky, Herbert Simon, Alan Newell are some of those who in the middle of the last century laid the foundation for the new paradigm with which these questions are studied under a completely new perspective. While there was literally no measurable progress in two thousand years of research under the philosophical paradigm, the last half century has already provided us with revolutionary insights into the functioning of mind and psyche. On this basis it is now possible to penetrate into further domains which to this day continue to be dominated by the old paradigms.

John McCarthy is usually mentioned as the inventor of the name for the discipline which has been founded on this new paradigm, namely Artificial Intelligence or AI. But that is not the full story. He prepared a proposal for the famous Dartmouth workshop in 1956 with a title mentioning "artificial intelligence" which was not meant by him as the name of such a new discipline however. In fact later he tried to introduce Cognetics as such a name. But the buzzword AI had already caught on and has stayed until this day.

The goals of the discipline of AI are to understand intelligence, mind and psyche in a way so as to be able to produce these phenomena even artificially. In other words, artificial intelligence is what this discipline wants to achieve (among other things) and is therefore a perfect name for its *goal* rather than for the discipline itself. But for strange reasons AI became the only discipline which uses the name of its goal at the same time for its own name. Since this is strange indeed and unique in the concert of disciplines and their names, I proposed *intellectics* as its name instead of AI. As long as no better proposal is put forward I will continue to stay with this proposal which, after two decades of familiarity, to me still sounds perfectly.

Intellectics then is the discipline for the study of mind and psyche in biological creatures such as man under the paradigm of intelligent information processing which, as we said, includes the development of artificial systems with intelligent features. Although we mentioned the key players whose ideas directly lead to Intellectics as a discipline they were of course embedded in the context of a particular Zeitgeist which produced this shift in paradigm and could be traced in various disciplines. Warren McCulloch, Walter Pitts, Donald Hebb and Claude Shannon are usually mentioned as pioneers in the years before 1956. But there are many more who would deserve to be mentioned in a comprehensive analysis of the history of Intellectics. Konrad Zuse would be among them with his Plankalkül and his chess program, the first in history (prior to Shannon's), the German school of cognitive (especially Gestalt) psychologists with names such as Otto Selz, Max Wertheimer and Karl Duncker, the physicist and physiologist Herrmann von Helmholtz, and, even further back into history, the logical tradition from Gottfried Wilhelm Leibniz to Gottlob Frege.

In the seventies it turned out that one can earn even money with intelligent systems, in fact a lot of money. This experience distracted the discipline some-

what because from there on the public identified AI with that part of intellectics which builds smart systems. Due to this one-sided focus the researchers with a remaining interest in the original questions about mind, psyche and cognition reassembled under the new label of Cognitive Science which focuses on the "non-AI" part in Intellectics. So today Intellectics may seem to consist of two parts, namely AI and cognitive science. One could as well think of these three as different names for the same thing, a position I prefer since a separation of AI from Cognitive Science would certainly hurt both.

Even when one restricts the focus on the Computer Science part of AI, its perspective has been different from, in fact complementary to, that of CS (or Informatics). Intellecticians from the beginning thought of systems simulating people in their mental functions (how they play games, prove theorems, understand language, etc.). In other words they imagined artificial agents performing such functions and tried to realize them on existing computers. Note the semantic ordering in this statement: given are first the functions and then we try to model them with current machines. Computer scientists think the other way around. They take the functionalities of computers as granted and then think about what other functionalities could be achieved with them, for instance functionalities for a more comfortable human interface. This thinking is bottom-up-oriented while that of intellecticians is top-down where man is taken to be top and machine down.

3 The Success of the New Paradigm

Why does the mind pose such a hard problem for science? Due to research in Intellectics – or more specifically in cognitive psychology, computational neurology and computer science –, we know today quite a bit about the functioning of the brain. For instance, the processes involved in color seeing have been discovered to an extent that they can be modeled and simulated with computers at a rather low level of detail, namely that of individual neurons. On the basis of these and many other insights it is clear that computational processes are heavily – if not even exclusively – involved in the functioning of mind. That is why often the analogy between the mind and a computer is drawn. Let us take this analogy to illustrate the particular difficulty in understanding the mind.

Imagine aliens coming to earth and watching us using our computers. Before they return home they take lots of notes about their observations of the functional behavior of these machines and carry some machines for further studies with them. Their goal is to understand how the machines work, without having access to our computer science literature, manuals etc. They would be in a similar position as we are with our own minds except that we in addition are conscious of our own thinking. What could they do to find such an understanding? Continue making observations about the functional behavior and forming hypotheses concerning the way this behavior is generated, just like the psychologists in the case of mind. Or open the machines, observe their static structures, trace the currencies through the chips and analyse this internal functioning of the gates, just like the neurologists. Wouldn't it be a formidably difficult task to

discover this way the principles on which modern computers are based and the details of their design? Since brains at least to some extent are computational like computers we are faced with the same kind of problems at a far higher quantitative level of complexity.

The nature of this problem consists in discovering the ideas behind a program from its functional behavior and from fragmentary observations of the flow of signals. Even for relatively simple algorithms we have not yet the right methodology for solving such a problem, let alone for "algorithms" involving trillions of neurons. As a first step it is therefore reasonable to try to analyze the functional behavior by inventing algorithms which produce it. These algorithms might be rather different from the actual ones, but at least we would get a first idea about their structure.

That is what we intellecticians do with our AI systems. For instance, neural nets were invented in this vein which became extremely helpful in modeling neural processes at a far greater detail than possible before. But basically all AI systems may serve such a purpose. These systems give us a lot of clues about human problem solving so that we can go on from there with our studies of the real system, ie. the brain. There is already considerable progress in this line of attacking the fundamental questions about mind and psyche. On the other hand, these systems have turned out to be extremely helpful in numerous applications in the real world. It is especially in this respect that Intellectics so far has been particularly successful. Since it would fill many books to compile all these successes, we can only mention a few selected ones.

Scientific disciplines often organize their research around the study of compactly understandable problems. The problem of understanding the physiological functioning of the *drosophila* in full detail is such a paradigmatic problem in biology. The fly drosophila is complex enough to make this study really interesting; on the other hand it is not that complex that with the current methodology progress is out of reach. Intellectics has had two "drosophilas" so far. The first one was chess. The study of playing chess lead to many interesting concepts for modeling intelligent behavior. On the basis of these concepts and the resulting theories eventually systems were developed which reached even the level of the best human chess players in the world. In 1997 the worldmaster in chess, Gary Kasparov, was beaten by such a chess machine (deep blue). No one claims from this that now we would "understand" how the mind is able to play chess. At the lowest level of detail there is still a huge gap between the algorithms realized in deep blue and those presumably at work in Kasparov's mind, although they behave very similar at the highest functional level. Nevertheless the high level success provides a starting point from which we now can proceed to lower levels of details for a deeper understanding (and possibly better technological performance).

The second "drosophila" of Intellectics is robocup. Once the chess problem was "settled" the next challenge needed to take a step closer to the human condition in toto. We humans are not just minds which solve problems like chess but we also have a body which must be coordinated in addition to solving real

problems in the world. Also we are social beings cooperating with others in our problem solving. All these features are given in the task of soccer playing. So in the mid-nineties the idea came up to develop artificial teams of soccer players. The longterm goal of this second "drosophila" in AI is to beat the worldmaster team in 2050. So leagues were set up and matches among teams of robots held, altogether called *robocup*. Given the few years of research, the progress is fascinating already now.

Mathematics is too sober to catch the interest of the public to an extent games such as chess or soccer do. But from an intellectual point of view beating the worldmaster in chess is far less spectacular than beating the best mathematicians within their own domain. This is what happened in 1996 when the artificial mathematicians EQP und Otter, so-called theorem provers, finally proved a conjecture [1] which in mathematics was open for sixty years and in vain attempted to be settled by a number of the best human mathematicians such as Alfred Tarski. Recall that mathematics has been called the queen of sciences and that mathematicians are for good reasons usually kept in highest regards concerning their intellectual capacities. And then imagine that a computer system outperforms some of the best mathematicians of the century by solving a problem they could not solve for such a long time. I feel that this event is a much more impressive achievement in our discipline than the chess event mentioned before.

Society longs for the spectacular shows such as robocup matches while the real progress may be experienced in many far less spectacular achievements. Let us summarize a few more of these achievements. Doug Smith with his system KIDS synthesized a logistics algorithm – among many others – which outperformed all similar algorithms published by human specialists by orders of magnitude [2]. Unnoticed even by many computer scientists, we are entering the period in which much of programming will be carried out by the computers themselves rather than by software engineers. What is left to humans is the specification of their problems in sufficient detail but without consideration of any aspects concerning the machine which today are still encoded in programming languages like C or Java. It is therefore a safe prediction that in the near future, "problem engineers" will complement – and eventually substitute – software engineers.

There are tens of thousands of systems in daily use which to some extent are based on coded knowledge of the kind we as students learn in books about any discipline whatsoever. These *knowledge-based systems* are routinely integrated in various system environments without special notice. Thirty years ago people could not even imagine that knowledge coded in such a declarative form could be processed algorithmically. Today KB systems provide for complex kinds of reasoning modes, so far the exclusive domain of people. The system CYC [3] in this way comprises general knowledge at the scale of a million knowledge facts or rules according to the state reached this year, and can activate this huge amount of general knowledge in various kindes of applications. The system Verbmobil allows the spontaneous communication between a German and a Japanese in their respective natural languages through the system's interpretation [4].

Vision systems are used in production lines for controlling the production processes, but also analyze the pictures taken of entire continents, to mention two out of hundreds of other applications. Robots are now used for rather complex tasks far beyond the restricted production manipulations applied in the previous generation. Spectacular uses could be experienced during recent space missions or during the search for victims of the terrorist attack in New York after the 11th September 2001. Systems connected to communication networks are able to trace nearly everything happening anywhere around the globe, be it a phone call, a fax, an email message etc. and analyze the data for certain features.

One could go on and on with the technological achievements enhancing our human cognitive capabilities. What is at least as impressive as the preformance of these systems is the theoretical insight into the underlying mechanisms. Obviously the spectacular systems performance is a function of the maturity of the theorical basis on which it is founded. True, sometimes engineers seem to be ahead in comparison to theoreticians. In building complex systems like those referred to here you are quickly lost however if the gap between theory and practice is more than marginal.

In fact we may observe the following pattern in the theory of Intellectics. Since so many different phenomena are involved, the theory is fragmented in many different branches. The results in each of them are applied independently from those in others although they pertain to the common phenomenon of intelligent behavior. Imagine all these disparate results were combined within a common system!

4 Big or Smart?

Huge **and** smart! In social life people tend to fight over dichotomies such as efficiency-oriented vs. socially-oriented economic politics. Is huge vs. smart such a dichotomie on the road to artificial minds? Well, like most dichotomies in politics huge and smart are not really opposites in view of intelligent systems. The central nervous system of humans has around a trillion of neurons, each forming an extremely complex physical processor, and around a hundred times as many synapses (connections). Given this unimaginable complexity one would of course not expect that all the relevant functions of the brain could be modeled by a single system comprising a few thousand lines of code. A truly intelligent system will predictably be big, even huge.

The development of such huge systems will remain a challenge, perhaps for ever. The software industry tries to make us believe that it is up to this challenge in areas like operating systems with appropriate solutions. Sincere computer scientists admit that the progress in this art is rather like that in the humanities. Not accidentally so since system development requires the cooperation of many humans hence falls in fact to some extent into the domain of the humanities. So big is a problem and huge even more so.

How about smart? Let us consider an interesting experience in this respect. Theorem proving is a thoroughly studied special field in Intellectics. Hundreds of systems, so-called theorem provers have been developed, one of which we

mentioned in the last section. Some of them took tens if not hundreds of man-years of development and comprise hundreds of thousands of lines of code. Jens Otten during one night in the year 2000 wrote a Prolog program consisting of three Prolog clauses which more a less amounts to the logical definition of the simplest calculus based on the connection method developed by the author [5, 6]. The surprising experience was the performance of this mini-prover, called leanCoP: in some sense it is comparable in performance to that of those mammut provers as careful and extensive comparisons have demonstrated [7].

leanCoP is surely a smart system in several respects. First, it is based on a proof method which, unlike any other known method, operates on a single formal representation of the problem description. The advantage of this locality feature has been ignored by the community which has preferred the more intuitive (and hence less focused) methods such as the resolution or the tableau methods as their logical basis. The connection method is also goal-oriented rather than an exhaustive method like resolution. The system also makes clever use of Prolog's features. In combination these features make the tiny system surprisingly efficient.

There is one more point to be noticed about leanCoP. The program forms a precise and declarative specification of the problem, nothing more. Recall that those supporting logic programming (like the present author) have dreamed of logical problem specifications as programs to be immediately processed by machines. Several fashion phases in software engineering after those dreams have begun I still believe that eventually this will be the only reasonable approach to system building. The logic will probably be different from the one used in Prolog, especially to account better for the dynamic character of the problems to be coped with in programming (cf. [8]). Also the system development environment will have to support the programmer, or rather the specifyer, in various novel ways unfamiliar from standard programming environments (cf. [9]). UML is a step towards such a direction but an insufficient one in many respects. And finally the synthesis and optimization systems which transform specifications to efficient code will have to be improved quite a bit further, heavily involving KB systems technology discussed in the last section.

The attractiveness of specifications in contrast to traditional programs lies in the additivity of specifications parts. An enhancement of leanCoP for a special handling of equality would amount to an addition of further Prolog clauses without affecting the definition given in the three current clauses. No other programming paradigm offers this fundamental advantage (including OO programming). If one envisions huge systems as we do here this advantage will become absolutely crucial for the success of the entire enterprise.

Another big advantage lies in the ease of maintenance. A change in the specification could be achieved simply by substituting the respective specificational part in the problem definition which is kept along with the resulting code as well as with all relevant synthesis information. The synthesis would therefore not need to be re-done in its entirety upon such a change but only the respective modifications would have to be traced down to the executive code.

Programming by specification may lead to smarter systems. That is the lesson which I wanted to suggest with the leanCoP experience. In contrast, standard programming results in monster systems like Windows, huge because of lack of appropriate organization, not so much because of its inherent functional complexity which could be achieved with much smaller and smarter systems. Good organization can only be achieved with a proper, elaborated specification available at the outset, or rather growing with the system development, and with systematic ways to synthesize such a specification into efficient code as studied in program synthesis.

Once we have such systematic ways to synthesize smart systems, then – and only then – will we be able to envision the development of huge systems needed to model the mind. In principle we know these ways already, but there is no incentive to realize them in respective development environments. The reason is the enormous investment needed prior to any benefits before such an approach becomes feasible. We are talking about such investments in the coming section.

5 Challenge Projects for Intellectics

Society has always found it worthwhile to invest enormous sums in projects related with the fundamental questions listed in Section 1. The physical accelerators built for studying physical forces and the structure of matter cost billions of dollars. The international space station (ISS) also is only secondarily of economic interest and primarily an enterprise about answering fundamental scientific questions. It as well costs billions of dollars.

In contrast to projects at this scale Intellectics projects were rather cheap so far. In addition most of the latter soon payed off. For instance, the project CYC is in the (low) hundreds of millions as is Verbmobil. Both have a great economic perspective so that they may even produce large returns before long. Similar returns from the huge investments in accelerators are not expectable in the forseeable future.

Although this might sound like arguing against big physical experiments, this would be a total misunderstanding of my point. Rather the point is that the fundamental questions pursued by Intellectics are at least as interesting for society as those pursued by Physics. Also we will not come closer to satisfactory answers unless society is ready to invest in Intellectics projects at the same scale which is billions of dollars. According to our previous experiences we might even expect an immediate economic return for society so that the investment promises to become profitable. In this final section I will briefly outline a few projects at this scale and their potential benefits for society.

I first refer to what I already outlined in the last section concerning a systematic synthesis of computer systems through precise specifications. Building a development environment of the kind envisioned there amounts to an international project with costs at the level of billions of dollars. Its benefits are obvious and immediate. Software production would be simplified by orders of magnitude. System development would be possible in a fraction of time in comparison to today's practice. It also would lead to machine provably correct systems and

thus far more reliable than current software. Maintenance would be simplified again by orders of magnitude.

The consequences of this single project alone on other disciplines and their applications would be enormous. Once we have the methodology to design and realize software at such a new level of magnitude, we could envision the grand project noted at the end of Section 3 which combines the disparate knowledge built up in the theory of Intellectics in the last decade. With the resulting smart and huge system grand problems could be attacked. Some of these are the following ones.

Following the vision and scientific work of nobel prize winner Herbert Simon, discovery science is developing at a rapid pace [10]. Imagine if we complemented the inventiveness of humans with the strengths of machines for handling massive data. Physicists, for instance, try to discover patterns in their scattering experiments in accelerators which could give clues for a unified theory of the forces. There are billions of data already available from past experiments of this kind which no human will ever study in any detail. Machines could process these amounts of data, discover such patterns and mechanically form theories accounting for them. For physicists this may sound like wishful thinking. For intellecticians this is already the bread and butter of their daily work for instance in the area of data mining where knowledge patterns are extracted from massive data. Physics is just one example. The mechanisms behind learning and theory formation are now understood good enough to apply this knowledge by machines to any of the natural sciences. But the first step for any of these applications would be to put all the knowledge available for any of these disciplines in a formalized knowledge base which can be subjected to our inferential mechanisms. This alone just for a single discipline amounts to a huge enterprise.

With our envisioned intelligent system also engineering of any kind would be revolutionized, a trend which to a limited extent and with limited system capacity is already under way. Imagine we are given an engineering problem (like building a bridge under given constraints or controlling traffic in a city). The knowledge about the domain under consideration is assumed to be available in the knowledge base of the system. The problem could then be specified using the terms of this knowledge base in the way we specify programming problems (such as theorem proving in the case of leanCoP). With the system development environment an engineering design could be synthesized like a program from this specification and the knowledge base. This is because, from an abstract point of view, the design of a program is exactly the same as the design of an engineering solution to the posed problem. Whether the quality of the solution is good enough for providing a solution in the real world depends solely on the quality of the engineering knowledge stored in the knowledge base.

Let us emphasize once more the arbitrariness of the kind of engineering applications. The fundamental design methods are all very similar at the simulation level in computers. The difference lies mostly in the relevant knowledge activated from the knowledge base if it is rich enough to cover the respective areas and includes area specific procedures. So whether we are thinking of drug design in

pharmacology or of designing an optimized burocracy for handling a university or of a strategy for improved teaching to achieve better school results or what have you, all these fall – from the point of view of Intellectics modeling in computers – under the same category of engineering problems. Engineers can hardly believe this as they rarely reflect about their own methods on a meta-level. Their strength consists in embodying that specialized knowledge combined with clever solution methods. Like most experts they are however not consciously aware of them.

As indicated with the last of these examples (teaching) Intellectics technology would finally reach also the humanities. There is already a huge amount of knowledge available in these disciplines because in the humanities in lack of theories researchers collect a lot of episodical and statistical knowledge. Just think of psychology or sociology. As in any knowledge base there is a great potential in these bodies of knowledge once they become accessible for the inferential methods from Intellectics. Theories could be induced from that knowledge, possibly so complicated ones that humans would never be able to induce them by hand due to the shere (quantitative) complexity. Explanations could be provided on the basis of that knowledge. Further knowledge could be deduced. Modeling could be performed on a solid basis. In short, the available knowledge would become activated to an extent not experienced ever before.

Particular goals of this nature consist in modeling (parts of) the body of particular human individuals on the basis of the entire knowledge available about body physiology in general and the specific body of the individual in particular which could provide precise explanations for symptoms exceeding the daily speculations of ordinary medical doctors by far. And the modeling could be done at rather low costs once the overall system is available. Or think of the modeling of the functioning of the brain in accordance with the knowledge neurologists have accumulated over the last decades. For parts like image and sounds processing this has already been done. The challenge is to combine these fragments to a coherent model which might vary considerably in the level of abstraction depending on the availability of insights. Similarly, the human psyche could be modeled on the basis of the knowledge about the functional relationships elaborated in numerous psychological studies. On the basis of both these models we could experiment about different strategies for school teaching, first in the computer and then tested in real life, ie. enter the discipline of pedagogy with quantified methods. Going a step higher, sociological structures like a small-size company could be modeled, linked to the model of the human psyche mentioned just before. This way humanities would become real sciences rather than carrying along the odor of being chat fora.

With Intellectics technology also decision processes in economics, politics and law could be laid on rational grounds. In individual projects the direction of such decision support systems has already been shown. But again, individual projects like PhD theses and one-man prototype systems are interesting for understanding the underlying principles; however, for practical purposes we need systems build on a much grander scale. In all applications just mentioned we know in principle

how to do it. But billions of dollars are needed to realize the required programs. Billions of dollars is little money compared to the amount of waste of money caused every year by irrational and hence often stupid economic and political decisions.

Some people argue that money alone will not help. The problems in such huge projects would be so hard to cope with that not enough smart people with the appropriate expertise would be available to carry the projects through to a successful end. I consider such arguments narrow-minded because they ignore the enormous expertise available in the application disciplines. As experiences with interdisciplinary projects of this kind show, people involved get really excited about the prospects of the resulting achievements and thus are highly motivated. High motivation counts much more than the question whether some participant might have attended a certain course in Computer Science or not.

Like with all scientific endeavors dangers lure around the corner of such projects of the kind of possible misuse by villains. We must face the fact that this menace is inherent in any technology. Already in the stone age, Kain could use the stone to grind the corn or to murder Abel. The same will be true for the technology envisioned here. We have to cope with this fact now as ever by prosecuting villains in order to keep their actions as restricted as possible.

References

1. William McCune. Solution of the Robbins problem. Journal of Automated Reasoning, 19(3):263-276, December 1997.
2. Douglas R. Smith. KIDS - A Knowledge-Based Software Development System. In M. R. Lowry and R. McCartney (Hg.), Automating Software Design, Kapitel 19, S. 483-514. AAAI Press, Menlo Park, 1991.
3. R. V. Guha and Douglas B. Lenat. Cyc: a midterm report. AI Magazine, ll(3):32-59, 1990.
4. Wolfgang Wahlster (Hg.). Verbmobil: Foundations of Speech-to-Speech Translation. Springer, Berlin, 2000.
5. W. Bibel. Matings in Matrices. Comm. ACM, 26:844-852, 1983.
6. W. Bibel. Deduction: Automated Logic. Academic Press, London, 1993.
7. Jens Otten and Wolfgang Bibel. leanCoP: Lean Connection-Based Theorem Proving. Journal of Symbolic Computation, 2001.
8. W. Bibel. Let's plan it deductively! Artificial Intelligence, 103(1-2):183-208, 1998.
9. W. Bibel, D. Korn, C. Kreitz, F. Kurucz, J. Otten, S. Schmitt and G. Stolpmann. A multilevel approach to program synthesis. In N. E. Fuchs (Hg.), Proceedings of the 7th Workshop on Logic Program Synthesis and Transformation (LOPSTR-97), Lecture Notes in Computer Science, Springer, Berlin, 1997.
10. Lindley Darden. Discovering Mechanisms: A Computational Philosophy of Science Perspective. In Klaus P. Jantke and Ayumi Shinohara (Hg.), Discovery Science - 4th International Conference, DS 2001, Washington DC, volume 2226 of LNAI, S. 3-15, Berlin, 2001. Springer.

Solving First-Order Constraints over the Monadic Class

Dimitri Chubarov and Andrei Voronkov

University of Manchester
{chubarov,voronkov}@cs.man.ac.uk

Abstract. First-order constraints over arbitrary theories or structures can be formalised as the formula instantiation problem as defined in [11]. Several re- sults have been previously obtained for the formula instantiation problem in the case of quantifier-free formulas of first-order logic. In this paper we prove the first general result on formula instantiation for quantified formulas, namely that formula instantiation is decidable for the monadic class without equality.

1 Introduction

Given a first-order formula $\exists\overline{x}\varphi(\overline{x})$ which expresses existence of elements with particular properties, we may be interested in obtaining these elements in an explicit form if such elements exist. Thus, we consider the formula $\varphi(\overline{x})$ as a constraint, and try to present a solution of this constraint in some explicit form. If this explicit term is simply a tuple of terms $\overline{t}$, the corresponding decision problem is called *formula instantiation*, see [11, 13].

Definition 1. *(Formula Instantiation, Witness)* Formula instantiation *is the following decision problem: given a formula $\varphi(\overline{x})$, does there exist a sequence of terms $\overline{t}$ in the signature Σ_φ of this formula such that $\varphi(\overline{t})$ is valid. Every such tuple of terms $\overline{t}$ is called a* witness *for $\varphi(\overline{x})$.* □

In other words, formula instantiation holds for $\varphi(\overline{x})$ if and only if $\varphi(\overline{x})$ has a witness. In fact, formula instantiation is a family of problems parametrized by a class of formulas (possible values for $\varphi(\overline{x})$) and a theory with respect to which validity is defined. Naturally, we can also consider the dual problem of finding witness terms which make $\varphi(\overline{t})$ satisfiable.

Formula instantiation has applications in tableau-based automatic reasoning in first-order logic. It naturally arises when one wants to close a tableau (or a tableau branch) by an appropriate substitution. The tableau, or a tableau branch can be considered as a formula $\varphi(\overline{x})$. Then formula instantiation holds for the formula $\neg\varphi(\overline{x})$ if and only if the branch (or the whole tableau) is closed. This search for a substitution that closes a quantifier-free tableau was formalized as the simultaneous rigid E-unification problem in [2]. A number of results about simultaneous rigid E-unification were proved (see [1] for an overview). The problem of formula instantiation was formulated in [11], where several results about

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 132–138, 2005.

Table 1. Complexity of formula instantiation for quantifier-free formulas.

Class	Complexity
no equality	Σ_2^p-complete
equality, binary symbols	undecidable
equality, one variable	decidable and EXPTIME-hard
Monadic signature:	
equality, general	open
equality, at most one unary symbol	decidable
one variable	PSPACE-complete

formula instantiation for quantifier-free formulas were also proved and several open problems related to formula instantiation were formulated.

Some of the results known so far are given in Table 1.

It is easy to see that the validity problem is a special case of formula instantiation, where the formula φ has no free variables. Therefore, it makes sense only to consider formula instantiation for classes with the decidable validity problem. The decidability of the validity problem does not, however, guarantee that formula instantiation is decidable. Two extreme examples of hardness of formula instantiation are the following.

1. The validity problem for quantifier-free formulas with equality is coNP-complete, but formula instantiation for quantifier-free formulas is undecidable already for formulas with two variables [9].
2. The validity problem for arbitrary ground-negative formulas with equality is Π_p^2-complete [12], but formula instantiation is undecidable even for quantifier-free ground-negative formulas with two variables [9], see also [10].

Despite a large number of results on formula instantiation for quantifier-free formulas, no nontrivial general results are known for quantified formulas. In this paper we will consider formula instantiation for the monadic class.

Definition 2. *(Monadic Class) A* monadic signature *is a signature which contains no function or relation symbols of arity greater than 1. The* monadic class *is the class of all formulas whose signature is monadic.* □

Note that formulas in the monadic class may have arbitrary quantifiers. The decidability of the monadic class was shown in [6, 3].

The main result of this paper is the following.

Theorem 3. *The formula instantiation problem for the monadic class is decidable.* □

To prove the decidability of formula instantiation for the monadic class, for each instance φ we first skolemize out all universally quantified variables in φ to obtain a formula φ' in an extended signature such that φ and φ' have the same witnesses in the signature of φ. We skolemize in such a way that only new

constants are introduced, but no function symbols, and hence the signature of φ' remains monadic. Then we note that the truth of φ' on a tuple of terms $\overline{t}$ can be expressed using a second-order formula of the term algebra in a monadic signature. Likewise, the existence of a witness in the original signature of φ can also be expressed by such a formula. Then we use the decidability of the monadic second-order theory of any term algebra in a monadic signature, which follows from the decidability of the monadic second-order theory of two successors [8].

2 Skolemization of Monadic Formulas

A formula is called *closed*, or a *sentence* if it contains no free variables. We call a *literal* an atomic formula or its negation. A formula is in *negation normal form* if it is built from literals using the connectives $\wedge$, $\vee$ and quantifiers $\forall$, $\exists$. A formula is called *prenex* if it has the form $Q\overline{x}A$, where $Q\overline{x}$ is a quantifier prefix and A a quantifier-free formula.

Our first step is to skolemize out universal quantifiers.

Lemma 4. *Let $\varphi(\overline{x})$ be a formula of a monadic signature Σ with free variables among those in $\overline{x}$. There exists a prenex existential formula $\exists\overline{y}\varphi'(\overline{x},\overline{y})$, where $\varphi'(\overline{x},\overline{y})$ is quantifier-free such that (i) $\varphi'(\overline{x},\overline{y})$ uses symbols in Σ and maybe some new constants; (ii) for all tuples of ground terms $\overline{t}$ of the signature Σ the formula $\varphi(\overline{t})$ is valid if and only if $\exists\overline{y}\varphi'(\overline{t},\overline{y})$ is valid.*

Proof. Let φ be a monadic formula. Without loss of generality we assume that φ is in negation normal form. Repeatedly using the distributivity laws and following equivalences to rewrite subformulas of φ:

$$\begin{gathered}\forall x(A \wedge B) \equiv \forall xA \wedge \forall xB;\\ \exists x(A \vee B) \equiv \exists xA \vee \exists xB;\\ \exists x(A(x) \wedge C) \equiv \exists xA(x) \wedge C;\\ \forall x(A(x) \vee C) \equiv \forall xA(x) \vee C;\\ \forall xC \equiv C;\\ \exists xC \equiv C;\end{gathered}$$

where x does not occur in C, we obtain a formula φ' equivalent to φ in which all quantifiers occur only in subformulas of the form $\exists x \bigwedge_i L_i(x)$ and $\forall x \bigvee_i L_i(x)$, where all of the L_i's are literals in which x occurs. Using the standard skolemization arguments, we can replace each subformula $\forall x \bigvee_i L_i(x)$ by $\bigvee_i L_i(c)$, where c is a new constant. Finally, we can pull the remaining existential quantifiers out to obtain the required prenex existential formula. □

This lemma implies the following result.

Lemma 5. *Let φ be a formula of a monadic signature Σ. Using φ one can effectively find a prenex existential formula φ' of the same variables as φ such that*

1. *the signature of φ' extends the signature of φ by zero or more constants;*
2. *for every tuple of terms $\overline{t}$ of the signature Σ, $\overline{t}$ is a witness for φ if and only if it is a witness for φ'.*

3 Term Algebras

Let Δ be a finite signature with at least one constant. Denote by T_Δ the set of all ground terms of the signature Δ . The *term algebra* of the signature Δ is the structure with the universe T_Δ such that every ground term of Δ is interpreted in this structure by itself, for details see, e.g., [4]. We will denote the term algebra of Δ in the same way as its universe T_Δ.

The term algebra is infinite if and only if Δ contains at least one function symbol of positive arity. We are interested in monadic second-order theories of term algebras defined as follows.

Extend the first-order language by adding *second-order variables. Monadic second-order formulas* extend first-order formulas by adding (i) a new kind of atomic formulas of the form $Z(t)$, where Z is a second-order variable and t is a term, and (ii) *second-order quantifiers* $\forall Z$ and $\exists Z$, where Z is a second-order variable.

Let $\mathfrak{A}$ be a structure with a universe A. Over this structure second-order variables are interpreted as subsets of A and an atom $Z(t)$ is interpreted as membership of t in Z. *Monadic second-order theory* of $\mathfrak{A}$ is the set of all monadic second-order sentences which hold in $\mathfrak{A}$.

Using the decidability of the monadic second-order theory of two successors [8] it is not hard to prove the following result.

Lemma 6. *The monadic second-order theory of any term algebra A in a finite monadic signature is decidable.*

Proof. (sketch). We will show how one can interpret the monadic second-order theory of A in the monadic second-order theory of two successors.

Recall that the monadic second-order theory of two successors $S2S$ is the monadic second-order theory of the structure $T = (\{0;1\}^*, succ_0, succ_1)$ over the set of binary words with the successor functions $succ_0(w) = w0$ and $succ_1(w) = w1$. We will write $x0$ and $x1$ instead of $succ_0(x)$ and $succ_1(x)$ and $X \subseteq Y$ instead of $\forall w(X(w) \supset Y(w))$.

Suppose that the signature of A consists of constants $c_1, \ldots, c_n$ and function symbols $f_1, \ldots, f_k$. Then every ground term of the signature of A (or, equivalently, every element of A) has the form $f_{i_1}(\ldots f_{i_m}(c_j))$ for some $i_1, \ldots, i_m, j$. Let us call the *code* of such a term the word $1^j 0 1^{i_m} \ldots 0 1^{i_1}$. We will show that for every formula φ of the monadic second-order theory of A there exists a formula φ' of $S2S$ which expresses the same property but on the codes of elements of A.

Consider the following formulas of S2S

$$\text{contains\_codes}(X) := \bigwedge_{i=1\ldots n} X(1^i) \wedge \bigwedge_{j=1\ldots k} \forall w(X(w) \supset X(w01^j));$$

$$\text{all\_codes}(X) := \text{contains\_codes}(X) \wedge \forall Y(\text{contains\_codes}(Y) \supset X \subseteq Y);$$

$$\text{is\_code}(x) := \exists Y(\text{all\_codes}(Y) \wedge Y(x));$$
$$\text{codes\_only}(X) := \exists Y(\text{all\_codes}(Y) \wedge X \subseteq Y).$$

It is not hard to argue that $\text{contains}_{\text{c}}\text{odes}(X)$ expresses that X contains all codes, $\text{codes}(X)$ expresses that X is the set of all codes, $\text{is code}(x)$ expresses that x is a code, and $\text{codes\_only}(X)$ expresses that all elements of X are codes.

For every formula φ of the monadic second-order language of A consider the formula φ' of $S2S$ defined as follows. Without loss of generality we assume that the atoms in φ have the form $x = f_i(y)$, $x = c_j$, or $x = y$, where x, y are variables.

1. If φ is $x = f_i(y)$, then φ' is $x = y01^i$.
2. If φ is $x = c_j$, then φ' is $x = 1^j$.
3. If φ is $x = y$, then φ' is also $x = y$.
4. If φ is $\varphi_1 \wedge \varphi_2$, then φ' is $\varphi'_1 \wedge \varphi'_2$, and similar for other connectives in place of $\wedge$.
5. If φ is $\forall x\varphi_1$ (respectively $\exists x\varphi_1$), where x is a first-order variable, then φ' is $\forall x(\text{is\_code}(x) \supset \varphi'_1)$ (respectively $\exists x(\text{is\_code}(x) \wedge \varphi'_1)$).
6. If φ is $\forall X\varphi_1$ (respectively $\exists X\varphi_1$), where X is a second-order variable, then φ' is $\forall X(\text{codes\_only}(X) \supset \varphi'_1)$ (respectively $\exists X(\text{codes\_only}(X) \wedge \varphi'_1)$).

Let ν be a valuation over A, i.e., a mapping which maps second-order variables into subsets of A and first-order variables into elements of A. Denote by ν' the corresponding valuation over T, that is $\nu'(X)$ for second-order X maps X into the set of codes of terms in $\nu(X)$, and $\nu'(x)$ for first-order x maps x into the code of $\nu(x)$. By induction on φ it is not hard to argue that for every valuation ν over A we have $A, \nu \models \varphi$ if and only if $T, \nu' \models \varphi'$. In particular, when φ is a sentence we have $A \models \varphi$ if and only if $T \models \varphi'$. Since $S2S$ is decidable [8], the monadic second-order theory of A is decidable too. □

4 Proof of the Main Theorem

Lemma 7. *For any monadic signature Σ and its extension $\Delta \supseteq \Sigma$ by constants there exists a monadic second-order formula $TERM_\Sigma(x)$ of one variable x, such that for every ground term t of Δ the formula $TERM_\Sigma(t)$ holds in the weak second-order theory of term algebra of Δ if and only if t is a term of the signature Σ.*

Proof. Let U be the set of all nonconstant symbols of Σ (and hence also of Δ). Define the formula $TERM_\Sigma$ as follows.

$$TERM_\Sigma(x) = \exists Z \left(Z(x) \wedge \bigwedge_{f \in U} \forall y(Z(f(y)) \Rightarrow Z(y)) \wedge \bigwedge_{d \in \Delta - \Sigma} \neg Z(d) \right).$$

The proof is straightforward. □

Evidently, this lemma can be proved for arbitrary signatures Σ and Δ .

Now we can prove the main theorem.

Proof. (of Theorem 3). We prove that Formula Instantiation is decidable for the monadic class. Given a formula $\varphi(\overline{x})$ in a monadic signature Σ, we have to effectively decide whether it has a witness in the signature Σ. Let $P_1, \dots P_n$ be all predicate symbols of Σ and $x_1, \dots, x_m$ be all variables in $\overline{x}$. Applying Lemma 5 we obtain an existential formula $\varphi'(\overline{x})$ of a signature Δ such that (i) the signature of $\varphi'(\overline{x})$ extends the signature of $\varphi(\overline{x})$ by zero or more constants; (ii) for every tuple of terms $\overline{t}$ of the signature Σ , $\overline{t}$ is a witness for $\varphi(\overline{x})$ if and only if it is a witness for $\varphi'(\overline{x})$. Therefore, formula instantiation holds for $\varphi(\overline{x})$ if and only if $\varphi'(\overline{x})$ has a witness $\overline{t}$ in the signature Σ.

Since $\varphi'(\overline{t})$ is existential, by Herbrand's theorem it is valid if and only if it holds in all Herbrand interpretations of the signature Δ. But this property can be expressed using the following monadic second-order formula of T_Δ.

$$\forall P_1 \dots P_n \varphi'(\overline{t}).$$

where $P_1 \dots P_n$ are considered as second-order variables.

It follows that $\varphi'(\overline{x})$ has a witness in the signature Σ if and only if the following second-order formula holds in T_Δ:

$$\exists x_1 \dots \exists x_m \left(\bigwedge_{i=1\dots m} TERM_\Sigma(x_i) \wedge \forall P_1 \dots \forall P_n \varphi'(\overline{x}) \right).$$

The validity of this second-order formula can be effectively decided by Lemma 6. □

5 Conclusion

Our proof uses the decidability of the monadic second-order theory of term algebras in monadic signatures. The complexity of such theories is not elementary recursive [7]. At the same time satisfiability of formulas in the monadic class can be solved in nondeterministic exponential time [5]. So it remains an open problem whether the complexity of formula instantiation for the monadic class is elementary.

Another interesting open question is whether existing resolution-based decision procedures can be modified to find witnesses for formula instantiation.

References

1. A. Degtyarev and A. Voronkov. Equality reasoning in sequent-based calculi. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 10, pages 609-704. Elsevier Science, 2001.
2. J.H. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 338-346. IEEE Computer Society Press, 1987.
3. Yu. Gurevich. The decision problem for the logic of predicates and operations. 8:284-308, 1969.

4. W. Hodges. *Model theory*. Cambridge University Press, 1993.
5. H. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21:317-353, 1980.
6. M. Löb. Decidability of the monadic predicate calculus with unary function symbols. *Journal of Symbolic Logic*, 32:563, 1967.
7. A. Meyer. Weak monadic second order theory of successor is not elementary-recursive. In R. Parikh, editor, *Logic Colloquium: Symposium on Logic Held at Boston*, volume 453 of Lecture Notes in Mathematics, pages 132-154. Springer Verlag, 1975.
8. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141(1):1-35, 1969.
9. M. Veanes. The undecidability of simultaneous rigid E-unification with two variables. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory. 5th Kurt Gödel Colloquium, KGC'97*, volume 1289 of Lecture Notes in Computer Science, pages 305-318, Vienna, Austria, 1997.
10. A. Voronkov. Strategies in rigid-variable methods. In M.E. Pollack, editor, *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 1, pages 114-119, Nagoya, Japan, August 23-29 1997.
11. A. Voronkov. Herbrand's theorem, automated reasoning and semantic tableaux. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 252-263. IEEE Computer Society Press, 1998.
12. A. Voronkov. The ground-negative fragment of first-order logic is Π_2^p-complete. *Journal of Symbolic Logic*, 64(3):984-990, 1999.
13. A. Voronkov. Simultaneous rigid E-unification and other decision problems related to Herbrand's theorem. *Theoretical Computer Science*, 224:319-352, 1999.

From MKRP to ΩMEGA

Manfred Kerber

The University of Birmingham,
School of Computer Science,
Birmingham, B15 2TT, England
M.Kerber@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~mmk

Abstract. Around 1990 the work on the first-order theorem prover MKRP stopped after a development going on for more than a decade. Instead a new system has been developed since then, the mathematical assistant ΩMEGA. In this contribution I try to summarise some of the discussions and decisions that led to this shift in focus and to the development of the ΩMEGA system, and I attempt in retrospect to give a tentative evaluation of some of the decisions.

1 Introduction

In the late 1980's the MKRP-project came to an end, after a development of more than ten years in which a couple of millions of Deutschmark were spent (a Deutschmark is roughly half a Euro or half a US-Dollar). A new project on a new system started, the ΩMEGA system[1]. In the preceding discussions many decisions were taken and paradigms discussed. Now more than 10 years later it may be worthwhile firstly to document some of the discussions – also since the whole endeavour was a significant step in Jörg Siekmann's work in Mechanised Theorem Proving, and secondly to try a cautious first evaluation of some of the decisions.

The MKRP-project started in Karlsruhe when Jörg Siekmann was a research fellow at the University of Karlsruhe and was continued when he took up there a professorship for artificial intelligence at the University of Kaiserslautern. In this paper some details about MKRP and the MKRP-project are discussed, but only insofar as they are relevant for the transition from MKRP to ΩMEGA. It is not a description of the development of MKRP (neither of ΩMEGA), but a report on the transition period.

In order to understand the transition let's first take a look at the context in which the decisions were made, then take a closer look at the MKRP system itself and discuss shortcomings of the system in the intended applications and means to overcome these.

[1] Initially the new system was called Ω-MKRP, later only Ω or Omega and at a time the logo ΩMEGA was introduced.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 139–153, 2005.

2 The Context

Preceding the birth of ΩMEGA a major shift in the work of Jörg Siekmann and his group as well as their working conditions took place. Jörg was at that time an (associate) professor (C3) for artificial intelligence at the Computer Science department of the University of Kaiserslautern and headed a group of around ten active teaching and research assistants mainly financed by the German National Science Foundation (Deutsche Forschungsgemeinschaft, DFG), mainly, in the Sonderforschungsbereich 314 on Artificial Intelligence and European Union Esprit projects. Only the DFG financed projects were directly related to MKRP. In Germany, research assistants and teaching assistants typically work towards their PhD alongside their project work, only the professor has a permanent position. At the end of the 1980's most researchers in Jörg Siekmann's group were in the final stages of their PhD theses. Some had already left the group, some were about to leave, or looking for different activities which went beyond MKRP. To a large degree the work in the final phase of the MKRP-project and the potential extensions can be described by the work done by the people working on it. I will briefly mention some of these results of people working in the core of MKRP.

MKRP was built on the so-called clause graph procedure, originally proposed by Robert Kowalski [Kow75]. In this approach, a graph is used to store all possible resolution steps, detect redundancies and simplification possibilities. A difficult question is whether the procedure is actually complete and confluent, that is, whether if one starts with an unsatisfiable clause set, from any intermediate state it is still possible to derive the empty clause, and – assumed a fair strategy is employed – the empty clause will eventually be derived.

Norbert Eisinger [Eis91] studied the theoretical properties of clause graph procedure like completeness and confluence. Already during the Karlsruhe phase, Christoph Walther [Wal83] developed the order-sorted logic[2] on which MKRP is based. In Kaiserslautern, Manfred Schmidt-Schauß [SS89] worked on extensions of this logic with term declarations. Hans-Jürgen Bürckert [Bür90] worked on constraint resolution, Alexander Herold [Her87] on the combination of unification algorithms, and Christoph Lingenfelder [Lin90] on the presentation of resolution proof in natural deduction. Karl Hans Bläsius [Blä86] and Axel Präcklein [Prä92] worked on different approaches to equality reasoning from human-oriented to traditional rewrite oriented ones. Hans Jürgen Ohlbach (but also many others) were very involved in the MKRP project or related projects. They played a crucial rôle in the development of MKRP and of extensions to MKRP. Hans Jürgen in the end chose a PhD topic on modal logic rather than MKRP-related issues [Ohl88].

While all these people finished their PhDs, two things crucial for the further development happened around the same time, firstly the German Research Centre for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) was founded in Saarbrücken and Kaiserslautern, and secondly Jörg Siekmann was offered and accepted a full professorship (C4) in Saarbrücken, jointly with the post of a director at the DFKI. This meant not only that a shift

[2] The idea of sorts is to replace particular unary predicate symbols by sort symbols. This has the advantage to shorten clauses and prevent meaningless unifications.

in the focus of his work was necessary, but also that senior people were needed to head the new research areas in the DFKI.

Initially, the ΩMEGA-group (in addition to Jörg Siekmann himself) consisted of Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, and Jörn Richts. While Xiaorong and Manfred had worked in the group for a couple of years already, Xiaorong on the verbalisation of natural deduction proofs, and Manfred on human-oriented theorem proving, the others recently joined. Michael had just met Jörg Siekmann at a seminar of the prestigious Studienstiftung. His special interest was to develop a logic close to the mathematical representation found in mathematical practice, concretely this meant to develop a sorted higher-order logic and its mechanisation. Erica had worked on analogy and related questions in a general context at the Humboldt University in Berlin; she joined the group after she met Manfred just before the fall of the wall at a conference on analogy in East Germany. Dan came – a short while after the others – from Peter Andrews' group to Saarbrücken. He brought in his tremendous experience in the implementation of the TPS system. This way TPS strongly influenced the style of the logical core of ΩMEGA. Dan was to become the main developer of the Keim implementation system, which has provided the implementational base for ΩMEGA [HKK+94]. Jörn had just finished his master thesis on MKRP and had a general interest in the new paradigm of ΩMEGA.

3 Paradigms of the MKRP System

As mentioned MKRP [MGR84,EO86,OS91,Prä92a] is a traditional theorem prover based on resolution and paramodulation for sorted first-order logic. In practice, the MKRP-system is a traditional first-order theorem prover; in spirit, however, Jörg Siekmann had from the very start of the project in mind to build a theorem prover following a human-oriented approach of reasoning. The original idea was to have a so-called logic engine, that is, the graph-based resolution engine, which is guided by a so-called "supervisor," that is, a module which tells the logic engine what to do next (see Fig. 1). Jörg Siekmann's idea was that the logic engine should be supplemented by a human-oriented component that guides the logic engine. However, more and more effort was put into the logic engine itself and the "supervisor" was never realised nor was there a serious attempt to design it. Many people started to work on it, but switched to work on the core system since there was concrete work to be done, which also was easier to sell to a community that is up to now not very interested in human-oriented theorem proving. At a time Axel Präcklein built an interactive component as an extension to MKRP which visualised all possible resolution steps and allowed to run MKRP in an interactive mode. Using this component, Axel Präcklein and Manfred Kerber tried to find patterns in the clause graph which allow to select the next resolution possibility heuristically. However, it turned out that it was already very difficult to find proofs at all this way and virtually impossible to detect any usable structure. Usually MKRP performed these steps much better than humans. From this it was concluded that automating a human-oriented selection module for resolution steps would be very difficult (it wouldn't have been very human-oriented anyway).

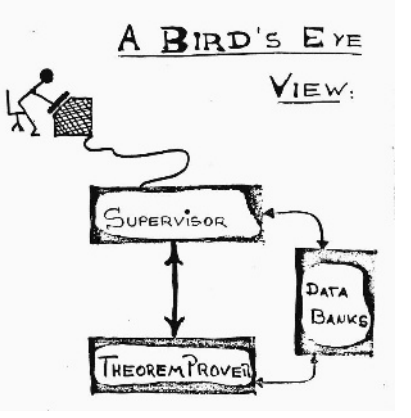


Fig. 1. A bird's eye view for the intended interplay between MKRP and the control component the "supervisor", taken from Jörg Siekmann's lecture notes.

The motivation force in the MKRP project was to prove a mathematical textbook fully using MKRP. The only ever fully proved mathematical textbook is Edmund Landau's "Grundlagen der Analysis" [Lan30]. This book was interactively proved in the Automath system see [NGdV94] by L.S. van Benthem-Jutting [Jut79]. It was a major attempt to prove a full mathematical textbook and it took van Benthem-Jutting five years to do it, although Landau's book is very formalised with 301 theorems on 134 pages. The effort needed indicates that Automath seems not to have been the right tool to prove textbooks. While it is a proof checker, MKRP is an automated theorem prover. The hope was that this would make it much easier to prove a book with MKRP. The textbook chosen was Peter Deussen's "Halbgruppen und Automaten" (Semi-groups and Automata) [Deu71]. The first 5 of 17 sections were actually proved with MKRP. As will be seen in the following, there were, however, severe problems with using MKRP for this task. When we formed plans for a new project, ΩMEGA, we naturally wanted to take into account all the lessons learned from the attempt to prove the text book. In the following the most important insights will be summarised which strongly influenced the further development (see also [Ker92] and [HKK+92]).

Logic. The representation of the mathematical concepts in the sorted first-order input language of MKRP is often clumsy and unnatural, also the representation was often *ad hoc*. The concepts and constructs of a typical mathematics textbook are quite rich and much better approximated by a higher-order language, we were forced to use sophisticated encoding techniques to translate them manually into the MKRP first-order input language. While the availability of sorts and the built-in equality predicate allow for a tolerably adequate translation, it is not always obvious what the theorems proved by MKRP have to do with the textbook theorems and hence what is actually proven. As a minimal requirement one would want an automatic translation technique from higher-order to first-order logic to support a user in the encoding task.

Knowledge Base. The MKRP-system, as most other automated theorem provers, has no integrated *mathematical* knowledge. Each time definitions and lemmas, which are used as preconditions for the actual theorem, must be coded and re-input. This is not only rather boring, but is also a serious source of error. Often (slightly) different formulations are chosen in different contexts, with the consequence that the correctness of the whole procedure of machine verification of textbooks is no longer assured. Moreover, the user may insert lemmas that cannot be proven in the given context. Discipline may be helpful, but as practice shows, automated assistance is indispensable. In short, a system that supports human mathematicians in proving theorems must include a database of mathematical knowledge that can be accessed and updated in a controlled way. This in itself is a major research task, still not fully solved.

Structuring in Subproblems. More often than not, real mathematical theorems are too hard to be proven automatically. This state of affairs can be ameliorated by strengthening the deductive power of the prover in various ways (and since then considerable progress has been made in the field, of course). For every system, however, there exist theorems that cannot be shown automatically. In order to be useful, the user must be given the opportunity to guide the proof process interactively. In a classical theorem-proving system this is almost impossible: the cycle of interaction consists of a complete restart with a different setting of the parameters or a reformulation of the clauses. The main influence the user has, consists in the appropriate choice and formulation of the problem. The way the preconditions of a theorem are selected, for instance, is of paramount importance for the performance of the system. An additional necessary facility is one for splitting the problem manually into subproblems, so that they can be proved separately and then used as lemmas later in the proof of the original theorem. Traditional theorem provers lack such support and the situation is far from satisfactory, as all structuring decisions and all proof plans are hand-crafted. In short, all of this requires too much care and skill from the user, and not surprisingly there are fewer than a handful of well-known experts who are renowned for their skill in proving difficult theorems with the help of a machine. Since it is always possible to break down difficult theorems into digestible subproblems, there is also the question to which degree this procedure can be called automatic.

All these points showed us that we needed a *new* system which should be developed from scratch and that there was no point in attempting to extend the existing MKRP system. However, we were also very reluctant to throw away more than ten years of development work and dozens of person years' work. This led to the idea to build an open system, in which it is possible to add external systems to solve subproblems. The first external system would be MKRP, others could then easily follow.

4 Discussions and Decisions

The principles for the ΩMEGA system were developed in many meetings, seminars and discussions, a picture of one of the blackboard summaries can be found in Fig. 2. These discussions were influenced by many people, who either came to the group to give presentations or with whom we were in intensive discussions. To mention just a few: Jörg Denzinger (with whom we had intensive discussions on proof planning), Christoph Kreitz (who explained to us the details of Nuprl), Wolfgang Reif (on the reuse of proof attempts), William Farmer (on the IMPS system). In addition, we had numerous discussions with Dieter Hutter, Claus Sengler, Jürgen Cleve and others, who just arrived in Saarbrücken at the same time as we, to work on Jörg Siekmann's verification project. Furthermore we read many articles outside the main paradigm of the MKRP system, such as papers on Automath [Bru80,NGdV94], Isabelle [Pau90], and proof planning [Bun88].

These discussions were about almost all aspects of a system, from the high-level philosophy of the system to the implementation language and the data structures employed. In the remainder of this section I want to summarise some key questions and decisions. Let us first take a look at the philosophy of the system. In this we took up the experience gathered in the work of proving the first third of [Deu71].

4.1 Automated Versus Interactive

Certainly a mathematician wants to guide the process of proving mathematical theorems. A system that puts you before an "all or nothing" approach seems highly unsatisfactory. Ideally you want a system that is as automatic as possible in order to support a user, but leave as much room for interaction as possible at the same time.

Fig. 2. A Blackboard Summary of Discussions in 1991 written by Jörg Siekmann.

A quote from [HKK+94a]:

On account of this, we believe that significantly more support for proof development can be provided by a system with the following two features:

- *The system must provide a comfortable human-oriented problem-solving environment. In particular, a human user should be able to specify the problem to be solved in a natural way and communicate on proof search strategies with the system at an appropriate level.*
- *Such a system is interesting only if it relieves the user of non-trivial reasoning tasks and provides the foundation for a practicable increased reasoning power. We are convinced that this requires not only task-specific tactics but also the strong reasoning power of a general logic engine.*

As a consequence we decided to build a system that on the one hand allows for tactical theorem proving, which allows to influence the proof search by calling tactics, and on the other hand has integrated strong external components like automated theorem provers which can be called as black boxes to solve subproblems (see Fig. 3). In summary we tried to find a viable compromise between automatic and interactive theorem proving. In retrospect, the decision to follow an automatic and interactive approach at the same time seems to have been the right one. The full strength of this synthesis seems to be coming to the fore only recently in work on agent-oriented theorem proving as developed by Christoph Benzmüller and Volker Sorge [BS01] and multi initiative proof planning by Andreas Meier and Erica Melis [MM00].

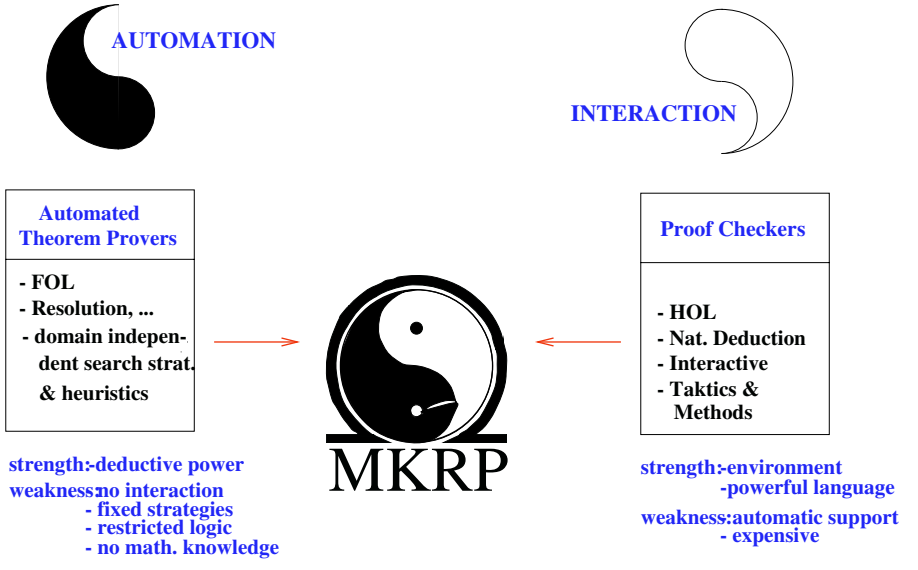


Fig. 3. Motivation Diagram for the ΩMEGA system.

4.2 Paradigm

Related to (but independent from) the question whether to follow an automated or an interactive theorem proving style, was the question what the type of the system should be, should it follow traditional machine-oriented theorem proving or human-oriented theorem proving.

Here again, we followed the approach that we wanted to include different paradigms. We wanted to include theorem provers like MKRP and Otter, but wanted also to take the original idea of a human-oriented approach very seriously (see Fig. 3). Some work on analogical theorem proving [Ker89] was done at that time. Around the same time we understood the significance of Alan Bundy's approach to use planning techniques in theorem proving in the form of proof planning [Bun88]. It was at the same time when the last proposal to reimplement MKRP and to bring it up to the most recent developments in resolution style theorem proving was discussed. We decided against this and started concrete work on the ΩMEGA system instead. We decided to make use of fast existing first-order theorem provers like Otter rather than to bring MKRP to speed.

The approach of proof planning looked very attractive since it allows – unlike MKRP – to include domain-dependent reasoning knowledge in form of proof planning methods [MS99]. Around this time we also adopted the idea that for automation incomplete approaches can be stronger than complete ones.

The overall architecture of ΩMEGA as discussed then and realised later can be seen in Fig. 4. Centred around the structure of Natural Deduction proofs, different components and the user can manipulate partial proofs to complete a proof, this can be done by inserting information from a knowledge base, by proof planning, by external components, or proof transformation.

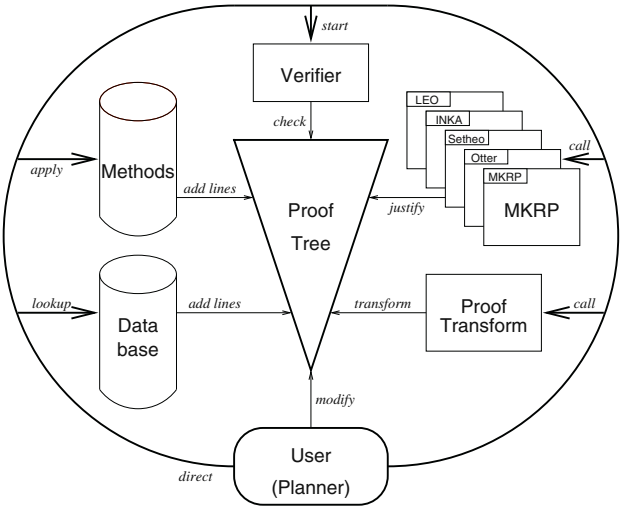


Fig. 4. Architecture of ΩMEGA.

4.3 The Logic Language – First-Order Versus Higher-Order

MKRP is a first-order theorem prover. For all theorems we tried to prove we found a representation in first-order logic. Also there has been work by Robert Boyer et al. [BLM+86] (later taken up by Art Quaife [Qua92]), which shows how first-order logic can be effectively used to prove large chunks of mathematics in some form of set theory. This is also the approach taken in Mizar [Rud92]. The advantage is that a few axioms suffice to build up all of the relevant mathematics. The disadvantage is that the approach seems to be far away from mathematical practice and a human-oriented approach. When you want to represent a function in mathematics, a function symbol in logic seems to be much closer to the informal notion than a left total and right unique relation.

The argument was strongly supported by the plan to build on existing sort systems (like that used in MKRP) to extend it to higher-order logic, so that it would be possible to speak about unary functions on real numbers, continuous, and differentiable ones and so on. Here two different aspects had to be considered. Firstly we wanted to use the language as the representation language, that is, we wanted it to be as strong as possible (including dependent sorts, to be able to encode structures like groups very naturally as a set $G : set$ with an operation $+ : G \times G \to G$, where the operation depends on G). Secondly Michael Kohlhase wanted to (and later did in collaboration with Christoph Benzmüller) build a theorem prover based on higher-order logic with sorts which should form the Logic Engine for Omega, Leo [Koh94,BK98]. Since this is a very difficult task, dependent sorts (as well as partiality as discussed in [KK96]) were not included for the time being.

4.4 Explicit Proofs, Proof Presentation, and Interface

The importance of an adequate presentation of proofs has always played a significant rôle in the MKRP as well as the ΩMEGA project. This started by the work of Christoph Lingenfelder [Lin89,Lin90,LP91] and Xiaorong Huang [Hua96] and has been continued by Armin Fiedler since. From the work to present proofs to humans it was a short step to arrive at the philosophical position that proofs must be checkable [HKK+94c]. This solved a major problem of the structure of the ΩMEGA system, namely, how is it possible to ensure correctness of proofs in the presence of many – quite heterogeneous black box components – which may contribute to a proof.

The ΩMEGA system went first with an Emacs interface only. In the initial discussions we discussed already the importance of a high-level user interface, but in the implementation we had to focus on the kernel of the system. Only later LOUI [SHB+99] was developed to provide a graphical user interface to ΩMEGA.

4.5 Knowledge Representation

The representation of mathematical knowledge was considered of great importance from the very beginning [SK93]. It was considered as a central question and

attempted even in days when there were still discussions to build a so-called "supervisor" for MKRP. In discussions between Norbert Eisinger, Jörg Siekmann, and Manfred Kerber it was decided to start the work on a knowledge based reasoning system with the representation of knowledge. It has been considered important to represent mathematics not only as mathematical formulae, but also to attribute to a formula a semantic status, whether it is an assumption, a precondition, a definition, a conjecture, or a theorem. Furthermore a context should be provided, problems can build up on each other, which makes inheritance of concepts possible. A particular interest was also put in re-representation issues [KP96], however, the question is still mainly unanswered.

4.6 External Systems

As external system to include we started with MKRP, also to justify that all the work that went into MKRP was not in vain. This led then to the next step to include other first-order theorem provers, which are much faster than MKRP, like Otter [McC90]. While this approach was criticised by reviewers, since we didn't follow a seamless approach of user-oriented theorem proving which employs human-oriented theorem proving like proof planning throughout, it made it possible to further extend the system later on. For instance, Leo [BK98] was anticipated as a loosely coupled and not tightly integrated system. Computer algebra systems followed [KKS98] as well as constraint solvers [MZM00]. This diversified the type of systems included. A generalisation of such a flexible integration was never anticipated in the early days, but the high flexibility needed can be viewed in retrospect as a seed which helped to generate the ideas of Mathweb [FK99] and OMDoc [Koh01] by Michael Kohlhase and others.

4.7 Application

As intended application for the ΩMEGA system we took over the old project of MKRP to prove a mathematical textbook without any discussion. In hindsight this is too narrow a view what such a system can be used for. Although we never explicitly excluded other potential applications, it might have been better to positively keep other potential applications – like the recently emerging applications in education (see Erica Melis' contribution in this volume) – in sight at an early stage already.

4.8 Programming Language

Different programming languages were discussed when we started the ΩMEGA project, to mention some, C (for efficiency), Prolog (for rapid prototyping), Lisp (as a compromise, which stood in the tradition of the MKRP project), and ML (viewed as a typed variant of Lisp). The decision against C or C++ was in retrospect good, since our ideas were not clear enough when we started with ΩMEGA that we could exactly specify it, Prolog may not have been flexible

enough, ML with its type system might have been the better choice and some people favoured it when we started, but in the end tradition prevailed and we decided for Lisp, because of the knowledge, the hardware, and the software available. Lisp proved to be a powerful, flexible language which allowed for many developments in the sequel, functional, object-oriented and concurrent system development. It was a good choice (although ML might have been a better).

ΩMEGA was not directly implemented in Lisp (Common-Lisp to be more precise), but in the Keim environment, which was a programming environment built on top of Lisp (in the German Focal Programme on Deduction, financed by the Deutsche Forschungsgemeinschaft). Its philosophy is (quote from [HKK+94]):

> *..., those who wish to apply techniques developed by the theorem-proving community face the choice of either learning this 'black art' themselves by developing their own prover from scratch, or jury-rigging available provers to get some kind of result.*

While Keim greatly facilitated the development of ΩMEGA, it remained still a 'black art' to build and extend ΩMEGA. Keim turned out to be a complex system, which is difficult to handle. Furthermore there are performance problems. In retrospect I think these are due to the attempt to build with Keim a system which is very general and construed to enable the implementation of a wide range of deduction systems. The question whether to build a system as general as possible versus to build a system as simple as possible was answered in favour of generality. This seems to me now to have been a mistake.

5 Conclusion

Most decisions turned out to be fruitful although they might not all have been optimal. As already mentioned an earlier focus on a different application domain and a different approach to the programming might have been beneficial. But by far not all developments were foreseen. In some aspects we were too optimistic what could be achieved in ten years – for instance, we thought that it would be much easier to formalise standard mathematics in sorted higher-order logic. In other aspects we didn't dare to hope for a state like the one achieved today – for instance, we didn't hope that the work would be applicable now already in education.

There are developments we did not anticipate, but which were made possible by ΩMEGA. I want to mention agent-oriented theorem proving [BS01], knowledge-based proof planning [MS99], Mathweb [FK99], and ΩDOC [Koh01]. In my view this shows that ΩMEGA has been a flourishing project.

ΩMEGA meant a change of direction in Jörg Siekmann's theorem proving group, a change in area, in approach, and in paradigm. This was of course very risky, since it meant a reduced possibility for publications for a significant amount of time and the potential knock-on effect on funding. Fortunately referees in the German funding organisations were very positive and supportive (I just want to mention Wolfgang Bibel and Michael M. Richter here). I think that ΩMEGA has been a scientific success and a worthwhile enterprise. Jörg Siekmann was already

an established scientist who could have rested on his laurels and continued with what he always did, when ΩMEGA started. He, however, actively initiated this major change in direction, which considerably deviated from the path originally envisaged in the MKRP project, when he was convinced that only in this way we would come closer to making the old dream true to automate mathematics. Leibniz had a dream, "Calculemus", namely to mechanise mathematics (actually he wanted to mechanise not only mathematics but all of human thought, we never were so ambitious in the ΩMEGA project). There are many problems in detail still to be solved to provide a powerful useful tool for mechanised mathematics. In the past there have been too many promises/predictions about what will be achieved in the near future. I don't want to add another one, but although we have not yet realised the dream, I feel that we are significantly closer to its realisation.

This paper is not a survey on the ΩMEGA system, but just on the discussions during the transition from MKRP to ΩMEGA. Many exciting developments have taken place since then and without doubt will take place in the future. They have to be reported somewhere else.

Acknowledgements

The transition from MKRP to ΩMEGA was possible since several things came together. Not only the researchers involved were filled with great enthusiasm, but also many undergraduate students worked with great enthusiasm on the project.

I feel that the description which I tried to provide here is more incoherent and subjective than I would like it to be; this is entirely my fault. That it is not even more incoherent is due to many helpful comments by Norbert Eisinger, Michael Kohlhase, and Erica Melis on a earlier draft. I would like to express my thanks to them hereby.

References

[BK98] Christoph Benzmüller and Michael Kohlhase. Leo – a higher-order theorem prover. In Claude Kirchner and Hélène Kirchner, eds., *Proceedings of the 15th CADE*, p. 81–97, Lindau, Germany, 1998. Springer, LNAI 1421.

[Blä86] Karl Hans Bläsius. *Equality Reasoning Based on Graphs.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1986.

[BLM+86] Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, **2**:287–327, 1986.

[Bru80] Nicolaas Govert de Bruijn. A survey of the project Automath. In J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism*, p. 579–606. Academic Press, London, UK, 1980.

[BS01] Christoph Benzmüller and Volker Sorge. Oants – an open approach at combining interactive and automated theorem proving. In Manfred Kerber and Michael Kohlhase, eds., *Symbolic Calculation and Automated Reasoning: The CALCULEMUS-2000 Symposium*, p. 81–97, St. Andrews, Scotland, 2001. A.K. Peters, USA.

[Bun88] Alan Bundy. The use of explicit plans to guide inductive proofs. In Ewing Lusk and Ross Overbeek, eds., *Proc. of the 9th CADE*, p. 111–120, Argonne, Illinois, USA, 1988. Springer, LNCS 310.

[Bür90] Hans-Jürgen Bürckert. *Constraint Resolution – A Resolution Principle for Clauses with Restricted Quantifiers*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[Deu71] Peter Deussen. *Halbgruppen und Automaten*, Volume 99 of *Heidelberger Taschenbücher*. Springer, 1971.

[Eis91] Norbert Eisinger. *Completeness, Confluence, and Related Properties of Clause Graph Resolution*. Pitman, London, UK, 1991.

[EO86] Norbert Eisinger and Hans Jürgen Ohlbach. The Markgraf Karl Refutation Procedure (MKRP). In Jörg H. Siekmann, ed., *Proc. of the 8th CADE*, p. 681–682, Oxford, UK, 1986. Springer.

[FK99] Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, ed., *Automated Deduction – CADE-16*, LNAI 1632, p. 217–221. Springer, 1999.

[Her87] Alexander Herold. *Combination of Unification Algorithms in Equational Theories*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1987.

[HKK+92] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Omega-MKRP – a proof development environment. Seki Report SR-92-22, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[HKK+94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. KEIM: A toolkit for automated deduction. In Alan Bundy, ed., *Automated Deduction – CADE-12*, Proceedings of the 12th International Conference on Automated Deduction, p. 807–810, Nancy, France, 1994. Springer. LNAI 814.

[HKK+94a] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP: A proof development environment. In Alan Bundy, ed., *Automated Deduction – CADE-12*, Proceedings of the 12th International Conference on Automated Deduction, p. 788–792, Nancy, France, 1994. Springer. LNAI 814.

[HKK+94c] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Dan Nesmith, and Jörn Richts. Guaranteeing correctness through the communication of checkable proofs (or: Would you really trust an automated reasoning system?). In David Basin, Fausto Giunchiglia, and Matt Kaufmann, eds., *Proceedings of the CADE-Workshop on Metatheoretic Extensibility of Automated Reasoning Systems*, p. 31–33, Nancy, France, 1994.

[Hua96] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. infix, St. Augustin, Germany, 1996.

[Jut79] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the* Automath *System*, Volume 83 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, The Netherlands, 1979.

[Ker89] Manfred Kerber. Some aspects of analogy in mathematical reasoning. In Klaus P. Jantke, ed., *Analogical and Inductive Inference; International Workshop AII '89*, p. 231–242, Reinhardsbrunn Castle, GDR, October 1989. Springer. LNAI 397.

[Ker92] Manfred Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.

[KK96] Manfred Kerber and Michael Kohlhase. A tableau calculus for partial functions. *Collegium Logicum – Annals of the Kurt-Gödel-Society*, **2**:21–49, 1996.

[KKS98] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[Koh94] Michael Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.

[Koh01] Michael Kohlhase. OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In Eugenio Roanes Lozano, ed., *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2000*, LNAI 1930. Springer, 2001.

[Kow75] Robert Kowalski. A proof procedure using connection graphs. *JACM*, **22**, 1975.

[KP96] Manfred Kerber and Axel Präcklein. Using tactics to reformulate formulae for resolution theorem proving. *Annals of Mathematics and Artificial Intelligence*, 18(2-4):221–241, 1996.

[Lan30] Edmund Landau. *Grundlagen der Analysis*. Akademische Verlagsgesellschaft, Leipzig, Germany, chelsea publishing, 1948 Edition, 1930.

[Lin89] Christoph Lingenfelder. Structuring computer generated proofs. In N.S. Sridharan, ed., *Proc. of the 11th IJCAI*, p. 378–383, Detroit, Michigan, USA, 1989. Morgan Kaufmann, San Mateo, California, USA.

[Lin90] Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[LP91] Christoph Lingenfelder and Axel Präcklein. Proof transformation with built-in equality predicate. In John Mylopoulos and Ray Reiter, eds., *Proc. of the 12th IJCAI*, p. 165–170, Sydney, 1991. Morgan Kaufmann, San Mateo, California, USA.

[McC90] William McCune. Otter 2.0. In Mark E. Stickel, ed., *Proc. of the 10th CADE*, p. 663–664, Kaiserslautern, Germany, 1990. Springer, LNAI 449.

[MGR84] Karl Mark G Raph. The Markgraf Karl Refutation Procedure. Technical Report Memo-SEKI-MK-84-01, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1984.

[MM00] Erica Melis and Andreas Meier. Proof planning with multiple strategies. In John Lloyd et al., ed., *Proc. of First International Conference on Computational Logic – CL 2000*, p. 644–659, Berlin, Germany, 2000. Springer, LNAI 1861.

[MS99] Erica Melis and Jörg H. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, **115**(1):64–105, 1999.

[MZM00] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Ch. Ringeissen, ed., *Frontiers of Combining Systems, Third International Workshop, FroCoS'2000*, p. 32–46. Springer, LNAI 1794, 2000.

[NGdV94] Rob Nederpelt, Herman Geuvers, and Roel de Vrijer, eds. *Selected Papers on Automath*, Volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, The Netherlands, 1994.

[Ohl88] Hans Jürgen Ohlbach. *A Resolution Calculus for Modal Logics*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1988.

[OS91] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl Refutation Procedure. In Jean-Louis Lassez and Gordon Plotkin, eds., *Computational Logic – Essays in Honor of Alan Robinson*, Chapter 2, p. 41–112. MIT Press, Cambridge, Massachusetts, 1991.

[Pau90] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, p. 361–386, 1990.

[Prä92] Axel Präcklein. *Integration of Rewriting, Narrowing, Compilation, and Heuristics for Equality Reasoning in Resolution-Based Theorem Proving*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[Prä92a] Axel Präcklein, ed. The MKRP User-Manual. SEKI Working Paper SWP-92-03, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[Qua92] Art Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, **8**(1):91–146, 1992.

[Rud92] Piotr Rudnicki. An overview of the Mizar project. Bastaad, Sweden, 1992.

[SHB+99] Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fielder, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. Loui: Lovely omega user interface. *Formal Aspects of Computing*, **11**:326–342, 1999.

[SK93] Barbara Schütt and Manfred Kerber. A mathematical knowledge base for proving theorems in semigroup and automata theory – Part I. SEKI Working Paper SR-93-02, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1993.

[SS89] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Springer, LNAI 395, 1989.

[Wal83] Christoph Walther. A many–sorted calculus based on resolution and paramodulation. In *Proc. of the 8th IJCAI*, p. 882–891, Karlsruhe, Germany, 1983. Morgan Kaufmann, San Mateo, California, USA.

Decidable Variants of Higher-Order Unification

Manfred Schmidt-Schauß

Johann Wolfgang Goethe-Universität,
Fachbereich Biologie und Informatik,
Institut für Informatik,
Postfach 11 19 32, D-60054 Frankfurt, Germany
Tel: (+49)69-798-28597, Fax: (+49)69-798-28919
schauss@ki.informatik.uni-frankfurt.de
www.ki.informatik.uni-frankfurt.de/

Abstract. Though higher-order unification is in general undecidable, there are expressive and decidable variants. Several interesting special cases and variants stem from restricting algorithms to search only unifiers where the number of bound variables is restricted. The intention of this paper is to summarize results in this area and to shed some light on the connections between context unification, decidable variants of higher-order, second order unification and string unification. Since this paper is intended to appear in a volume in celebrating Jörg Siekmann's 60th birthday, we will take the opportunity to give hints on how the motivating power of Jörg Siekmann has contributed to and put forward the research in unification.

1 Introduction

Unification is solving equations and computing their solutions. There are applications in several areas of Computer Science: Automated Deduction, Computational Logic, Logic Programming, Functional Programming, Type Checking, and Program Specification.

First-order unification [Rob65, Sie84, BS94, BN98] is a fundamental operation in several areas of computer science. The generalization to higher-order unification increases its expressiveness, its applicability and improves the level of abstraction. This explains the interest in higher-order extensions such as higher-order logics and higher-order deduction systems [And86, Pau94, GLM97, And01, Pfe01], higher-order (functional) programming languages [BMS80, Tur85, Pau91, Bar90, Bir98], higher-order logic programming languages [Mil91, HKMN95], higher-order rewriting [Nip91, Klo92, DJ90] and higher-order unification [Hue75, Dow01].

Jörg Siekmann's contributions to the field of unification start early after Robinson's paper on resolution and first order unification [Rob65]. Influenced also by work of Plotkin [Plo72] he worked on string unification with the intention to find efficient unification algorithms and to explore the general properties of sets of unifiers [Sie75]. Makanin [Mak77] settled the question, and gave an

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 154–168, 2005.

algorithm, however, looking at the algorithm it turned out that this was one of the most complex algorithms described so far for a problem in computer science.

Jörg Siekmann was also a supervisor of Peter Szabó's thesis on unification [Sza82], which treated unification and matching under equational theories. Here the open question of decidability of unification in a theory of only two distributive axioms first appeared. The driving and motivating force of Jörg shows up in several overview papers, one of the first ones was [Sie84], a more recent one was written together with Franz Baader, a former researcher in his group [BS94].

It is an interesting co-occurrence that the equational theory of associativity as well as the theory of two-sided distributivity have a connection to second-order unification, in particular to context unification. String unification is a specialization of context unification and the decidability of unification under two distributive axioms was shown using decidability of a fragment of context unification. Jörg Siekmann was also the supervisor of my thesis, and his motivating power has been influencing me since.

2 First Order Unification

First order unification is solving equations of terms over a signature Σ of function symbols and over an infinite set of variables. Every function symbol f in the signature comes with a fixed arity $ar(f)$. The terms without variables (ground terms) are defined using the grammar

$$t ::= f(t_1, \ldots, t_{ar(f)})$$

Equations are defined between terms with variables, defined by the grammar

$$t ::= x \mid f(t_1, \ldots, t_{ar(f)})$$

An equation is written as $s \doteq t$. Solutions, called *unifiers*, of a set of equations $\{s_i \doteq t_i \mid i = 1, \ldots, n\}$ are substitutions for the variables that solve the equations.

The algorithm in [Rob65] is exponential. The algorithm described in [MM79] is polynomial. It is well-known that there is a linear algorithm for deciding unifiability of first order unification problems [PW78] (for further information see the overview article [BS94]). In the case of first order unification it is possible to efficiently compute a single most general unifier that represents all unifiers of a given problem.

3 String Unification

First order unification is generalized into different directions. In particular, there has been research into unifying equations given a set of equational axioms. We confine ourselves to the equational theory of associativity. We will see that there is a strong relationship between context unification, which is a special case of higher-order unification, and associative unification. Associative unification is based on a signature with one binary symbol f and infinitely many constants, the axiom being

$$f(x, f(y, z)) = f(f(x, y), z)$$

If there are also the axioms $f(x, \varepsilon) = x$, and $f(\varepsilon, x) = x$, where ε is a constant meaning the empty string, then it is also called *string unification*.

The problem of (un-)decidability of solving associative term equations was open for over a decade. There was some work on unification procedures that compute complete and/or minimal sets of unifiers [Plo72, Sie75]. The problem of decidability was then solved by Makanin [Mak77] in a seminal paper giving a description of a rather complex algorithm. An important lemma used by Makanin states an upper bound on the so-called exponent of periodicity of a minimal unifier. The exponent of periodicity of a unifier is the maximal number n, such that the repetition w^n for a nontrivial word w is a subterm of some term t_i in the unifier $\{x_i \mapsto t_i \mid i = 1, \ldots m\}$. This bound was improved to an almost optimum by [KP96], where it is shown that the number is at most exponential, i.e., $O(2^{c_1 * n + c_2})$, and hence the representational size of the number is linear.

The original algorithm of Makanin was of non-elementary complexity. Recently, there was a series of improvements of decision algorithms: NEXPTIME [Gut98], and PSPACE [Pla99]. The last result was an unexpected and striking result and could only be obtained by inventing a new algorithm, but also using the optimal bound on the exponent of periodicity. Nevertheless, there remains a complexity gap, since the best known lower bound is that string unification is $\mathcal{NP}$-hard.

There are associative unification problems with an infinite number of (incomparable) most general unifiers, for example $f(x, a) \doteq f(a, x)$.

4 Higher-Order Unification

We give a short description of the foundations.

There is a grammar for types

$$T ::= T_0 \mid (T \to T)$$

where $T_0 \neq \emptyset$ is the set of *elementary types*. The symbols α, τ range over types, and ι ranges over elementary types.

A shorter notation for types of the form $\tau = (\alpha_1 \to (\alpha_2 \ldots (\alpha_n \to \iota) \ldots))$ is $(\alpha_1 \to \alpha_2 \to \ldots \to \alpha_n \to \iota)$. The number n is called the *arity* of the type τ, denoted $ar(\tau)$, and ι is called the *target type* of τ.

There is a signature Σ of *function symbols*, where every function symbol f has a type $type(f)$ and an arity $ar(f) := ar(type(f))$.

Function symbols f of elementary type (i.e., $ar(f) = 0$) are called *constant symbols*. We assume that Σ contains for every type τ a countably infinite set of function symbols. For every type τ there are infinitely many variables V_τ. Variables are denoted as x, y, z. As for function symbols, with $type(x)$ we denote the type of x. The arity $ar(x)$ of x is $ar(x) := ar(type(x))$. A variable of elementary type is also called *first-order variable*. The order of a type is defined as follows: $ord(\iota) = 1$, for an elementary type ι, and $ord(\tau_1 \to \ldots \to \tau_n \to \iota) = max\{ord(\tau_i) \mid i = 1, \ldots, n\}$.

Definition 4.1. *For every type τ we define the set $Term^\tau$ of terms of type τ:*

$$Term^\tau ::= f^\tau \mid x^\tau \mid (Term^{\tau' \to \tau}\ Term^{\tau'}) \mid \lambda x^{\tau_1}.Term^{\tau_2}$$

where f is a function symbol of type τ, and x is a variable of type τ. The term $\lambda x^{\tau_1}.Term^{\tau_2}$ is only valid if $\tau_1 \to \tau_2 = \tau$.

The *head* of an application is the subterm that is in the leftmost position in the flat representation. For example, f is the head of $(f\ t_1\ \ldots\ t_n)$.

4.1 β and η-Reduction and Equality

The $\beta\eta$-rules for the simply typed lambda-calculus impose the following equations between terms:

(α)	$\lambda x.t$	$= \lambda y.t[y/x]$	y is a fresh variable
(β)	$((\lambda x.t)\ s)$	$= t[s/x]$	(the capture-free substitution)
(η)	t	$= \lambda x^\tau.(t\ x)$	if $type(t) = \tau \to \tau_1$ and $x \notin \mathcal{FV}(t)$.

Of course we also assume that the equality $=_{\beta\eta}$ defined by $(\alpha), (\beta), (\eta)$ is an equivalence relation and a congruence, i.e. $s =_{\beta\eta} t \Rightarrow C[s] =_{\beta\eta} C[t]$, where $C[s]$ denotes the term that results from plugging s into the context $C[]$.

Usually, the equations for $(\beta), (\eta)$ are directed. We will employ η-expansion, denoted as $\overline{\eta}$.

(β) $C[(\lambda x.t)\ s]$	$\to C[t[s/x]]$	for all contexts $C[]$.
(η) $C[\lambda y.(t\ y)]$	$\to C[t]$	If $y \notin \mathcal{FV}(t)$. The rule is applicable for all contexts $C[]$.
$(\overline{\eta})$ $C[t]$	$\to C[\lambda y.(t\ y)]$	if t is not an abstraction, $type(t)$ is not an elementary type, and t in $C[t]$ is a maximal application. The variable y must be a fresh variable of appropriate type. This reduction is valid for all contexts $C[]$.

If a term cannot be further reduced by $\beta\overline{\eta}$, then it is in $\beta\overline{\eta}$-normal form, also called η-long β-normal form.

It is well-known that the reduction relation defined by these two reductions is strongly terminating and Church-Rosser [Wol93, Hue76, Bar84]. Hence for every term t, there is a $\beta\overline{\eta}$-normal form $t{\downarrow_{\beta\overline{\eta}}}$, which is unique up to $=_\alpha$.

Proposition 4.2. *The following equivalences hold:*

$$s =_{\beta\eta} t \Leftrightarrow s{\downarrow_{\beta\overline{\eta}}} =_\alpha t{\downarrow_{\beta\overline{\eta}}} \Leftrightarrow s{\downarrow_{\beta\eta}} =_\alpha t{\downarrow_{\beta\eta}}$$

4.2 A Higher-Order Unification Procedure

The first complete unification procedures were presented in [Pie73, JP76, Hue75]. We give a short account of a procedure using a non-deterministic set of rules.

The goal is to search (type-correct) instantiations for the free variables in the equations $s_i \doteq t_i, i = 1, \ldots, n$, where $type(s_i) = type(t_i)$ for $i = 1, \ldots, n$, such that the equations are solved modulo $=_{\beta\eta}$.

It is not hard to see that a procedure using the following rules generates a complete set of solutions, though it may not terminate. The higher-order unification algorithm distinguishes the equations according to the head of the η-long forms: A term $\lambda\overrightarrow{x}.X\ \overrightarrow{u}$ where X is a free variable is called *flexible*, a term $\lambda\overrightarrow{x}.v\ \overrightarrow{u}$ is called *rigid*, if v is either a function symbol or a variable in $\overrightarrow{x}$.

So-called flexible-flexible equations are always solvable. For rigid-rigid and rigid-flexible equations, the following rules can be used.

Fail	$\dfrac{\lambda\overrightarrow{x}.u\ \overrightarrow{t} \doteq \lambda\overrightarrow{x}.v\ \overrightarrow{s}}{Fail}$	rigid-rigid and $u \neq v$
Decompose	$\dfrac{\lambda\overrightarrow{x}.u\ \overrightarrow{t} \doteq \lambda\overrightarrow{x}.u\ \overrightarrow{s}}{\lambda\overrightarrow{x}.t_1 \doteq \lambda\overrightarrow{x}.s_1, \ldots}$	rigid-rigid
Imitiation	$\dfrac{\lambda\overrightarrow{x}.X\ \overrightarrow{t} \doteq \lambda\overrightarrow{x}.f\ \overrightarrow{s} \in \Gamma}{\{X \mapsto \lambda\overrightarrow{y}.f\ (X_1\ \overrightarrow{y}) \ldots (X_k\ \overrightarrow{y})\}\Gamma}$	rigid-flexible
Projection	$\dfrac{\lambda\overrightarrow{x}.X\ \overrightarrow{t} \doteq \lambda\overrightarrow{x}.f\ \overrightarrow{s} \in \Gamma}{\{X \mapsto \lambda\overrightarrow{y}.y_j\ (X_1\ \overrightarrow{y}) \ldots (X_k\ \overrightarrow{y})\}\Gamma}$	rigid-flexible

If there are only flexible-flexible equations, then the system of equations is called *presolved*.

4.3 Properties of Higher-Order Unification

It is well-known that higher order unification is undecidable. This was shown for third-order unification in [Hue73]. That second order unification is undecidable was shown later in [Gol81]. This result was sharpened in [Far91] for a restricted signature and in [Lev98, LV00] for severe restrictions on the arity of second order variables and on occurrences of first order and second order variables. The monadic restriction was shown to have an undecidable unification problem, even if types are restricted to third order, in [Nar90]. Here monadic means that function symbols and all types are restricted to be monadic.

4.4 Higher Order Patterns

Higher order patterns are special lambda-terms, where the arguments of every free variable are different bound variables [Mil91]. The intended application for these higher order patterns was a logic programming language with higher order terms. The special case of unification of higher order patterns is decidable [Mil91]. There is also work on further special cases [Pre95].

5 Bounded Second-Order Unification

Syntactic restrictions rule out certain unification problems. Instead it is possible to allow all unification problems, however, restricting the unifiers. One such restriction is bounded second order unification.

We describe the specialization of higher order unification to second-order unification:

- All function symbols in the signature have a type of the form $\iota_1 \to \ldots \to \iota_n$, where ι_i are elementary types.
- In unification problems, every type of a subterm is either elementary or a function type of the form $\iota_1 \to \ldots \to \iota_n$. In particular, every free variable has elementary type (is first-order), or has type $\iota_1 \to \ldots \to \iota_n$ (is second-order). This corresponds to the distinction between first-order variables and second-order variables. The only subterms of function type are the second-order variables.
- There are no subterms that are abstractions in the equations. Hence every variable in the equations is a free variable.

To allow several elementary types leads to a kind of multi-sorted logic and unification with disjoint types. However, from a computational point of view, there is only a minor difference to the simpler case where there is only one elementary type. Hence the usual assumption is that there is exactly one elementary type ι.

It follows also that every bound variable in a unifier has type ι.

Bounded second order unification is the question whether there is a unifier of a second order equational problem, if the set of substitutions is restricted such that the number of bound variables in the solution is smaller than a given constant. Note that this does not restrict the size of unifiers.

The following holds:

Theorem 5.1. *Bounded second order unification is decidable [SS99, SS01a].*

This result shows that from a theoretical viewpoint, an unbounded number of bound variables in substitutions is essential for the undecidability results of second order unification.

The decision algorithm for bounded second order unification has similarities to the algorithm for stratified context unification (see subsection 7.1). The algorithm operates on a set of equations using the unification rules with a tight control. The goal is to non-deterministically transform the equations into presolved form. Again a theorem on the exponent of periodicity is used to bound the number of repetitions of certain rules applications.

We give a short account of the main ideas of the algorithm:

A basic notion is the *surface position*, which is a position in a term that is not in an argument of a free variable. I.e. not within t_i of the subterm $x\ t_1 \ldots t_n$.

Let the relations $\sim_1$ and $>_1$ on the variables V_S of the problem be defined as follows:

If $x \ldots \doteq y \ldots$ is an equation, then $x \sim_1 y$; If $x \ldots \doteq f\ t_1 \ldots t_n$ is an equation, and y is on a surface position of some t_i, then $x >_1 y$. Let $\sim$ denote the

equivalence relation generated by $\sim_1$. Denote the equivalence class of a variable x by $[x]_\sim$. For equivalence classes D_1, D_2 of $\mathcal{V}_S/\sim$ define $D_1 \rhd_1 D_2$ if there exist $x_i \in D_i$ such that $x_1 >_1 x_2$. Let $\rhd$ denote the transitive closure of $\rhd_1$.

There are two cases:

1. The relation $\rhd$ is an irreflexive partial order on $\mathcal{V}_S/\sim$. In this case one can make progress by applying imitation for the variables in maximal equivalence classes.
2. $\rhd$ is not an irreflexive partial order on $\mathcal{V}_S/\sim$. Then there is a sequence of equations which can be seen as a cycle. In this situation the idea is to perform transformations to shorten cycles, or to perform imitations along such a cycle, where the number of rounds can be bounded by the bound on the exponent of periodicity.

There are good reasons for the following conjecture, where we assume that the bound n is given in the input as a string of length n.

Conjecture 5.2. Bounded Second Order Unification is NP-complete

A possible proof of this conjecture could be based on a result of Plandowski on context free grammars that generate exactly one word. The complexity for detecting whether two cfg's generate the same word is polynomial in the size of the grammars [Pla94].

Monadic second order unification is second order unification where the signature is restricted to monadic function symbols. This problem is already treated in [Far88, Hue75], where it is shown to be decidable by using decidability of string unification. The conjecture above would imply that monadic second order unification is $\mathcal{NP}$-complete. Compared to string unification, this is better than the PSPACE upper bound.

6 Bounded Higher-Order Unification

The generalization of bounded second order unification to higher order unification is as follows. There is a given bound that enforces that the overall number of bound variables in unifiers is restricted. This is worked out in [SSS01a], where not only the number of occurrences of bound variables are counted, but also the number of occurrences of lambdas in the unifier. Moreover, it is assumed that the unifiers are in $\beta\overline{\eta}$-normal form. Under these restrictions, unification becomes decidable:

Theorem 6.1. *Bounded Higher Order unification is decidable [SSS01a].*

The algorithm is a generalization of the the algorithm for bounded second order unification. It operates by instantiating free variables that are not in the body of abstractions. The instantiation rules can be seen as a big-step combinations of projection and imitation. The final argument is that a kind of presolved problem is generated: all equations in it are of the form $x \ldots \doteq y \ldots$. Such systems have always a trivial solution that instantiates a constant function for all the variables in heads of equations.

This decidability result is also based on an upper bound on the exponent of periodicity of minimal unifiers. In the case of bounded higher-order unification,

this bound is non-elementary. The proof is based on the corresponding proof for context unification [SSS98].

7 Context Unification

If second order unification is restricted, such that second order variables are unary, and second order variables can be instantiated only by terms with exactly one hole, then this problem is called *context unification*. The following slightly larger fragment is equivalent to context unification: If second order variables may have any arity and if second order variables may be instantiated by lambda terms, where every bound variable occurs at least once, but at most n times for a given number $n \geq 1$.

Context unification can be applied, if the problem in question can be formulated as a second order unification problem, but there is the further constraint, that arguments of context variables must be used exactly once in the solution. This happens frequently in applications.

Open Question 7.1. *Is context unification decidable?*

It is known that context unification is $\mathcal{NP}$-hard (cf. [SSS98]), and that satisfiability of formulas in a logical theory of context unification is undecidable [NPR97a, Vor98].

Applications of context unification are for example in computational linguistics [NPR97a, EN00], in particular as a uniform framework for semantic underspecification of natural language [NPR97b]. The (decidable) fragment of stratified context unification (see below) is expressive enough for applying it in computational linguistics.

A result on the structure of minimal unifiers of context unification problems is obtained, similar in spirit to the exponent of periodicity in string unification. In [SSS98] it is shown that the exponent of periodicity of minimal unifiers of context unification problems is of order $O(2^{c_0+2.14*size(\Gamma)})$, which shows that the representational size of this number is linear. This result is the basis for several good upper complexity estimations.

There are a number of specializations of context unification, where an algorithm for deciding unifiability is possible, i.e. which have a decidable unification problem.

If every function symbol has arity 0 or 1, and the signature has to be used also for the unifiers, the this case is called *monadic context unification*, which is equivalent to string unification, and hence decidable. This means on the other hand that an algorithm that solves context unification has also to solve string unification as a special case.

If for every context variable X, all occurrences of X have the same argument [Com98a, Com98b], then context unification becomes decidable.

Restricting the number of occurrences of each variable to at most 2 gives the fragment of varity-2 context unification. This case is decidable, as shown by Jordi Levy in [Lev96][1].

[1] Note that the proof of decidability for the stratified case is flawed in this paper.

Another decidable case is the (syntactic) fragment, where at most two context variables are permitted, but the number of occurrences is arbitrary and the number as well as the occurrences of first order variables are not restricted [SSS00]. This result is based on a lemma that analyses the possible overlaps of several occurrences of two contexts.

Unfortunately, the author's joint effort together with Klaus Schulz to extend the result to more than two variables did not lead to a (terminating) algorithm, even for three variables.

7.1 Stratified Context Unification

The fragment of *stratified context unification* is defined by a syntactic restriction on the permitted problems. In a system of context equations, let the *second order prefix* of a position be the word of context variables that are on the path from the root of a term to this occurrence. If for every variable, all its occurrences in a unification problem have the same second order prefix, then the context unification problem is called *stratified*.

The fragment of *stratified context unification* was used to solve the question of decidability of the so-called "two-sided distributive unification" [Sza82, Sie84, Sie89]. This problem was tackled in [Con92, Con93, SS92], and finally shown to be decidable in [SS94, SS96, SS98]. The algorithm in [SS98] is based on an algorithm to solve stratified context unification for a restricted signature containing a single binary function symbol and arbitrarily many free constant.

Stratified context unification for arbitrary signatures was shown to be decidable in [SS01b]. The algorithm consists of non-deterministic transformations that keep the property "stratified" and finally use rules for first order unification. Again the bound on the exponent of periodicity helps to avoid infinite repetitions of certain transformation rules.

We give an informal account on the main ideas of the algorithm for unification of stratified context unification problems. An important notion is that of a SO-cycle. A set of equations $X_1(s_1) \doteq r_1, \ldots, X_n(s_n) \doteq r_n$ is called a *second-order cycle (SO-cycle)*, if the following holds: X_i occurs in r_{i-1} for $i = 2, \ldots, n$, X_1 occurs in r_n, and at least one such occurrence is not at the top. The *length* of an SO-cycle is the number of equations in it.

The algorithm distinguishes two cases:

1. There is no SO-cycle. Then the equations can be used to define an irreflexive partial order on the variables. Applying imitation to a maximal equivalence class of variables is a step that makes progress.
2. There is an SO-cycle. Then the transformations either intend to generate a shorter SO-cycle, or the algorithm perform imitations along such a cycle, where the number of rounds can be bounded by the bound on the exponent of periodicity.

It is remarkable that decidability of stratified context unification also shows satisfiability of so-called rewrite constraints, since both problems are equivalent [NTT00]. A recent complexity estimation is that stratified context unification is in PSPACE [SS01c].

8 Higher-Order Matching and Context Matching

8.1 Higher-Order Matching

This is the problem to decide whether in a higher-order unification problem with closed right hand sides unifiability is decidable, where again the equality $=_{\beta\eta}$ is used [Hue76].

Open Question 8.1. *Is higher-order matching decidable?*

For special cases there are decidability results. Second-order and third order matching are decidable [Dow92] and $\mathcal{NP}$-complete [CJ97], while fourth-order matching is decidable [Pad00], but NEXPTIME-hard [Wie99]. For orders above four it is still open whether higher-order matching is decidable [Dow01]. There is a lower non-elementary bound for higher order matching [Wie99]: It is at least as hard as $\beta\eta$-equality, hence at least non-elementary [Sta79]. The case of higher-order matching, where all right hand sides are elementary constants is decidable [Pad96]. There is a terminating procedure by Wolfram [Wol93], which was conjectured to be complete for higher-order matching, but a proof is still missing.

The case of higher-order matching where only $\alpha\beta$-equality is used, is recently claimed to be undecidable [Loa01].

8.2 Second-Order Matching

Similar as in the case of second order unification, second order matching restricts elementary types to one type, and there are no abstractions in the equations.

An algorithm for general second-order matching is given by Huet and Lang [HL78]. Hirata, Yamada and Harao [HYH99] have studied the complexity landscape of the second-order matching problem with respect to several restrictions, i.e. number of second-order variables, number of occurrences of variables, ground, function-free, but not stratification.

Linear higher-order matching was shown $\mathcal{NP}$-complete by de Groote [dG00], where linear in his paper means that only solutions where all functions are linear, i.e. contain each of their bound variable exactly once, are considered.

8.3 Context Matching

A context may be viewed as a linear second-order function with one argument, where the binder is left implicit and the hole is the single occurrence of the bound variable. Thus context matching is a special variant of second order matching, where some solutions are excluded. It can be seen as a restricted form of linear higher-order matching [dG00]. Context matching may have applications in processing XML-documents, since there are similarities to XPath-matching [CD99].

Context matching was shown to be $\mathcal{NP}$-complete in [SSS98].

A study of context matching was undertaken in [SSS01b], where it is shown that stratified context matching is $\mathcal{NP}$-complete, and also that context matching is $\mathcal{NP}$-complete, if every context variable has at most two occurrences. A

specialization that is in P, in fact in $O(n^3)$ is the varity-1 context matching problem; I.e. every variable in the problem occurs exactly once. This result may have practical applications. It formalizes a kind of pattern search in a ground term as follows: $X(f(Y(a), Z(g(b)))) \doteq t$ is the question, whether in t there is an occurrence of the symbol f, such that the first argument contains an a, and the second contains a subterm $g(b)$.

Even here an open problem remains: Determine the complexity of stratified context matching, if the structure of the second order prefixes is linear instead of a tree.

9 Conclusion

Second order and higher order unification have applications in several fields. This holds even for fragments and specializations of a syntactic and/or semantic nature.

Two main open problems remain unsolved: decidability of context unification and of higher order matching.

It is the hope of the author that this paper contributes to the motivation of other researchers to extend the applicability of the different forms of higher order unifications and also to attack and perhaps settle the open questions.

References

[And86] Peter Andrews. *An introduction to mathematical logic and type theory: to truth through proof*. Academic Press, 1986.

[And01] Peter Andrews. Classical type theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 15, pages 965–1007. North-Holland, 2001.

[Bar84] Henk P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, New York, 1984.

[Bar90] Henk P. Barendregt. Functional programming and lambda calculus. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 7, pages 321–363. Elsevier, 1990.

[Bir98] Richard Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall, 1998.

[BMS80] R. Burstall, D. MacQueen, and D.T. Sanella. Hope: an experimental applicative language. In *Proc. LISP Conference*, pages 1363–143, 1980.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BS94] Franz Baader and Jörg Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994.

[CD99] James Clark and Steve DeRose, editors. *XML Path Language (XPath) Version 1.0*. W3C, 16 November 1999.
`http://www.w3.org/TR/1999/REC-xpath-19991116`

[CJ97] Hubert Comon and Yan Jurski. Higher-order matching and tree automata. In *Proc. of CSL 97*, volume 1414 of *Lecture Notes in Computer Science*, pages 157–176, 1997.

[Com98a] Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. of Symbolic Computation*, 25(4):397–419, 1998.

[Com98b] Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. of Symbolic Computation*, 25(4):421–453, 1998.

[Con92] Evelyne Contejean. Unification under distributivity, 1992. Unification workshop 1992, Dagstuhl, Germany.

[Con93] Evelyne Contejean. Solving $*$-problems modulo distributivity by a reduction to AC1-unification. *J. of Symbolic Computation*, 16:493–521, 1993.

[dG00] Philippe de Groote. Linear higher-order matching is NP-complete. In *Proceedings of the 11th Int. Conf. on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 127–140. Springer-Verlag, 2000.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 6, pages 243–320. Elsevier, 1990.

[Dow92] Gilles Dowek. Third order matching is decidable. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 2–10, 1992.

[Dow01] Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001.

[EN00] Katrin Erk and Joachim Niehren. Parallelism constraints. In *Proceedings of the 11th Int. Conf. on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 110–126, 2000.

[Far88] W.A. Farmer. A unification algorithm for second order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988.

[Far91] W.A. Farmer. Simple second-order languages for which unification is undecidable. *J. Theoretical Computer Science*, 87:173–214, 1991.

[GLM97] J. Goubault-Larrecq and I. Mackie. *Proof Theory and Automated Deduction*, volume 6 of *Applied Logic Series*. Kluwer, may 1997. ISBN 0-7923-4593-2.

[Gol81] Warren. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[Gut98] C. Gutierrez. Satisfiability of word equations with constants is in exponential space. In *Proceedings FOCS'98*, pages 112–119, Palo Alto, California, 1998. IEEE Computer Society Press.

[HKMN95] M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A truly functional logic language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995.

[HL78] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[Hue73] Gérard Huet. Undecidability of unification in third-order logic. *Information and Control*, 22:257–267, 1973.

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre 1,2,...ω*. Thèse de doctorat d'etat, Université Paris VII, 1976. In French.

[HYH99] Kouichi Hirata, Keizo Yamada, and Masateru Harao. Tractable and intractable second-order matching problems. In *COCOON 1999*, pages 432–441, 1999.

[JP76] D. Jensen and Tomasz Pietrzykowski. Meachanizing ω-order type theory through unification. *Theoretical Computer Science*, 3(2):123–171, 1976.

[Klo92] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E.Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 2–116. Oxford University Press, 1992.

[KP96] Antoni Kościelski and Leszek Pacholski. Complexity of Makanin's algorithms. *Journal of the Association for Computing Machinery*, 43:670–684, 1996.

[Lev96] Jordi Levy. Linear second order unification. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 332–346, 1996.

[Lev98] Jordi Levy. Decidable and undecidable second order unification problems. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 47–60, 1998.

[Loa01] Ralph Loader. Higher-order β matching is undecidable, 2001. draft.

[LV00] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000.

[Mak77] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.

[MM79] A. Martelli and Ugo Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4(2):258–282, 1979.

[Nar90] Paliath Narendran. Some remarks on second order unification. Technical report, Inst. of Programming and Logics, Department of Computer Science, Univ. of NY at Albany, 1990.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. LICS*, pages 342–349, 1991.

[NPR97a] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *Proceedings of the International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Computer Science*, pages 34–48, 1997.

[NPR97b] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 410–417, Madrid, Spain, 1997.

[NTT00] Joachim Niehren, Sophie Tison, and Ralf Treinen. On rewrite constraints and context unification. *Information Processing Letters*, 74:35–40, 2000.

[Pad96] Vincent Padovani. Decidability of all minimal models. In M. Coppo and S. Beradi, editors, *Types for Proofs and Programs*, volume 1158 of *Lecture Notes in Computer Science*, 1996.

[Pad00] Vincent Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3):361–372, 2000.

[Pau91] Lawrence C. Paulson. *ML for the working programmer*. Cambridge University Press, 1991.

[Pau94] Lawrence C. Paulson. *Isabelle*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Pfe01] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 17, pages 1063–1147. North-Holland, 2001.

[Pie73] Tomasz Pietrzykowski. A complete mechanization of second-order type theory. *J. ACM*, 20:333–364, 1973.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms in context-free languages. In *ESA 94*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470, 1994.

[Pla99] Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *FOCS 99*, pages 495–500, 1999.

[Plo72] Gordon Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[Pre95] Christian Prehofer. *Solving Higher-order Equations: From Logic to Programming*. Ph.D. thesis, Technische Universität München, 1995. In German.

[PW78] Michael S. Paterson and Mark N. Wegman. Linear unification. *JCSS*, 16(2):158–167, 1978.

[Rob65] J.Alan Robinson. A machine oriented logic based on the resolution principle. *J. of the ACM*, 12(1):23–41, 1965.

[Sie75] Jörg H. Siekmann. String-unification. internal report Memo CSM-7, Essex university, 1975.

[Sie84] Jörg H. Siekmann. Universal unification. In *Proc. of 7th CADE*, volume 170 of *LNCS*, pages 1–42, 1984.

[Sie89] Jörg H. Siekmann. Unification theory: A survey. *J. Symbolic Computation*, 7(3,4):207–274, 1989.

[SS92] Manfred Schmidt-Schauß. Some results for unification in distributive equational theories. Internal Report 7/92, Fachbereich Informatik, J.W. Goethe-Universität Frankfurt, Frankfurt, Germany, 1992.

[SS94] Manfred Schmidt-Schauß. An algorithm for distributive unification. Internal Report 13/94, Fachbereich Informatik, J.W. Goethe-Universität Frankfurt, Frankfurt, Germany, 1994.

[SS96] Manfred Schmidt-Schauß. An algorithm for distributive unification. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 287–301, 1996.

[SS98] Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208:111–148, 1998.

[SS99] Manfred Schmidt-Schauß. Decidability of bounded second order unification. Technical Report Frank-report-11, FB Informatik, J.W. Goethe-Universität Frankfurt am Main, 1999.

[SS01a] Manfred Schmidt-Schauß. Decidability of bounded second order unification, 2001. submitted for publication.

[SS01b] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *Journal of Logic and Computation*, 2001. accepted for publication.

[SS01c] Manfred Schmidt-Schauß. Stratified context unification is in PSPACE. In *Proceedings of CSL'01*, 2001. to appear.

[SSS98] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 61–75, 1998.

[SSS00] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *Journal of Symbolic Computation*, 2000. accepted for publication.

[SSS01a] Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher order unification. Frank report 15, Institut für Informatik, 2001.

[SSS01b] Manfred Schmidt-Schauß and Jürgen Stuber. On the complexity of linear and stratified context matching problems. Frank-Report 14, Fachbereich Informatik, J.W. Goethe-Universität Frankfurt, Frankfurt, Germany, 2001. available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html`

[Sta79] Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.

[Sza82] Peter Szabó. *Unifikationstheorie Erster Ordnung*. Ph.D. thesis, Universität Karlsruhe, 1982. In German.

[Tur85] D. A. Turner. Miranda: A non-strict functional language with polymorphic types. In *Functional Programming Languages and Computer Architecture*, number 201 in Lecture Notes in Computer Science, pages 1–16. Springer, 1985.

[Vor98] Sergei Vorobyov. $\forall\exists^*$-equational theory of context unification is Π_1^0-hard. In *MFCS 1998*, volume 1450 of *Lecture Notes in Computer Science*, pages 597–606. Springer-Verlag, 1998.

[Wie99] Tomasz Wierzbicki. Complexity of the higher-order matching. In *Proc.* 16^{th} *CADE*, Lecture Notes in Computer Science, pages 82–96. Springer-Verlag, 1999.

[Wol93] David A. Wolfram. *The clausal theories of types*. Number 21 in Cambridge tracts in theoretical computer science. Cambridge University Press, 1993.

Normal Natural Deduction Proofs (in Non-classical Logics)*

Wilfried Sieg[1] and Saverio Cittadini[2]

[1] Carnegie Mellon University, Department of Philosophy,
5000 Forbes Ave., Pittsburgh, PA 15213, USA
sieg@cmu.edu
[2] Università di Siena,
Dipartimento di Matematica e Scienze Informatiche "R. Magari",
Via del Capitano 15, I-53100 Siena, Italy**
cittadini@unisi.it

Abstract. We provide a theoretical framework that allows the direct search for natural deduction proofs in some non-classical logics, namely, intuitionistic sentential and predicate logic, but also in the modal logic S4. The framework uses so-called intercalation calculi to build up broad search spaces from which normal proofs can be extracted, if a proof exists at all. This claim is supported by completeness proofs establishing in a purely semantic way normal form theorems for the above logics. Logical restrictions on the search spaces are briefly discussed in the last section together with some heuristics for structuring a more efficient search. Our paper is a companion piece to [15], where classical logic was treated.

1 Proof Search for ISL

Proofs and Types, the lively and informative book by Girard, Lafont and Taylor [10], expresses a peculiar tension between the presentation of proofs in sequent and natural deduction calculi. Nd calculi are claimed to be limited to intuitionistic logic (p. 8), and yet we are to think of natural deductions as the "true proof objects" (p. 39). Sequent calculi give the "prettiest illustration of the symmetries of logic" and present "numerous analogies with natural deduction, without being limited to the intuitionistic case" (p. 28). However, they have a serious shortcoming from an algorithmic point of view: the lack of a Curry-Howard isomorphism prevents their use "as a typed λ-calculus" (p. 28).

As far as automated theorem proving (via PROLOG or tableau methods) is concerned, the authors of [10] argue that the sequent calculus provides the underlying ideas: "What makes everything work is the sequent calculus with its deep symmetries, and not particular tricks." (p. 28) And yet, as far as proofs are concerned, the system of sequents is not viewed as primitive: the sequent

* This paper is dedicated to Jörg Siekmann, pragmatic visionary.

** Most of this paper was written in May 1998 while the second author was a visiting researcher in the Deparment of Philosophy at Carnegie Mellon University.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 169–191, 2005.

calculus "sometimes inconveniently complicates situations" (p. 41), as the "rules of the calculus are in fact more or less complex combinations of rules of natural deduction" (p. 39). The sequent calculus is viewed as "only a system which enables us to work on these objects [i.e., the true proof objects]" (p. 39).

There is a technically convenient and heuristically motivated framework that allows the direct search for normal nd-proofs: the intercalation calculus. This calculus was introduced in 1987 for classical sentential logic in the context of the Carnegie Mellon Proof Tutor project; for a first description see [17]. An appropriate extension to classical predicate logic was given in [14], and a much improved (and corrected) version of this material is contained in [15]. The latter paper exploits also a natural Skolem-Herbrand extension (joined with a generalized unification procedure) in order to transfer strategic considerations for proof search from sentential to predicate logic. The heuristically motivating idea for the intercalation calculus is straightforward: given assumptions $A_1, \ldots, A_n$ and a goal G to be derived from the assumptions, one tries to "close the gap" between $A_1, \ldots, A_n$ and G by systematically using elimination rules "from above" and inverted introduction rules "from below". We say that formulas are *intercalated* between assumptions and conclusion. The search space of all possible direct ways of closing the gap is generated in this way; it allows us either to extract a normal nd-proof or to obtain a countermodel in case the inference from $A_1, \ldots, A_n$ to G is invalid.

This, obviously, provides a semantic argument of a normal form theorem for nd-proofs (with a family resemblance to the dual considerations for the cut-free sequent calculus). The intercalation framework makes it very natural to consider restricted classes of normal nd-proofs and to investigate the effect of particular strategies on the form of the resulting nd-proofs. (These matters have been pursued by John Byrnes in his dissertation [3].) What is being exploited here are not the left-right symmetries of the rules for sequents, but rather the deep logical structure of branches in normal derivations. For classical logic that is made possible by a suitable formulation of the negation rules; the claim that nd calculi are limited to intuitionistic logic is, so it seems to us, quite incorrect. In any event, our paper is to demonstrate that the basic ideas of proof search in classical logic can be used for the treatment of non-classical logics, paradigmatically, for intuitionistic sentential and predicate logic (in Sect. 2 and 3) and for the modal system S4 (in Sect. 4). We remark that a uniform approach to proof search in non-classical logics has been pursued in many ways: see for example [2], where the authors use labelled Natural Deduction systems for various non-classical logics which admit a Kripke-style semantics; or [1], where the authors exploit a type-theoretical Logical Framework to encode nd-systems for various modal logics. The beginnings of our work go back to 1991, when Cittadini investigated intuitionistic sentential logic from this perspective in his M.S. thesis under Sieg's direction (see [4]); Cittadini also treated S4 in his paper [5]. A version of the first four sections of this paper appeared in 1998 as [16].

We begin with intuitionistic sentential logic (ISL). As for notation, we follow the conventions of [15], p. 71. The language for ISL has sentential variables,

logical connectives $\wedge$, $\vee$, $\rightarrow$, and the logical constant $\bot$ for absurdity. Negation $\neg\varphi$ is defined as usual by $\varphi \rightarrow \bot$. Nd-rules for ISL are the *proper* elimination (E-) and introduction (I-) rules for $\wedge$, $\vee$ and $\rightarrow$ given in [12]:

$$\frac{\varphi_1 \wedge \varphi_2}{\varphi_i}\ i = 1 \text{ or } 2 \qquad \frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2}$$

$$\frac{\varphi_1 \vee \varphi_2 \quad \begin{array}{c}[\varphi_1]\\ \vdots\\ \psi\end{array} \quad \begin{array}{c}[\varphi_2]\\ \vdots\\ \psi\end{array}}{\psi} \qquad \frac{\varphi_i}{\varphi_1 \vee \varphi_2}\ i = 1 \text{ or } 2$$

$$\frac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2} \qquad \frac{\begin{array}{c}[\varphi_1]\\ \vdots\\ \varphi_2\end{array}}{\varphi_1 \rightarrow \varphi_2}$$

plus the following "ex falso quodlibet" rule $\bot_{\mathbf{q}}$, where φ is taken to be different from $\bot$:

$$\frac{\bot}{\varphi}$$

In [15], for classical logic, negation $\neg$ is a primitive connective, and two rules $\bot_{\mathbf{c}}$ and $\bot_{\mathbf{i}}$ are given for it:

$$\frac{\neg\varphi \quad \begin{array}{c}[\neg\psi]\\ \vdots\\ \varphi\end{array} \quad \begin{array}{c}[\neg\psi]\\ \vdots\\ \neg\varphi\end{array}}{\psi}$$

and

$$\frac{\neg\psi \quad \begin{array}{c}[\psi]\\ \vdots\\ \varphi\end{array} \quad \begin{array}{c}[\psi]\\ \vdots\\ \neg\varphi\end{array}}{\neg\psi}$$

These rules are considered as both E- and I-rules, but not as proper E- or I-rules. The concepts of *p-normal* and *normal* nd-proof are defined as follows: a proof is called p-normal, when no segment of formula occurrences in the proof is such that the first formula in the segment is the conclusion of a proper I-rule or $\bot_{\mathbf{c}}$ and the last formula the major premise of a proper E-rule; it is called normal, if it is p-normal and satisfies the adjacency condition, i.e., the major premise of a $\bot$-rule is not inferred by a $\bot$-rule. For the intuitionistic calculus $\bot_{\mathbf{i}}$ is a derived rule, while $\bot_{\mathbf{c}}$ is of course not; the distinction between p-normal and normal does not apply and, consequently, a proof is called normal if it does not contain a segment whose first formula is the conclusion of an I-rule and whose last formula is the major premise of a proper E-rule.

Paths, taken in the sense of [12], through normal proofs have this special property: they contain a uniquely determined *E-part* and *I-part*, consisting of segments that are major premises of proper E- and I-rules, respectively; these two parts are separated by the *minimum segment* that is a premise of an I-rule. The formulas occurring in the segments of the E-part (I-part) are strictly positive subformulas of the formula occurring in the path's first (last) segment; the formula of the minimum segment is a strictly positive subformula of the formula in the first or last segment. This implies the crucial subformula property of normal proofs: every formula occurring in a normal proof is (the negation of) a subformula either of the goal or of one of the assumptions. The parenthetical addition in the last sentence is needed only for the classical calculus.

Now we give the rules for the intuitionistic sentential ic-calculus IIC_0. Those corresponding to proper E-rules and inverted proper I-rules are just as in the classical case (cf. [15], p. 71; for the $\downarrow$-rules, we also include the local side conditions of p. 74.) We recall that the rules are formulated as Post-productions, and the symbol $\Longrightarrow$ has to be understood informally as follows: to answer the question on the left side of $\Longrightarrow$ affirmatively, it suffices to answer the question(s) on the right side of $\Longrightarrow$ affirmatively:

$\wedge_i \downarrow$**:** $\alpha; \beta?G, \varphi_1 \wedge \varphi_2 \in \alpha\beta, \varphi_i \notin \alpha\beta \Longrightarrow \alpha; \beta, \varphi_i?G$ for $i = 1$ or 2
$\vee \downarrow$**:** $\alpha; \beta?G, \varphi_1 \vee \varphi_2 \in \alpha\beta, \varphi_1 \notin \alpha\beta, \varphi_2 \notin \alpha\beta \Longrightarrow \alpha, \varphi_1; \beta?G$ AND $\alpha, \varphi_2; \beta?G$
$\rightarrow\downarrow$**:** $\alpha; \beta?G, \varphi_1 \rightarrow \varphi_2 \in \alpha\beta, \varphi_2 \notin \alpha\beta, \varphi_1 \neq G \Longrightarrow \alpha; \beta?\varphi_1$ AND $\alpha; \beta, \varphi_2?G$
$\wedge \uparrow$**:** $\alpha; \beta?\varphi_1 \wedge \varphi_2 \Longrightarrow \alpha; \beta?\varphi_1$ AND $\alpha; \beta?\varphi_2$
$\vee_i \uparrow$**:** $\alpha; \beta?\varphi_1 \vee \varphi_2 \Longrightarrow \alpha; \beta?\varphi_i$ for $i = 1$ or 2
$\rightarrow\uparrow$**:** $\alpha; \beta?\varphi_1 \rightarrow \varphi_2 \Longrightarrow \alpha, \varphi_1; \beta?\varphi_2$

Moreover, we have the following rule corresponding to "ex falso quodlibet" (with $\perp$ different from G):

$\perp_{\mathbf{q}}$**:** $\alpha; \beta?G \Longrightarrow \alpha; \beta?\perp$

The search tree (or ic-tree) for ISL is defined just as for classical sentential logic (cf. [15], pp. 72–75) by using all available rules plus $\perp_{\mathbf{q}}$, and is clearly always finite. The assignment of **Y** and **N** to the nodes of the tree is also straightforward, as is the definition of ic-derivation. From an ic-derivation one can construct uniquely an nd-proof, and that proof is normal. The proof of this fact is the same as for classical logic given in [15], pp. 76–77; the only novel case is that of the rule $\perp_{\mathbf{q}}$, which is trivial thanks to the corresponding rule of natural deduction. So for ISL we easily get the *Proof Extraction Theorem*: for any α^* and G^*, if the ic-tree Σ for $\alpha^*?G^*$ evaluates to **Y**, then a normal nd-proof of G^* from assumptions in α^* can be found.

In case the ic-tree for $\alpha^*?G^*$ evaluates to **N**, we want to use the tree itself to define a semantic counterexample to the inference from α^* to G^*. Novel considerations have to come in now, because a semantic counterexample here means a Kripke model $\mathcal{M} = \langle W, R, \Vdash \rangle$ and a world u in W, such that $u \Vdash \varphi$ for all $\varphi \in \alpha^*$, and $u \nVdash G^*$. A *Kripke model for* ISL is a triple $\mathcal{M} = \langle W, R, \Vdash \rangle$, where W is a non-empty set, R is a reflexive and transitive relation on W, and $\Vdash$ is a relation between elements of W and formulas such that, for any $u \in W$:

1. for any sentential variable p, if $u \Vdash p$ and uRv, then $v \Vdash p$;
2. $u \nVdash \bot$;
3. $u \Vdash \varphi_1 \wedge \varphi_2$ iff $u \Vdash \varphi_1$ and $u \Vdash \varphi_2$;
4. $u \Vdash \varphi_1 \vee \varphi_2$ iff $u \Vdash \varphi_1$ or $u \Vdash \varphi_2$;
5. $u \Vdash \varphi_1 \to \varphi_2$ iff for all v such that uRv, if $v \Vdash \varphi_1$, then $v \Vdash \varphi_2$.

Remark 1. A Kripke model is completely determined by W, R, and the behavior of $\Vdash$ on sentential variables.

Given the natural deduction calculus and Kripke semantics for ISL, the *completeness theorem* is standardly formulated as follows: either there is an intuitionistic nd-proof of G from α, or there exist a Kripke model $\mathcal{M} = \langle W, R, \Vdash \rangle$ and a $u \in W$ such that $u \Vdash \varphi$ for all $\varphi \in \alpha$, and $u \nVdash G$. By using our counterexample construction, we will prove a sharpened version where "intuitionistic nd-proof" is replaced by "normal intuitionistic nd-proof". That allows us then to prove a *normal form theorem* by purely semantic means – the topic of the next section.

2 Normal Form Theorem

Assume that the ic-tree Σ for $\pi_0 = \alpha^*?G^*$ evaluates to $\mathbf{N}$, and let $\preceq$ be the natural order relation on Σ. The first step in the construction of a countermodel consists in choosing a subtree P of Σ, and selecting both a set W of question nodes and sets of formulas from P. The construction proceeds in stages. We put π_0 into W and construct inductively a subtree P_0 of Σ, with π_0 as root, along with two sets of formulas T_0 and F_0. Then we select applications of the rule $\to\uparrow$ in P_0 and put the question nodes $\pi_1, \ldots, \pi_k$ thus reached into W. Now, we repeat the first stage starting from these nodes, i.e., we construct subtrees P_j and sets of formulas T_j and F_j $(1 \leq j \leq k)$ just as we did for π_0; then we repeat the second stage and so on, as long as possible. The construction has to terminate, since Σ is finite: the subtree P is the union of all the P_j's (actually, we construct the P_j's and P just as sets of nodes; but we can treat them as subtrees, by considering them ordered by the appropriate restrictions of $\preceq$).

We have to be careful in this process to choose an appropriate ordering of the rules. This makes our construction more intricate than that for classical logic: the fact that an application of $\to\downarrow$ may result in losing track of the goal forces us, in the first stage, to deal with these situations only after all other applicable rules have been tried. Moreover, because of the truth definition for conditionals in a Kripke model, we have to be careful in the second stage when choosing the nodes to which we apply $\to\uparrow$: we choose only those nodes that have been reached after all rules have been tried, except possibly for $\to\downarrow$ whose applications lead to the aforementioned situations). The sets T_j and F_j, for $\pi_j \in W$ have good closure properties, and these properties can be used to define a Kripke model $\mathcal{M}$ on W; $\mathcal{M}$ turns out to be a counterexample to the inference from α^* to G^*.

Now, put π_0 into W and construct sets $P_0(n)$ of question nodes all evaluating to $\mathbf{N}$ by induction on the level n of the nodes. P_0 shall be the union of the $P_0(n)$'s. For the base case, let $P_0(0) = \{\pi_0\}$. Assume that $P_0(n)$ has been defined, with all nodes of $P_0(n)$ evaluating to $\mathbf{N}$. Let $P_0(n) = \{\pi_{n,1}, \ldots, \pi_{n,l}\}$. For $1 \leq i \leq l$, we define $P_0(n+1)_i$ in the following way:

Case 1: $\wedge_1 \downarrow$ applies to $\pi_{n,i}$ of the form $\alpha;\beta?G$ with at least one formula of the form $\varphi_1 \wedge \varphi_2$ in $\alpha\beta$, $\varphi_1 \notin \alpha\beta$. Pick the first such formula in the sequence. Above the rule node is a branch leading to $\alpha;\beta,\varphi_1?G$ which evaluates to **N**. Let $P_0(n+1)_i = \{\alpha;\beta,\varphi_1?G\}$.

Case 2: $\wedge_2 \downarrow$ applies to $\pi_{n,i}$. The situation is as in case 1 with $\alpha;\beta,\varphi_2?G$ in place of $\alpha;\beta,\varphi_1?G$.

Case 3: $\vee \downarrow$ applies to $\pi_{n,i}$ of the form $\alpha;\beta?G$ with at least one formula of the form $\varphi_1 \vee \varphi_2$ in $\alpha\beta$, $\varphi_1 \notin \alpha\beta$, $\varphi_2 \notin \alpha\beta$. Pick the first such formula in the sequence. Above the rule node is a conjunctive branching leading to $\alpha,\varphi_1;\beta?G$ and $\alpha,\varphi_2;\beta?G$. At least one of these nodes evaluates to **N**. If $\alpha,\varphi_1;\beta?G$ evaluates to **N**, $P_0(n+1)_i = \{\alpha,\varphi_1;\beta?G\}$; otherwise, $P_0(n+1)_i = \{\alpha,\varphi_2;\beta?G\}$.

Case 4: $\rightarrow\downarrow$ applies to $\pi_{n,i}$ of the form $\alpha;\beta?G$ (with at least one formula $\varphi_1 \rightarrow \varphi_2$ in $\alpha\beta$, where $\varphi_2 \notin \alpha\beta$, $\varphi_1 \neq G$) and leads to $\alpha;\beta,\varphi_2?G$ and $\alpha;\beta?\varphi_1$, *such that the former evaluates to* **N** (the second possibility, with $\alpha;\beta,\varphi_2?G$ evaluating to **Y** and $\alpha;\beta?\varphi_1$ to **N**, will be treated in case 7). Pick the first such formula in the sequence $\alpha\beta$, and let $P_0(n+1)_i = \{\alpha;\beta,\varphi_2?G\}$.

Case 5: $\wedge \uparrow$ applies to $\pi_{n,i}$ of the form $\alpha;\beta?\varphi_1 \wedge \varphi_2$. Above the rule node is a conjunctive branching leading to $\alpha;\beta?\varphi_1$ and $\alpha;\beta?\varphi_2$. At least one of these nodes evaluates to **N**. If $\alpha;\beta?\varphi_1$ evaluates to **N**, $P_0(n+1)_i = \{\alpha;\beta?\varphi_1\}$; otherwise, $P_0(n+1)_i = \{\alpha;\beta?\varphi_2\}$.

Case 6: $\vee \uparrow$ applies to $\pi_{n,i}$ of the form $\alpha;\beta?\varphi_1 \vee \varphi_2$. Above the rule node is a disjunctive branching leading to $\alpha;\beta?\varphi_1$ and $\alpha;\beta?\varphi_2$. Both of these nodes evaluate to **N**. Let $P_0(n+1)_i = \{\alpha;\beta?\varphi_1,\alpha;\beta?\varphi_2\}$.

Case 7: the previous cases do not apply to $\pi_{n,i} = \alpha;\beta?G$. Let $\varphi_1 \rightarrow \psi_1$, ...,-$\varphi_r \rightarrow \psi_r$ be the list of all conditionals in $\alpha\beta$, with $\psi_h \notin \alpha\beta$, $\varphi_h \neq G$ (note that r may be 0, in which case the list is empty and we simply put $P_0(n+1)_i = \emptyset$). For $1 \leq h \leq r$, above the rule node is a conjunctive branching leading to $\alpha;\beta,\psi_h?G$ and $\alpha;\beta?\varphi_h$. The latter has to evaluate to **N**, since otherwise case 4 would have applied. Let $P_0(n+1)_i = \{\alpha;\beta?\varphi_1,\ldots,\alpha;\beta?\varphi_r\}$.

To complete the inductive step for $n+1$, define $P_0(n+1) = \bigcup_{1\leq i\leq l} P_0(n+1)_i$. Since Σ is finite, the construction terminates, and there is a natural number m such that for any $n \geq m$ we have $P_0(n) = \emptyset$; with μ being the least such number, we define $P_0 = \bigcup_{0\leq n\leq\mu} P_0(n)$. Let T_0 be the set of all formulas occurring on the left side of the question mark in some node of P_0, and F_0 be the set of all formulas occurring on the right side of the question mark in some node of P_0.

For the second stage of our construction, we select those nodes $\pi = \alpha;\beta?G$ of P_0 to which case 7 applied and where G has the form $\varphi \rightarrow \psi$. Then, above the rule node is a branch leading to $\alpha,\varphi;\beta?\psi$ evaluating to **N**. Let $\pi_1,\ldots,\pi_k$ be the nodes thus reached, and put them into W. The process can be repeated starting from these nodes. The whole construction has to terminate, since Σ is finite. In the end, we let P be the union of all the P_j's. Note that the ordering of the cases in the inductive step is irrelevant, except for case 7, which must be the

last one (the reason for this will become clear in the proofs of Lemma 3). Note also that the rule $\perp_{\mathbf{q}}$ is not used in the construction of P: P is used to define sets with good closure properties for which $\perp_{\mathbf{q}}$ is not needed; $\perp_{\mathbf{q}}$ is needed to prove the key property of the Kripke model M formulated in Lemma 4. But we prove first that the sets T_j and F_j, for nodes π_j in $W \subseteq P$, have good closure properties.

Lemma 2. *For any $\pi_j \in W$, the following claims hold:*
(a) $\varphi_1 \wedge \varphi_2 \in T_j$ implies $\varphi_1 \in T_j$ and $\varphi_2 \in T_j$
(b) $\varphi_1 \vee \varphi_2 \in T_j$ implies $\varphi_1 \in T_j$ or $\varphi_2 \in T_j$
(c) $\varphi_1 \rightarrow \varphi_2 \in T_j$ implies $\varphi_1 \in F_j$ or $\varphi_2 \in T_j$
(d) $\varphi_1 \wedge \varphi_2 \in F_j$ implies $\varphi_1 \in F_j$ or $\varphi_2 \in F_j$
(e) $\varphi_1 \vee \varphi_2 \in F_j$ implies $\varphi_1 \in F_j$ and $\varphi_2 \in F_j$
(f) $\varphi_1 \rightarrow \varphi_2 \in F_j$ implies that there exists a $\pi_h \in W$ such that $\pi_j \preceq \pi_h$, $\varphi_1 \in T_h$ and $\varphi_2 \in F_h$.

Proof. For (a)-(e), the key element is that conditionals on the left side of the question mark, all conjunctions and all disjunctions are always dealt with during the construction of P_j. Consider for example (a): if there is a node $\alpha;\beta?G$ in P_j with $\varphi_1 \wedge \varphi_2 \in \alpha\beta$, then this formula is dealt with in cases 1 and 2, hence $\varphi_1 \in T_j$ and $\varphi_2 \in T_j$. Similarly, (b) follows from case 3, (c) from cases 4 and 7, (d) from case 5, (e) from case 6. For (f), if a node $\alpha;\beta?\varphi_1 \rightarrow \varphi_2$ is in P_j, there is a node $\pi_h \in W$ that has been reached by an application of $\rightarrow\uparrow$, such that $\pi_j \preceq \pi_h$, $\varphi_1 \in T_h$ and $\varphi_2 \in F_h$. □

The following lemma shows other important features of the sets T_j and F_j, namely, the T_j's and F_j's do not have common sentential variables, and the T_j's are cumulative.

Lemma 3. *(i) No sentential variable belongs to $T_j \cap F_j$;*[1]
(ii) if $\pi_j, \pi_h \in W$ and $\pi_j \preceq \pi_h$, then $T_j \subseteq T_h$.

Proof. (i) Assume $p \in T_j \cap F_j$, for a sentential variable p. This means that in P_j there are nodes $\rho = \alpha;\beta?G$ with $p \in \alpha\beta$ and $\rho' = \alpha';\beta'?p$. We distinguish three cases.

Case 1: If $\rho \preceq \rho'$, then $p \in \alpha'\beta'$, since no rule of IIC_0 takes away a formula on the left side of the question mark. Thus ρ' evaluates to $\mathbf{Y}$, contrary to the fact that all nodes in P evaluate to $\mathbf{N}$.

Case 2: If $\rho' \prec \rho$, then G must be different from p, since otherwise ρ would evaluate to $\mathbf{Y}$. This means, the formula on the right side of the question mark has been modified in the construction, and since p is a sentential variable this may have happened only through case 7 with an application of $\rightarrow\downarrow$. Let ρ'' be the node to which case 7 applied; clearly, $\rho' \preceq \rho'' \preceq \rho$. Now, the cases that add formulas to the left side of the question mark have been dealt with before case 7, hence they cannot apply above ρ'', and the set of formulas on the left side of the

[1] Actually, one can prove that $T_j \cap F_j = \emptyset$.

question mark remains unchanged in P_j above ρ''. This means that $\rho'' = \alpha;\beta?p$, and since $p \in \alpha\beta$, it evaluates to $\mathbf{Y}$, a contradiction.

Case 3: Assume that ρ and ρ' are on different branches, and let ρ'' be the node at which the highest branching below ρ and ρ' occurred; so either case 6 or case 7 applied to ρ''. But these cases do not change the sequence of formulas on the left side of the question mark, hence any formula that occurs on the left side of the question mark on a branch occurs also on the left side of the question mark on the other branch. Thus, $p \in \alpha'\beta'$, and so ρ' evaluates to $\mathbf{Y}$, a contradiction[2].

(ii) Let $\rho = \alpha;\beta?G$ be a node in P_j, and $\varphi \in \alpha\beta$; we show that $\varphi \in T_h$. If $\rho \preceq \pi_h$, this is immediate since no rule of IIC_0 takes away a formula on the left side of the question mark. If ρ and π_h are on different branches, let $\rho' = \alpha';\beta'?G'$ be the node at which the highest branching occurred. If it occurred through case 6 or 7 in the construction of P_j, then we see as in (i) that any formula that occurs on the left side of the question mark on one branch occurs also on the left side of the question mark on the other branch; hence we conclude $\varphi \in T_h$. Assume then that the branching occurred with an application of $\rightarrow\uparrow$ after the construction of P_j. This means that case 7 applied to ρ'. But then we see, as in (i), that cases 1–4 cannot apply above ρ', and therefore the set of formulas on the left side of the question mark remains unchanged in P_j above ρ'. So $\varphi \in \alpha'\beta'$, and since $\rho' \preceq \pi_h$ we get $\varphi \in T_h$. □

Finally, we come to the definition of the Kripke countermodel: let $\mathcal{M} = \langle W, \preceq, \|- \rangle$, where $\preceq$ is restricted to W, and for any $\pi_j \in W$ and any sentential variable p, $\pi_j \ \|- p$ iff $p \in T_j$. This is enough to define a Kripke model, by Remark 1; condition 1 of the definition of Kripke model holds, because of Lemma 3(ii). The following lemma gives the key property of $\mathcal{M}$.

Lemma 4. *For any $\pi_j \in W$ and any formula φ, the following claims hold:*
(1) $\varphi \in T_j$ implies $\pi_j \|\!\!- \varphi$
(2) $\varphi \in F_j$ implies $\pi_j \not\|\!\!- \varphi$.

Proof. By induction on the complexity of φ; we treat the case of atomic formulas and conditionals; the remaining cases of conjunctions and disjunctions are routine.

Assume φ is a sentential variable p. Then (1) follows from the definition of $\|\!\!-$, and (2) is a consequence of Lemma 3(i).

Assume $\varphi = \bot$. Then (2) follows from the definition of a Kripke model. For (1), suppose there is a node $\rho = \alpha;\beta?G$ in P_j such that $\bot \in \alpha\beta$. Then, when we apply $\bot_{\mathbf{q}}$ to ρ (in the full ic-tree Σ), it leads to a node $\alpha;\beta?\bot$ which evaluates to $\mathbf{Y}$. Hence ρ evaluates to $\mathbf{Y}$, too, contradicting the fact that all nodes in P evaluate to $\mathbf{N}$. So $\bot \notin T_j$, from which (1) follows.

[2] This last case might have been proved in a simpler way by using the fact that the cases in the construction of P_j that add formulas to the left side of the question mark, i.e. cases 1–4, precede also case 6; but we do not want this to be a decisive feature of our countermodel construction, since in the extension to predicate logic we shall not have the same situation.

Assume $\varphi = \varphi_1 \to \varphi_2$. Then, for (1), suppose $\varphi_1 \to \varphi_2 \in T_j$. By Lemma 3(ii), for any $\pi_h \in W$ with $\pi_j \preceq \pi_h$, $\varphi_1 \to \varphi_2 \in T_h$. By Lemma 2(c), this implies $\varphi_1 \in F_h$ or $\varphi_2 \in T_h$. So by induction hypothesis we have $\pi_h \nVdash \varphi_1$ or $\pi_h \Vdash \varphi_2$, and since this holds for any $\pi_h \in W$ with $\pi_j \preceq \pi_h$, by definition of $\Vdash$ we obtain $\pi_j \Vdash \varphi_1 \to \varphi_2$. For (2), assume $\varphi_1 \to \varphi_2 \in F_j$. By Lemma 2(f), this implies that there exists a $\pi_h \in W$ with $\pi_j \preceq \pi_h$ such that $\varphi_1 \in T_h$ and $\varphi_2 \in F_h$. So by induction hypothesis we have $\pi_h \Vdash \varphi_1$ and $\pi_h \nVdash \varphi_2$, and since $\pi_j \preceq \pi_h$, by definition of $\Vdash$ we obtain $\pi_j \nVdash \varphi_1 \to \varphi_2$. □

By applying Lemma 4 to the root node of Σ, we obtain the *Counterexample Extraction Theorem* immediately: if the ic-tree for $\alpha?G$ evaluates to **N**, then it is possible to define from it a counterexample to the inference from α to G, that is, a Kripke model that verifies all the formulas of α and refutes G.

Putting the Proof Extraction Theorem and the Counterexample Extraction Theorem together, we obtain the *Completeness Theorem* for IIC_0 and the sharpened form discussed at the end of Sect. 1 for the nd-calculus.

Theorem 5. *Either the ic-tree for $\alpha?G$ contains an ic-derivation of $\alpha?G$ (from which a normal nd-proof of G from α can be constructed) or it allows the definition of a counterexample to the intuitionistic inference from α to G.*

Soundness and completeness of the ic-calculus provide us with a purely semantic proof of the *Normal Form Theorem* for intuitionistic sentential natural deduction[3]:

Theorem 6. *For every nd-proof there is a normal nd-proof with the same assumptions and conclusion.*

Our completeness proof parallels the one for semantic tableaux given by Fitting in [8]. In that proof, signed formulas are used: i.e., formulas preceded by T (resp. F). These correspond in IIC_0 to formulas on the left (resp. right) side of the question mark. Roughly, the argument in Fitting's proof goes as follows. First one extends the notion of model to signed formulas (more precisely, to sets of signed formulas with suitable closure properties, so-called Hintikka collections). Now assume that the formula φ is not provable in the tableaux system, that is, the set $\{F\varphi\}$ is consistent. Exploiting this hypothesis, construct a Hintikka collection that contains $F\varphi$, and obtain from it a model for this signed formula, i.e. a countermodel for φ. Our proof does not start with a single formula, but with a question $\alpha^*?G^*$. (The approaches are equivalent. Fitting's proof can easily be adapted to start with a set of signed formulas $T\varphi_1, \ldots, T\varphi_n, FG$, with the φ_i's corresponding to the formulas in our α^*.) We assume that $\alpha^*?G^*$ is not "provable" in IIC_0, i.e. the ic-tree for it evaluates to **N**, and use this hypothesis to construct a model for the formulas in α^* which does not verify G^*, i.e. a countermodel for the inference from α^* to G^*. The condition with which a leaf node evaluates to **Y** in IIC_0 corresponds to the condition that makes a set of signed formulas closed; moreover, having an ic-tree evaluating to **Y** corresponds to having a closed tableau.

[3] Because of the finiteness of ic-trees in IIC_0, the ic-calculus also provides a decision procedure for ISL.

3 Extension to Predicate Logic

In this section we extend the metamathematical considerations for ISL to intuitionistic predicate logic (IPL), as was done for classical logic in Sect. 4 of [15]. We use the following nd-rules for the quantifiers (where writing φt assumes that t is free for x in φx or that some bound variables in φx have been renamed):

$$\frac{(\forall x)\varphi x}{\varphi t}\ \forall\mathrm{E} \qquad \frac{\varphi y}{(\forall x)\varphi x}\ \forall\mathrm{I}$$

$$\frac{(\exists x)\varphi x \quad \begin{array}{c}[\varphi y]\\ \vdots\\ \psi\end{array}}{\psi}\ \exists\mathrm{E} \qquad \frac{\varphi t}{(\exists x)\varphi x}\ \exists\mathrm{I}$$

The usual restrictions apply to $\forall$I (y does not have a free occurrence in any assumption on which the derivation of φy depends) and to $\exists$E (y must not have free occurrences in ψ or $(\exists x)\varphi x$ nor in any assumption – other than φy – on which the proof of the upper occurrence of ψ depends).

The ic-calculus for IPL, IIC_1, has all the rules of IIC_0 plus the following ones for the quantifiers, where $\mathcal{T}(\gamma)$ denotes the finite set of terms occurring in the formulas of γ:

$\forall\downarrow$: $\alpha;\beta?G$, $(\forall x)\varphi x \in \alpha\beta$, $t \in \mathcal{T}(\alpha\beta, G)$, $\varphi t \notin \alpha\beta \Longrightarrow \alpha;\beta,\varphi t?G$
$\exists\downarrow$: $\alpha;\beta?G$, $(\exists x)\varphi x \in \alpha\beta$, there is no t such that $\varphi t \in \alpha\beta$, y is new for α, $(\exists x)\varphi x, G \Longrightarrow \alpha,\varphi y;\beta?G$
$\forall\uparrow$: $\alpha;\beta?(\forall x)\varphi x$, y is new for α, $(\forall x)\phi x \Longrightarrow \alpha;\beta?\varphi y$
$\exists\uparrow$: $\alpha;\beta?(\exists x)\varphi x$, $t \in \mathcal{T}(\alpha\beta, (\exists x)\varphi x) \Longrightarrow \alpha;\beta?\varphi t$

In the rules $\exists\downarrow$ and $\forall\uparrow$ the new variable y is chosen in a canonical way (say, the first available one in a fixed ordering of the variables).

Ic-trees are defined as in the classical case. Since in general they are not finite, for the evaluation of nodes we use **Y** and **N** as before, but also the value **U** to evaluate partial ic-trees (see [15], pp. 86–87). If the ic-tree Σ for $\alpha^*?G^*$ evaluates to **Y** it is possible, just as for IIC_0, to extract from Σ a normal nd-proof (i.e. a proof extraction theorem holds). In case Σ evaluates to **N** or **U**, we want to construct from Σ a semantic counterexample to the inference from α^* to G^*. To this end, let us recall Kripke semantics for IPL (see e.g. [6], [9]).

Let D be a nonempty set, and $\mathcal{L}(D)$ be the first-order language with constant symbols for elements in D. A *Kripke model for intuitionistic predicate logic over* D is a quadruple $\mathcal{M} = \langle W, R, \delta, \Vdash \rangle$, where W is a non-empty set, R is a reflexive and transitive relation on W, δ is a function from W to nonempty subsets of D satisfying the *monotonicity condition* (i.e. uRv implies $\delta(u) \subseteq \delta(v)$) and $\Vdash$ is a relation between elements of W and sentences of $\mathcal{L}(D)$ such that $\langle W, R, \Vdash \rangle$ is a Kripke model for ISL, and for any $u \in W$ and any quantified sentence of $\mathcal{L}(D)$:

1. $u \Vdash (\exists x)\varphi(x)$ iff $u \Vdash \varphi(c)$ for some $c \in \delta(u)$;
2. $u \Vdash (\forall x)\varphi(x)$ iff for every v such that uRv, $v \Vdash \varphi(c)$ for every $c \in \delta(v)$.

Remark 7. A Kripke model for intuitionistic predicate logic over D is completely determined by W, R, δ and the behavior of $\Vdash$ on atomic sentences of $\mathcal{L}(D)$.

Now let us treat the counterexample extraction. As in classical logic, the case which requires novel considerations with respect to sentential logic is the extraction of a counterexample from an infinite ic-tree Σ evaluating to **U**. We treat this case by extending the technique used in Sect. 2 for ISL. The construction of P_0 goes as that for ISL in cases 1–6. Then we have the following:

Case 7: $\forall \downarrow$ applies to $\pi_{n,i}$ of the form $\alpha; \beta?G$ with at least one formula of the form $(\forall x)\varphi x$ in $\alpha\beta$ and a term $t \in \mathcal{T}(\alpha\beta, G)$ such that $\varphi t \notin \alpha\beta$. Pick the first such formula in the sequence $\alpha\beta$, and the first such term in $\mathcal{T}(\alpha\beta, G)$ (in some fixed ordering of $\mathcal{T}(\alpha\beta, G)$). Above the rule node is a branch leading to a question node $\alpha; \beta, \varphi t?G$ which evaluates to **U**. Let $P_0(n+1)_i = \{\alpha; \beta, \varphi t?G\}$.

Case 8: the previous cases do not apply to $\pi_{n,i} = \alpha; \beta?G$. Let $\varphi_1 \to \psi_1, \ldots, \varphi_r \to \psi_r$ be the list of all conditionals in $\alpha\beta$, with $\psi_h \notin \alpha\beta$, $\varphi_h \neq G$. For $1 \leq h \leq r$, above the rule node is a conjunctive branching leading to nodes $\alpha; \beta, \psi_h?G$ and $\alpha; \beta?\varphi_h$. The latter has to evaluate to **U**, since otherwise case 4 would have applied. Let $(\exists x_1)\vartheta_1 x_1, \ldots, (\exists x_s)\vartheta_s x_s$ be the list of all existentials in $\alpha\beta$, and y_h (for $1 \leq h \leq s$) be the first variable (in the fixed ordering) which is new for $\alpha, (\exists x_h)\vartheta_h x_h, G$. For $1 \leq h \leq s$, above the rule node is a branch leading to a question node $\alpha, \vartheta_h y_h; \beta?G$ which evaluates to **U**. Finally, if $G = (\exists x)\chi x$, i.e. $\exists \uparrow$ applies to $\pi_{n,i}$, above the rule node is a disjunctive branching leading to nodes of the form $\alpha; \beta?\chi t$ (one for each $t \in \mathcal{T}(\alpha\beta, (\exists x)\chi x)$), all evaluating to **U**; in this case, let $X_0(n+1)_i = \{\alpha; \beta?\chi t \mid t \in \mathcal{T}(\alpha\beta, (\exists x)\chi x)$, otherwise, let $X_0(n+1)_i = \emptyset$. Now let $P_0(n+1)_i = \{\alpha; \beta?\varphi_1, \ldots, \alpha; \beta?\varphi_r, \alpha, \vartheta_1 y_1; \beta?G, \ldots, \alpha, \vartheta_s y_s; \beta?G\} \cup X_0(n+1)_i$ (note that r or s may be 0, in which case the corresponding list is empty; if they are both 0, we simply put $P_0(n+1)_i = X_0(n+1)_i$).

The main reason for having such an intricate case 8 is the rule $\exists \downarrow$. In fact, this rule introduces new variables, so it might cause the construction to go on indefinitely. But we want to treat the special applications of $\to\downarrow$ as the last case; hence we have to reach this at some finite stage. Therefore we are forced to treat them at the same time as the applications of $\exists \downarrow$. The introduction of new variables due to $\exists \downarrow$ also forces us to apply $\exists \uparrow$ (if it is the case) at this stage if we want to get the appropriate closure property.

This construction is of course infinite, but we can still define P_0 as the union of all the $P_0(n)$'s, for $0 \leq n < \omega$, and T_0 and F_0 as in the sentential case, and proceed with the second stage of the construction. We select those nodes $\pi = \alpha; \beta?G$ of P_0 to which case 8 has applied and such that $\to\uparrow$ or $\forall \uparrow$ applies to π, i.e. G has the form $\varphi \to \psi$ or $(\forall x)\vartheta x$. Then, above the rule node is a branch leading to a question node $\alpha, \varphi; \beta?\psi$, which evaluates to **U**, or a branch leading to a question node $\alpha; \beta?\vartheta y$, with y new for $\alpha, (\forall x)\vartheta x$. Let $\pi_1, \ldots, \pi_k, \ldots$ be the nodes thus reached, and put them into W. Now the process can be repeated starting from these nodes, with the definition of P_j, T_j and F_j, and so on. In the end, we let P be the union of all the P_j's[4].

[4] Here, clearly, the index j has to range over countable ordinals and not just natural numbers.

We obtain the following extension of Lemma 2.

Lemma 8. *The closure properties (a)–(f) of Lemma 2 hold for the predicate case. Furthermore, we have the following:*

(g) $(\exists x)\varphi x \in T_j$ *implies* $\varphi t \in T_j$*, for some* t *occurring in* P_j*;*

(h) $(\exists x)\varphi x \in F_j$ *implies* $\varphi t \in F_j$*, for every* t *occurring in* P_j*;*

(i) $(\forall x)\varphi x \in T_j$ *implies that for every* $\pi_h \in W$ *such that* $\pi_j \preceq \pi_h$ *and every* t *occurring in* P_j*,* $\varphi t \in T_h$*;*

(j) $(\forall x)\varphi x \in F_j$ *implies that there exist a* $\pi_h \in W$ *such that* $\pi_j \preceq \pi_h$ *and a* t *occurring in* P_h *such that* $\varphi t \in F_h$*.*

Proof. (a)–(f) are as in the sentential case. (g) follows from case 8. For (h), case 8 gives us $\varphi t \in F_j$ for every t occurring in P_j up to the stage to which the case has applied; but $\exists \downarrow$ and $\exists \uparrow$ are treated at the same time, and the applications of $\exists \downarrow$ do not change the shape of the goal, so the new terms t' that occur in P_j are treated in a successive case 8, and for all of them we get again $\varphi t' \in F_j$. (i) follows from case 7, and the fact that no rule of IIC_1 takes away a formula on the left side of the question mark. For (j), if $(\forall x)\varphi x \in F_j$, then case 8 gives us a new node $\pi_h \in W$ and a term y occurring in P_h such that $\varphi y \in F_h$. □

Moreover, we can prove the following analogue of Lemma 3. The proof requires only slight modifications, hence we omit it.

Lemma 9. *(i) No atomic formula belongs to* $T_j \cap F_j$*;*[5]

(ii) if $\pi_j, \pi_h \in W$ *and* $\pi_j \preceq \pi_h$*, then* $T_j \subseteq T_h$*.*

Finally, for the definition of our countermodel, let D be the set of terms occurring in P, and for any node π_j, $\delta(\pi_j)$ shall be the set of terms occurring in P_j. The monotonicity condition holds, because of the canonical choice of new variables for $\exists \downarrow$ in the construction of each P_j. Let $\mathcal{M} = \langle W, \preceq, \delta, \Vdash \rangle$, where $\preceq$ is restricted to W, and for any $\pi_j \in W$ and any atomic formula φ of $\mathcal{L}(D)$, $\pi_j \Vdash \varphi$ iff $\varphi \in T_j$ (this is enough to define a Kripke model, by Remark 7).

We then obtain the following analogue of Lemma 4. The proof is essentially identical (induction on the complexity of φ, using Lemmas 8 and 9 instead of 2 and 3).

Lemma 10. *For any* $\pi_j \in W$ *and any formula* φ *of* $\mathcal{L}(D)$*, the following hold:*

(1) $\varphi \in T_j$ *implies* $\pi_j \Vdash \varphi$

(2) $\varphi \in F_j$ *implies* $\pi_j \nVdash \varphi$*.*

As in the sentential case, we obtain the *Counterexample Extraction Theorem* by applying the last lemma to the root node of Σ, and again from this we get the *Completeness Theorem* and the *Normal Form Theorem*.

4 The Modal Logic S4

We now apply the ideas underlying intercalation calculi to modal logic by giving an appropriate ic-calculus for the modal system S4. The language contains now

[5] Again, one can prove that $T_j \cap F_j = \emptyset$.

sentential variables, logical connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$, and the modal operator $\Box$. The modal operator $\Diamond$ is defined: $\Diamond\varphi = \neg\Box\neg\varphi$ (this is to follow Prawitz's notation [12], and to save work in the proof of the Counterexample Extraction Theorem; it is not difficult to give a version where $\Diamond$ is primitive, too).

Following [12], we use a nd-system for S4 which has all the rules of classical sentential logic plus the following ones for $\Box$:

$$\frac{\Box\varphi}{\varphi}\ \Box\text{E} \qquad\qquad \frac{\varphi}{\Box\varphi}\ \Box\text{I}$$

The I-rule for $\Box$ has to satisfy certain restrictions. Prawitz gives three versions of such restrictions and shows that the resulting systems are actually equivalent. In the first version, the rule can be applied only if all the open assumptions on which φ depends in the deduction are of the form $\Box\psi$ (in Prawitz's terms, they are *modal formulas*). To define the second version, the notion of *essentially modal formula* is introduced inductively as follows:

1. all modal formulas are essentially modal;
2. if φ_1 and φ_2 are essentially modal, then so are $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$.

The restrictions on $\Box$I are liberalized for the second version: essentially modal formulas are allowed as φ's open assumptions. Finally, in the third version, the restrictions are further liberalized: $\Box$I can be applied to φ when, for each open assumption ψ on which φ depends there is an essentially modal formula ϑ such that

(i) ϑ is ψ or φ, or ϑ occurs on the path from ψ to φ,

and

(ii) all assumptions on which ϑ depends are also assumptions on which φ depends.

The reason for this liberalization is that the first two versions, as Prawitz shows, do not allow a normal form theorem. We shall discuss this issue in connection with the Proof Extraction Theorem; but first, we introduce the S4 ic-calculus.

Here we have all the rules for the classical calculus $\mathrm{IC}_0(\mathcal{F})$ (cf. [15], p. 71), i.e. the $\downarrow$- and $\uparrow$-rules of IIC_0, in particular, the following $\perp$-rules:

$\perp_{\mathbf{c}}(\mathcal{F})$**:** $\alpha;\beta?G,\varphi \in \mathcal{F}(\alpha,\neg G) \Longrightarrow \alpha,\neg G;\beta?\varphi$ AND $\alpha,\neg G;\beta?\neg\varphi$
$\perp_{\mathbf{i}}(\mathcal{F})$**:** $\alpha;\beta?\neg G,\varphi \in \mathcal{F}(\alpha,G) \Longrightarrow \alpha,G;\beta?\varphi$ AND $\alpha,G;\beta?\neg\varphi$

where $\mathcal{F}(\gamma)$ is the set of all unnegated proper subformulas of formulas in γ and the unnegated part of all negations which are subformulas of formulas in γ, and the following rules for $\Box$:

$\Box\downarrow$**:** $\alpha;\beta?G,\Box\varphi \in \alpha\beta \Longrightarrow \alpha;\beta,\varphi?G$
$\Box\uparrow$**:** $\alpha;\beta?\Box\varphi \Longrightarrow (\alpha\beta)_\nu?\varphi$

where $(\alpha\beta)_\nu$ is the sequence of the modal formulas occurring in $\alpha\beta$. The choice of these rules is inspired by the semantic tableaux version of S4 given by Fitting

in [9]. Note that in the rule $\Box \uparrow$ we have $(\alpha\beta)_\nu$ and not $\alpha_\nu; \beta_\nu$ on the left side of the question mark. In fact, when we write $\alpha; \beta?G$ we mean that the formulas of β have been obtained from those of α via $\downarrow$-rules, and we cannot claim in general that formulas of β_ν can be obtained from formulas of α_ν via $\downarrow$-rules. Thus, we take the sequence of modal formulas in $\alpha\beta$ as assumptions of a new proof of φ (clearly, if in this new proof $\downarrow$-rules are used, we find nodes of the form $\alpha'; \beta'?G'$ once again).

The definition of the S4 *ic-tree* is straightforward, as are the assignment of **Y** and **N** to the nodes of the tree and the definition of S4 *ic-derivation*. S4 ic-trees are clearly finite.

Now, if we refer to Prawitz's third version of the S4 nd-system, we can easily obtain a *proof extraction theorem*: the argument, as we shall see, proceeds by induction on the height of the ic-derivation, just as in the classical and the intuitionistic case (again, see [15], pp. 76–77). But before doing that, we want to describe why such an argument would not work if we refer to one of the first two versions: this may give a better insight into the liberalization on the constraints for $\Box$I which defines the third version.

The problem is that, in the induction step, some of the ic rules introduce new open assumptions, and these may be neither modal nor essentially modal. Thus, if there is an application of $\Box$I in the nd-proof we obtain by induction hypothesis, the restrictions on $\Box$I in the considered version may be violated. In order to obtain the result, then, the restrictions must be liberalized in such a way that they do not refer only to the shape of the open assumptions, but also to that of the formulas obtained from the open assumptions via $\downarrow$-rules. As the $\downarrow$-rules correspond to the E-rules, we must allow the application of $\Box$I to a formula φ when there are (essentially) modal formulas obtained via E-rules in any path from φ to an open assumption: exactly what Prawitz does with his third version! This, by the way, is not so surprising, since our central concern is to provide a semantic proof of a normal form theorem for S4 natural deduction, and such a theorem does not hold for the first two versions.

We have the following *Proof Extraction Theorem*:

Theorem 11. *For any α and G, if the* S4 *ic-tree for $\alpha?G$ evaluates to* **Y**, *then a p-normal nd-proof (in the third version of the* S4 *nd-system) of G from assumptions in α can be found.*

Proof. (sketch): By induction on the height of the ic-derivation. The treatment of classical rules is identical to that for classical logic ([15], pp. 76–77). The ic-rules for $\Box$ are handled with the corresponding nd-rules. In the case of $\Box \uparrow$ the restrictions on $\Box$I are satisfied, thanks to the restriction to $(\alpha\beta)_\nu$ of the set of formulas $\alpha\beta$ on the left side of the question mark. Moreover, the restrictions on $\Box$I are preserved through all induction steps. Indeed, in each case we have by induction hypothesis, for all possible application of $\Box$I to a formula φ and all open assumptions ψ on which φ depends, a formula ϑ satisfying the conditions of the definition: and this ϑ is still present whichever rule we apply, even new open assumptions are introduced. Moreover, the nd-proofs extracted from ic-derivations are clearly p-normal, again exploiting the fact that $\downarrow$-rules are only applied from above and $\uparrow$-rules only from below. $\Box$

Now we give an example to show how the $\downarrow$-rules combine with $\Box\uparrow$ so that the S4 nd-proofs extracted from S4 ic-derivations satisfy the restrictions imposed on $\Box$I (i.e., they are indeed S4 nd-proofs). In this example, an application of $\rightarrow\downarrow$ puts in β a formula of the form $\Box\varphi$, which is not present in α (the original set of hypotheses), but is necessary to make the top node evaluate to **Y** after the application of $\Box\uparrow$. Consider the following S4 ic-derivation for $p, p \rightarrow \Box q ? \Box\Box q$:

$$\begin{array}{c} \mathbf{Y} \\ | \\ \Box q ? \Box q \\ | \\ \Box\uparrow \\ | \\ p, p \rightarrow \Box q ; \Box q ? \Box\Box q \\ | \\ \rightarrow\downarrow \\ | \\ p, p \rightarrow \Box q ? \Box\Box q \end{array}$$

From it we can extract the following S4 nd-proof, where $\Box q$ is the ϑ required for the application of $\Box$I:

$$\dfrac{\dfrac{p \quad p \rightarrow \Box q}{\Box q}}{\Box\Box q}$$

Now we turn to completeness; we start by recalling Kripke semantics for S4. A *Kripke model for* S4 is a triple $\mathcal{M} = \langle W, R, \Vdash \rangle$, where W is a non-empty set, R is a reflexive and transitive relation on W, and $\Vdash$ is a relation between elements of W and formulas such that, for any $u \in W$:

1. $u \Vdash \varphi_1 \wedge \varphi_2$ iff $u \Vdash \varphi_1$ and $u \Vdash \varphi_2$;
2. $u \Vdash \varphi_1 \vee \varphi_2$ iff $u \Vdash \varphi_1$ or $u \Vdash \varphi_2$;
3. $u \Vdash \varphi_1 \rightarrow \varphi_2$ iff $u \Vdash \varphi_1$ implies $u \Vdash \varphi_2$;
4. $u \Vdash \neg\varphi$ iff $u \nVdash \varphi$;
5. $u \Vdash \Box\varphi$ iff for all v such that uRv, $v \Vdash \varphi$.

Clearly, Remark 1 holds also for Kripke models of S4. As in the classical and intuitionistic cases, we want to prove a *Counterexample Extraction Theorem*: that means, in this case, that an S4 ic-tree Σ for $\alpha ? G$ which evaluates to **N** can be used to define a Kripke model for S4 $\mathcal{M} = \langle W, R, \Vdash \rangle$ and a $u \in W$ such that $u \Vdash \varphi$ for all $\varphi \in \alpha$, and $u \nVdash G$.

Before proceeding with the detailed proofs we sketch the argument for the counterexample extraction from Σ. The proof combines the technique used for the classical case (i.e. the construction of a canonical branch) with that for the intuitionistic case (i.e. for the construction of a Kripke model). The construction proceeds in stages. At the first stage, we select *a single branch* P_0 of Σ, all of whose nodes evaluate to **N** (this is done by using the $\perp$-rules systematically, as in the classical case), and put the root node of Σ in a set W. The following stage

applies $\Box \uparrow$ to all nodes of P_0 to which this rule is applicable, except for the root (hence these nodes will become branching points in our subtree). Then we put the nodes thus reached in W, and start the construction again from them. In this way we obtain sub-branches $P_1, \ldots, P_k$, and then the process continues. Of course, the construction has to terminate, since the whole tree is finite. The union of all the P_j's will be a subtree of Σ, with W as a subset. Then each node $\pi_j \in W$ will be the root of some P_j. Moreover, it will be possible to prove appropriate closure properties of the P_j's, and then to define a Kripke model on W with the required property of being a counterexample for $\alpha?G$.

Now, assume the ic-tree Σ for $\alpha?G$ evaluates to $\mathbf{N}$. We begin with the construction of P_0. Define φ^- and φ^+ as in the classical case (namely, $\varphi^- = \psi$ if $\varphi = \neg\psi$ and $\varphi^- = \neg\varphi$ otherwise; $\varphi^+ = \psi$ if $\varphi = \neg\neg\psi$ and $\varphi^+ = \varphi$ otherwise), and enumerate $\mathcal{F}(\alpha, G^-)$ by $\langle H_i \rangle_{i \in I}$, where $I = \{i \mid 1 \leq i \leq n\}$. Put the node $\alpha?G$ in W.

The sequence of nodes $P_0^*(0), \ldots$ is defined as follows. First, let $\alpha_0 = \alpha$, $\lambda_0 = 0$, $G_0 = H_0 = G$. Then, λ_{m+1} is defined according to the following cases:

Case 1: there is a j such that $\lambda_m < j \leq n$ and H_j is not of the form $\Box\varphi$ and $H_j \notin \alpha_m$ and $\neg H_j \notin \alpha_m$. Then λ_{m+1} is the least such j.

Case 2: the previous case does not apply, but there is a j such that $\lambda_m < j \leq n$ and H_j is of the form $\Box\varphi$ and $H_j \notin \alpha_m$ and $\neg H_j \notin \alpha_m$ and $\alpha_m?\neg H_j$ evaluates to $\mathbf{N}$. Then λ_{m+1} is the least such j.

Case 3: the previous cases do not apply, but there is a j such that $\lambda_m < j \leq n$ and H_j is of the form $\Box\varphi$ and $H_j \notin \alpha_m$ and $\neg H_j \notin \alpha_m$ and $\alpha_m?\neg H_j$ evaluates to $\mathbf{Y}$. Then λ_{m+1} is the least such j.

Case 4: the previous cases do not apply. Then let $\lambda_{m+1} = 0$.

Then, let $G_m = \neg H_{\lambda_m}$ if $\alpha_m?\neg H_{\lambda_m}$ evaluates to $\mathbf{N}$ and $G_m = H_{\lambda_m}$ otherwise, $\alpha_{m+1} = \alpha_m, G_m^-$, $P_0^*(2m) = \alpha_m?G_m$, $P_0^*(2m+1) = \bot_{\mathbf{i}}, H_{\lambda_{m+1}}$ if G_m is a negation, $P_0^*(2m+1) = \bot_{\mathbf{c}}, H_{\lambda_{m+1}}$ otherwise.

Let μ be the smallest m with $\lambda_{m+1} = 0$, and define P_0 to be P_0^* restricted to $\{m \mid m \leq 2\mu\}$. Now, consider the nodes of the form $\alpha'?\Box\varphi$ in P_0 (excluding the root). These nodes appear in P_0 only because of case 3, hence only after all the formulas of $\mathcal{F}(\alpha, G^-)$ not of the form $\Box\varphi$ have been used (this fact will be crucial for proving the closure properties of the sets P_j). To each of these nodes, the rule $\Box \uparrow$ is applicable, leading to a node of the form $\alpha'_\nu?\varphi$ (which evaluates to $\mathbf{N}$). Call $\{\pi_1, \ldots, \pi_k\}$ the nodes thus obtained. Put each π_j in W, and start from it the construction of a branch P_j, choosing at each stage the following node according to the cases for P_0. Then, repeat the process. Finally, let P be the union of all the P_j's.

Now, let $W = \{\pi_0, \ldots, \pi_r\}$, each π_j being the root of P_j. For $0 \leq j \leq r$, let $\bar{\pi}_j = \alpha_{\mu_j}?G_{\mu_j}$ be the top node of P_j, and define $A_j = \{\varphi \mid \varphi \in \alpha_{\mu_j}, G_{\mu_j}^-\}$. The following lemma describes the important syntactic closure properties of the sets A_j.

Lemma 12. *For $0 \leq j \leq r$, the following claims hold:*
(i) if $\varphi \in A_j$, then $\varphi^- \notin A_j$;
(ii) if φ is a subformula of an element in A_j, then $\varphi^+ \in A_j$ or $\varphi^- \in A_j$;
(iii) if $\neg\neg\varphi \in A_j$, then $\varphi \in A_j$;
(iv) if $\varphi_1 \wedge \varphi_2 \in A_j$, then $\varphi_1^+ \in A_j$ and $\varphi_2^+ \in A_j$;
if $\neg(\varphi_1 \wedge \varphi_2) \in A_j$, then $\varphi_1^- \in A_j$ or $\varphi_2^- \in A_j$;
(v) if $\varphi_1 \vee \varphi_2 \in A_j$, then $\varphi_1^+ \in A_j$ or $\varphi_2^+ \in A_j$;
if $\neg(\varphi_1 \vee \varphi_2) \in A_j$, then $\varphi_1^- \in A_j$ and $\varphi_2^- \in A_j$;
(vi) if $\varphi_1 \rightarrow \varphi_2 \in A_j$, then $\varphi_1^- \in A_j$ or $\varphi_2^+ \in A_j$;
if $\neg(\varphi_1 \rightarrow \varphi_2) \in A_j$, then $\varphi_1^+ \in A_j$ and $\varphi_2^- \in A_j$;
(vii) if $\Box\varphi \in A_j$, then for each i with $\pi_j \preceq \pi_i$ it holds that $\varphi^+ \in A_i$;
if $\neg\Box\varphi \in A_j$, then there is an i with $\pi_j \preceq \pi_i$ and $\varphi^- \in A_i$.

Proof. (i)-(vi) are proved exactly as in the Closure Lemma for the classical case ([15], pp. 80–82; indeed, the rules of IC_0 are all available here).

To prove the first part of (vii), observe that if $\Box\varphi \in A_j$, then $\Box\varphi$ must appear on the left side of the question mark below any node of P_j to which $\Box \uparrow$ is applied, that is, before any new node is put in W. This is because of the ordering of the cases: indeed, having $\Box\varphi$ on the left side means that this formula has been dealt with in case 2, and new nodes are put in W only when case 3 has applied. Moreover, formulas of the form $\Box\varphi$ are never taken away from the left side of the question mark. From these two facts it follows immediately that for each i such that $\pi_j \preceq \pi_i$ it holds $\Box\varphi \in A_i$. But now it is not possible that $\varphi^- \in A_i$: in fact, an application of $\Box \downarrow$ would make $\bar{\pi_j}$ evaluate to $\mathbf{Y}$, while all the nodes in P_j evaluate to $\mathbf{N}$. Therefore, by (ii), $\varphi^+ \in A_i$.

For the second part of (vii), observe that if $\neg\Box\varphi \in A_j$, then $\Box\varphi$ has been dealt with in case 3. Thus a new node $\pi_i = \alpha'?\varphi$ has been put in W, and since the rule applied to π_i is either $\bot_{\mathbf{i}}$ or $\bot_{\mathbf{c}}$, φ^- appears on the left side of the question mark in P_i, whence $\varphi^- \in A_i$. □

Now consider $\mathcal{M} = \langle W, \preceq, \Vdash \rangle$, where for any sentential variable p and any $\pi_j \in W$ we set $\pi_j \Vdash p$ iff $p \in A_j$.

Lemma 13. *For any $\pi_j \in W$ and any formula φ, if $\varphi \in A_j$, then $\pi_j \Vdash \varphi$.*

Proof. Straightforward induction on the complexity of φ, using the closure properties of Lemma 12. □

By applying Lemma 13 to the root node π_0 of the tree, we conclude that $\pi_0 \Vdash \varphi$, for each $\varphi \in \alpha, G^-$, that is, π_0 verifies all formulas in α and falsifies G. Thus we have a semantic counterexample for the inference from α to G, and so the *Counterexample Extraction Theorem* is proved.

Again, this implies the *Completeness Theorem* for the S4 ic-calculus and, since the nd-proofs obtained from ic-derivations are p-normal, the *p-Normal Form Theorem* for the version of the S4 nd-system considered. Moreover, the finiteness of S4 ic-trees yields another proof of the *Finite Model Property* for this logic (namely, if a formula is not provable in S4, then it is falsified in some *finite* Kripke model for S4), and a decision procedure for it.

To obtain a *normal* form theorem, restrictions on the set F of contradictory pairs available for the $\perp$-rules can be introduced, as for the classical case (see [15], pp. 83–84). With these restrictions, nd-proofs obtained from ic-derivations are normal, and we are still able to prove the Counterexample Extraction Theorem: however, now not only the $\perp$-rules and the $\Box$-rules are involved in the counterexample construction, but possibly all the other rules. We can obtain a sharpened completeness theorem and a normal form theorem (for the third version of Prawitz's system!), in the following form:

Theorem 14. *Either the* S4 *ic-tree for* $\alpha?G$ *contains an* S4 *ic-derivation of* $\alpha?G$ *(and hence allows to construct a normal* S4 *nd-proof of* G *from* α*) or it allows the definition of a counterexample to the inference from* α *to* G*.*

Theorem 15. *For every* S4 *nd-proof there is a normal* S4 *nd-proof with the same assumptions and conclusion.*

5 Heuristics for Search

In this last section we discuss very briefly strategic issues for proof search in intuitionistic sentential as well as predicate logic. Logical restricions on the search space and appropriate heuristics are needed in order to obtain an efficient procedure. As a first step in our discussion we review the coarse structure of proof search in classical predicate logic.

The search for an answer, i.e., an ic-derivation, to the question $\alpha; \beta?G$ involves three distinct components: (i) use of $\downarrow$-rules, (ii) use of $\uparrow$-rules, (iii) use of $\perp$-rules (with a limited set of contradictory pairs of formulas). It is step (i) that is central and that is taken in a *goal-directed way.* If the question:

(*) Is G a strictly positive subformula of a formula in $\alpha\beta$?

has an affirmative answer, this step provides sequences of $\downarrow$-rule applications that *extract* strictly positive occurrences of G in elements of $\alpha\beta$. The connecting formula sequences consist of the major premisses of the $\downarrow$-rules and require in general answers to new questions, namely, those raised in the minor premisses of the rule applications.

The Skolem-Herbrand expansion was introduced in [15], in order to obtain an appropriate generalization of this *extraction strategy.* It will be described below. Here we just emphasize that the goal-directedness of applications of the $\downarrow$-rules (including the quantifier rules) is obtained by generalizing the question (*) to

(**) Is G unifiable with a strictly positive canonical subformula of a formula in $\alpha\beta$?

A subformula is considered to be *canonical*, if quantifiers are instantiated by terms that match the $\downarrow$-quantifier rules of the Skolem-Herbrand expansion, i.e., those terms would be used by the extracting $\downarrow$-rules. Having indicated the point of the Skolem-Herbrand expansion, let us describe it in reasoned detail.

We assume that the language for the intercalation calculus has just the set $X = \{x, x_0, x_1, \ldots\}$ as its set of variables. Then the language of the Skolem-Herbrand expansion has in addition a set $Y = \{y, y_0, y_1, \ldots\}$ of bound variables,

a set $Z = \{z, z_0, z_1, \ldots\}$ of parameters, and a set $F = \{f, f_0, f_1, \ldots\}$ of function symbols. (X, Y and Z are all disjoint, and F contains infinitely many function symbols for each arity n, n a natural number.) If γ is a sequence of formulas, by $FV(\gamma)$ we mean (a sequence of all) the parameters from the set Z which occur as terms in the elements of γ. The calculus is obtained by replacing the quantifier rules with the following ones:

$\forall \downarrow$: $\alpha; \beta?G$, $(\forall x)\varphi x \in \alpha\beta \Longrightarrow \alpha; \beta, \varphi z?G$ for some new z
$\exists \downarrow$: $\alpha; \beta?G$, $(\exists x)\varphi x \in \alpha\beta$, $\bar{z} = FV(\alpha, (\exists x)\varphi x, G) \Longrightarrow \alpha, \varphi f(\bar{z}); \beta?G$ for some new f
$\forall \uparrow$: $\alpha; \beta?(\forall x)\varphi x$, $\bar{z} = FV(\alpha, (\forall x)\varphi x) \Longrightarrow \alpha; \beta?\varphi f(\bar{z})$ for some new f
$\exists \uparrow$: $\alpha; \beta?(\exists x)\varphi x \Longrightarrow \alpha; \beta?\varphi z$ for some new z

Correctness for the Skolem-Herbrand expansion in the classical case is proved in [15], Sect. 6, using an appropriate notion of unification: a derivation in the intercalation calculus for $\alpha?G$ exists if and only if a derivation in the Skolem-Herbrand expansion for $\alpha?G$ exists. Thus, the expansion can be considered "as a convenient technical tool for automated proof search" ([15], p. 95). A technical tool that is, as we pointed out, of critical importance for generalizing the (sentential) extraction strategy, i.e., the goal-directed use of elimination rules.

Although we do not go into the details, our claim is that the same approach can be pursued for the intuitionistic case. This may look surprising, in view of the considerations of Shankar in [13]. In fact, Shankar claims that "the impermutability of certain pairs of inferences in LJ makes it incorrect to directly use Herbrandization for proof search" (here LJ stands for intuitionistic predicate sequent calculus, and "Herbrandization" stands, roughly, for what we have called "Skolem-Herbrand expansion"). Shankar's remark is quite correct for the calculus he proposes. So let us see, why it does not apply to the intuitionistic intercalation calculus and its Skolem-Herbrand expansion.

Shankar notes that, when using "Herbrandization" for LJ, unwanted unifications may arise, and result in "proving" statements that are not intuitionistically valid. (See his example on pp. 527–528, i.e., the formula $(\forall x)(\varphi x \vee \psi) \rightarrow \psi \vee (\forall x)\varphi x$ – which is classically, but not intuitionistically provable.) Of course, if one attempts to translate such flawed proofs into the sequent calculus, one does not obtain valid sequent proofs (typically, the restrictions on the quantifier rules are violated). In classical logic one can manage to get valid proofs by permuting the applications of certain rules; but these permutations may be disallowed in the intuitionistic case.

The Skolem-Herbrand expansion of the intercalation calculus has an important different feature, however: the introduction of new function symbols in the rules $\exists \downarrow$ and $\forall \uparrow$. The effect of these new function symbols is, indeed, to exclude unwanted unifications. It is easy to check, for example, that the flawed proof of the formula $(\forall x)(\varphi x \vee \psi) \rightarrow \psi \vee (\forall x)\varphi x$ described in [13] cannot arise in the Skolem-Herbrand expansion of the intuitionistic intercalation calculus.

Complications that arise for proof search in intuitionistic logic are also discussed in Wallen's book [20]. In Chap. 1, §4, Wallen points to three kinds of "redundancies" in the search space of the intuitionistic sequent calculus, namely

to the *non-permutability* of some inferences (we discussed that already), to *notational redundancy* (the same piece of information can occur repeatedly in the search space), and to *irrelevance* (some of the information may be useless for finding a proof). The last two problems clearly affect also the proof search via the ic-calculus, as the full ic-tree may contain the same piece of information on many of its branches, and "irrelevant" premisses, for example, can become a problem for the efficiency of the search algorithm. An algorithm can deal with notational redundancy, at least partially, by storing information in order to avoid answering the same question more than once: that can be done both for positive answers ("a proof has been found") and negative answers ("all proof attempts failed"). The problem of irrelevance can be addressed by exploiting strong syntactic connections between assumptions and goal: that is more intricate in intuionistic than in classical logic, and is reflected already at the sentential level. One may recall that, in terms of computational complexity, the set of classically provable sentential formulas is in Co-NP, while the set of intuitionistically provable sentential formulas is PSPACE-complete [18]. Let us illustrate some of these complications.

As a first example, we consider a remark of Dyckhoff [7]. In examining the intuitionistic sentential sequent calculus, Dyckhoff notes that the rule for conditional introduction on the left gives rise to the problem of detecting loops, and writes: "We could, following standard practice, use a stack to detect looping – but the looping tests are expensive, and complicate the task of extending the technique to the first-order case" ([7], p. 796). Such a remark applies to the intercalation calculus even more strongly, a looping is the condition for closing a branch of an intercalation tree with **N**: the detection of loopings cannot be avoided. This means that we really have to find good search heuristics, if we want to improve the efficiency of the algorithm.

So let us turn to the case of classical sentential logic and the strategic considerations underlying the implementation of an algorithm based on the ic-calculus (in the Carnegie Mellon Proof Tutor)[6]. When faced with a question $\alpha; \beta?G$, this algorithm can form three different kinds of strategies: extraction strategies (if G is a positive subformula of some formula in $\alpha\beta$), inversion strategies (if G is not atomic), indirect strategies. Then the strategies are ranked: the first two, when available, are preferred to the third one, with the exception of the cases when the goal is an atom, a negation or a disjunction (this is heuristically motivated, by the fact that in many common problems the indirect rules must indeed be used to prove an atom, negation or disjunction). So we ask, how these considerations have to be modified for the intuitionistic case (i.e. where the differences with the classical case actually lie). The extraction and inversion strategies can be formed here as well, and the treatment is exactly the same as in the classical case, except for atoms and disjunctions (recall that we consider $\neg$ a defined connective). As the indirect strategy is not available, the necessary changes concern essentially $\bot$ and $\vee$.

[6] Complementary considerations underly the algorithm MAMBA in Tennant's book [19], pp. 136–140.

For $\bot$ the situation is quite simple, since the only rule we have for it is $\bot_{\mathbf{q}}$. This rule has nothing to do with the shape of the goal (provided it is not $\bot$), and is needed to make sure that anything can be proved from an inconsistent set of assumptions. It is clear that a strategy based on this rule should be tried only as a last resort, that is, if all other strategies have failed. This could be the case, for instance, if the goal G is an atom and is not a positive subformula of one of the assumptions: in fact, in this case we can use neither the extraction nor the inversion strategy, and thus our only hope to get G is finding an inconsistency in the assumptions.

The issue about disjunction is more complex. Yet, in some particular situations, the features of intuitionistic logic can help. This is the case when the goal G is a disjunction $\varphi \vee \psi$, and all assumptions are *Harrop formulas* (i.e. formulas which do not contain disjunctions as strictly positive subformulas). In fact, it follows from a theorem of Harrop [11] that in such a case G is an intuitionistic consequence of the assumptions if and only if at least one among φ and ψ is. This means that, in such a situation, one has to pursue the inversion strategy (even if the goal is a disjunction!) before trying something else.

What if there are non-Harrop formulas among the assumptions? In that case, the presence of disjunctions among the assumptions (or the possibility of extracting them) allows the use of the rule $\vee\downarrow$. Now this rule can play, in some sense, the part of the indirect rules in the classical case: it may make sense to try it before other strategies, if the goal is a disjunction. This is due to the fact that many common problems actually need to use of $\vee\downarrow$ to prove a disjunction (think of the the commutative law for $\vee$, i.e. $\varphi \vee \psi; ?\psi \vee \varphi$). Yet one has to be careful: while it seems not to be too expensive to use the indirect strategy in the classical case (one just has to examine a few contradictory pairs), here the amount of computation may become difficult to deal with, especially if several disjunctions are present. The situation is complicated further by the fact that, in certain cases, the $\vee\downarrow$-strategy would be preferrable even if the goal is a not a disjunction (think of $(\varphi \wedge \psi) \vee (\varphi \wedge \psi); ?\varphi \wedge \psi$).

These examples suggest that we have to include, in our heuristics, looking for connections between the goal and disjunction(s) we have in the assumptions (say, check whether one of the two is a positive subformula of the other, whether they have sentential variables or even a disjunct in common). This, though, is not enough. In fact, there are cases in which $\vee\downarrow$ is needed even if the shape of the goal has nothing to do with that of the disjunction in the assumptions: for instance, $\varphi \vee \psi, \varphi \to \chi, \psi \to \chi; ?\chi$. Similarly, one may consider questions like $\varphi_1 \vee \psi_1, \varphi_2 \vee \psi_2, \varphi_1 \to \chi, \psi_1 \to \chi; ?\chi$: here, clearly, the first disjunction is helpful for proving the goal, while the second is not. These examples show the $\vee\downarrow$-strategy and the extraction strategy have to be somehow combined: while forming extraction strategies for the goal, one should also check whether the open questions met have some relationship with a disjunction that occurs as a positive subformula of one of the assumptions, in order to use $\vee\downarrow$ at the appropriate time.

Summing up this sketch of heuristic considerations, we can conclude that forming strategies in the intuitionistic (sentential) case is almost straightforward: one can form the extraction strategies (when the goal is a positive subformula of one of the assumptions), the inversion strategies (if the goal is not an atom) and the $\vee\downarrow$-strategies (if there are non-Harrop formulas among the assumptions), leaving the $\perp_{\mathbf{q}}$-strategies as a last resort. But complications arise for a good ranking of the strategies, when non-Harrop assumptions occur. (These complications provide a heuristic explanation of the higher computational complexity of the set of intuitionistically provable sentential formulas.) Finally, the Skolem-Herbrand expansion is our tool to extend strategies (particularly, the extraction strategies) to the case of predicate logic.

References

1. A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding Modal Logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, 1998.
2. D. Basin, S. Matthews, and L. Viganò. Natural Deduction for Non-Classical Logics. *Studia Logica*, 60(1):119–160, 1998.
3. J. Byrnes. *Proof Search and Normal Forms for Natural Deductions*. PhD thesis, Carnegie Mellon University, 1999.
4. S. Cittadini. Intercalation calculus for intuitionistic propositional logic. Technical Report PHIL-29, Philosophy, Methodology, and Logic, Carnegie Mellon University, 1992.
5. S. Cittadini. Deduzione Naturale e Ricerca Automatica di Dimostrazioni in varie logiche: il Calcolo delle Intercalazioni. In C. Cellucci, M. C. Di Maio, and G. Roncaglia, editors, *Atti del Congresso* Logica e filosofia della scienza: problemi e prospettive*, Lucca, 7–10 gennaio 1993*, pages 583–591, Pisa, 1994. Edizioni ETS.
6. D. van Dalen. *Logic and Structure*. Springer-Verlag, Berlin-Heidelberg-New York, second edition, 1989.
7. R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *The Journal of Symbolic Logic*, 57(3):795–807, 1992.
8. M. Fitting. *Intuitionistic Logic Model Theory and Forcing*. North-Holland Publishing Company, Amsterdam-London, 1969.
9. M. Fitting. *Proof Methods for Modal and Intuitionistic Logic*. D. Reidel Publishing Company, Dordrecht/Boston/Lancaster, 1983.
10. J. Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, Cambridge, 1989.
11. R. Harrop. Concerning formulas of the types $A \to B \vee C$, $A \to (Ex)B(x)$ in intuitionistic formal systems. *The Journal of Symbolic Logic*, 25(1):27–32, 1960.
12. D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almqvist & Wiskell, Stockholm, 1965.
13. N. Shankar. Proof search in the intuitionistic sequent calculus. In D. Kapur, editor, *Automated Deduction: CADE-11, vol. 607 of LNCS*, pages 522–536, Berlin, 1992. Springer-Verlag.
14. W. Sieg. Mechanisms and Search: Aspects of Proof Theory. AILA Preprint 19, Associazione Italiana di Logica e sue Applicazioni, Padova, 1992.
15. W. Sieg and J. Byrnes. Normal Natural Deduction Proofs (in classical logic). *Studia Logica*, 60(1):67–106, 1998.

16. W. Sieg and S. Cittadini. Normal Natural Deduction Proofs (in non-classical logics). Rapporto Matematico n. 351, Dipartimento di Matematica, Università di Siena, 1998.
17. W. Sieg and R. Scheines. Searching for proofs (in sentential logic). In L. Burkholder, editor, *Philosophy and the Computer*, pages 137–159. Westview Press, Boulder, San Francisco, Oxford, 1992.
18. R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9(1):67–72, 1979.
19. N. Tennant. *Autologic*. Edinburgh University Press, Edinburgh, 1992.
20. L. A. Wallen. *Automated Proof Search in Non-Classical Logics*. MIT Press, Cambridge, Mass., 1990.

History and Future of Implicit and Inductionless Induction: Beware the Old Jade and the Zombie!

Claus-Peter Wirth

Dept. of Computer Science, Universität des Saarlandes,
D–66123 Saarbrücken, Germany
cp@ags.uni-sb.de

Abstract. In this survey on implicit induction I recollect some memories on the history of implicit induction as it is relevant for future research on computer-assisted theorem proving, esp. memories that significantly differ from the presentation in a recent handbook article on "inductionless induction". Moreover, the important references excluded there are provided here. In order to clear the fog a little, there is a short introduction to inductive theorem proving and a discussion of connotations of implicit induction like "*descente infinie*", "inductionless induction", "proof by consistency", implicit induction orderings (term orderings), and refutational completeness.

1 What Is Inductive Theorem Proving (*ITP*)?

Inductive reasoning can be seen as extending deductive reasoning in that infinite deductive proofs may be represented in a finite cyclic form, as suggested in the following example, where $\Gamma(x_0, y)$ is a proposition over the natural numbers, where 's' denotes the successor function ($x \mapsto x{+}1$), and where the formulas below each line (sub-goals) imply the formula (goal) above: An infinite deductive proof of $\Gamma(x_0, y)$

$$\dfrac{\Gamma(x_0,y)}{\dfrac{\Gamma(0,y)}{\vdots} \quad \dfrac{\Gamma(\mathsf{s}(x_1),y)}{\dfrac{\Gamma(\mathsf{s}(0),y)}{\vdots} \quad \dfrac{\Gamma(\mathsf{s}(\mathsf{s}(x_2)),y)}{\dfrac{\Gamma(\mathsf{s}(\mathsf{s}(0)),y)}{\vdots} \quad \ddots} \quad \ddots} \quad \ddots}$$

(using $x_i = 0 \vee \exists x_{i+1}.\ x_i = \mathsf{s}(x_{i+1})$) should be captured in something like

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 192–203, 2005.

$$\dfrac{\dfrac{}{\Gamma(0,y)} \qquad \dfrac{\dfrac{}{\Gamma(x_0,y)}}{\Gamma(\mathsf{s}(x_1),y)}}{\vdots} \qquad \dfrac{\dfrac{}{\Delta \Rightarrow \Gamma(\mathsf{s}(x_1),y)}}{\vdots} \qquad \dfrac{\dfrac{}{\Pi \Rightarrow \Gamma(\mathsf{s}(x_1),y)}}{\vdots}$$

$$\dfrac{\Delta' \vee \Gamma(x_1,s)}{\text{(back to top)}} \qquad \dfrac{\Pi' \vee \Gamma(x_1,t)}{\text{(back to top)}} \qquad \dfrac{\Pi''}{\vdots}$$

using $\Delta \vee \Pi$. This kind of *cyclic* argument – namely inferring $\Gamma(x_1,s)$ and $\Gamma(x_1,t)$ from $\Gamma(x_0,y)$ – is sound if for each (ground) instantiation of the theorem (here: $\Gamma(\mathsf{s}^m(\mathsf{0}),\mathsf{s}^n(\mathsf{0}))$) the deductive proof terminates. This can be guaranteed by requiring that each cycle in the proof (graph) *terminate*, i.e. its preconditions (usually called *induction hypotheses* – here: $\Gamma(x_1,s)$ and $\Gamma(x_1,t)$) be smaller than the *"induction" conclusion* (here: $\Gamma(x_0,y)$ or $\Gamma(\mathsf{s}(x_1),y)$) w.r.t. some wellfounded ordering, called *induction ordering* (here e.g. the usual ordering on the natural numbers applied to the first argument of Γ). Thus, while the property of being an inductive theorem depends only on the specification (i.e. a language and a set of axioms) and the choice of a specific notion of inductive validity, cf. [74], an inductive proof of an inductive theorem also depends on an additional parameter, namely some induction ordering which must be chosen appropriately during the proof.

2 Explicit Versus Implicit Induction

Although there is no generally accepted characterization of the two paradigms of *explicit* and *implicit* induction in the research community, in [67], which is a comprehensive survey on explicit induction, the following is said:

> Research on automated induction these days is based on two competing paradigms: *Implicit induction* (also termed *inductive completion*, *inductionless induction*, or, less confusingly, *proof by consistency*) evolved from the *Knuth–Bendix Completion Procedure* .... The other research paradigm ... is called *explicit induction* and resembles the more familiar idea of induction theorem proving using induction axioms.

In accordance with this view, one reason to call the latter paradigm "*explicit*" is that in the underlying inference systems every cyclic argument must be made explicit in a single inference step applying a so-called *induction rule*. Besides generating *induction base* formulas, this step joins induction hypotheses and conclusions in *induction step* formulas and explicitly guarantees the termination of their cycles by a sub-proof or -mechanism for the wellfoundedness of the induction ordering resulting from the step formulas. In the explicit induction proof corresponding to the example induction proof of Section 1 the induction base formula is $\Gamma(0,y)$ and the induction step formulas are $\Delta \Rightarrow (\Gamma(x_1,s) \Rightarrow \Gamma(\mathsf{s}(x_1),y))$ and $\Pi \Rightarrow (\Gamma(x_1,t) \Rightarrow \Gamma(\mathsf{s}(x_1),y))$. The explicit induction proof then has the following form:

$$\dfrac{\dfrac{}{\Gamma(0,y)} \qquad \dfrac{\Delta \Rightarrow (\ \Gamma(x_1,s) \Rightarrow \Gamma(\mathsf{s}(x_1),y)\ )}{\Gamma(x_0,y)} \qquad \dfrac{\Pi \Rightarrow (\ \Gamma(x_1,t) \Rightarrow \Gamma(\mathsf{s}(x_1),y)\ )}{\Gamma(x_0,y)}}{\Gamma(x_0,y)}$$

Note that the first inference in this proof is an application of an *induction axiom* in the sense of [67].

As the example induction proof of Section 1 illustrates, the cyclic arguments and their termination in *implicit* induction proofs need not be confined to single inference steps, as is in explicit induction. Therefore, the induction axioms corresponding to the cyclic arguments in a finite implicit induction proof can only be determined by analyzing the whole proof, whereas in the case of explicit induction each applied induction axiom is given by a single application of the induction rule.

The phrase "more familiar idea" attributed to explicit induction in the above quotation requires some remarks. Implicit induction in the style of *descente infinie* was already known to the ancient Greeks and – as will be explained below – is the standard method of mathematical induction since Pierre Fermat (1607?–1665) rediscovered the method and named it *descente infinie* (*ou indéfinie*). Nevertheless more familiar is the idea of explicit induction to computer scientists who – inspired by J. Alan Robinson's resolution method of the year 1963 – wanted to solve all problems of logical inference via reduction to machine-oriented inference systems. Instead of implementing mathematical ITP, they decided to reduce it via the induction rule (cf. above) in the following fashion: Apply the induction rule backwards; then do purely deductive reasoning; if this fails, repeat the process! The so called "waterfall" of [15] refines this process into a fascinating heuristic and the ITP system NQTHM [15, 16] shows that in this case the reduction approach was quite successful – so successful actually that it is hard to understand why implicit induction was able to gain ground again, as we will see in the next section.

3 1988 – Start of Practical Interest in Implicit Induction

Cf. e.g. [62, 19] for historical surveys on implicit induction, which, however, have to be taken with a grain of salt. After several papers on implicit induction in purely equational theories already in the year 1980, [32, 36, 47, 51], there was a sequence of papers on technical improvements, [37, 23, 30, 44], which was topped by Leo Bachmair's paper [6] in the year 1988. Up to 1987 the papers on implicit induction had a theoretical nature, but some of them were very inspiring, [38] being my favorite.

Bachmair's paper [6] is the first one that gave good reason for a hope to develop the method into practical usefulness. And – in 1990 – there was real hope, cf. [34, p. 1]:

> This approach has some very attractive and promising characteristics compared to classical approaches for explicit inductive theorem proving based on induction schemas, which will be pointed out later on.

As a consequence, in the end of the 1980s and the first half of the 1990s several researchers tried to clearly understand what "*implicit induction*" means from a practical point of view and whether it would be useful for practical ITP.

4 1996 – End of Most Interest in Implicit Induction

During the Induction Workshop on the 13th Int. Conf. on Automated Deduction, New Brunswick (NJ), 1996, there was an agreement on better not to use the term "implicit induction" in the future, for the following two reasons, which will be discussed in the next two subsections.

1. Different researchers understand this term differently.
2. The notion has lost its potential relevance for the practice of ITP.

4.1 What Is *Implicit* Induction?

While it is generally accepted that [6] is implicit and [15] is explicit, we report the following different views on what it is that makes induction implicit:

Descente Infinie. In explicit induction there is something like an "induction rule" (cf. Section 2) whose addition turns a deductive inference system into an inductive inference system without further changes on the deductive part. Concerning the concept of induction hypothesis, the explicit induction "hides" several (applications of) induction hypotheses in a single inference step. To the contrary, in *descente infinie*, the inference system "knows" what an induction hypothesis is, i.e. it includes inference rules that provide or apply induction hypotheses, given that certain ordering conditions resulting from these applications can be met by an induction ordering.
Note that *descente infinie* is important for human-oriented ITP because this is the style in which working mathematicians do induction since Pierre Fermat (1607?–1665) rediscovered and named the method which was already known to the ancient Greeks. The working mathematician applies it in the following fashion.

> He starts with the conjecture and simplifies it by case analysis. When he realizes that the current goal becomes similar to an instance of the conjecture, he applies the instantiated conjecture just like a lemma, but keeps in mind that he has actually applied an induction hypothesis. Finally, he searches for some wellfounded ordering in which all the instances of the conjecture he has applied as an induction hypothesis are smaller than the original conjecture itself.

This view on the notion of "implicit induction" (i.e. *descente infinie*) was the one of the majority on the Induction Workshop in 1996.
The name *descente infinie* for this aspect of implicit induction was coined later in [71] after suggestions and complaints on the occurrence of "implicit induction" in the title of [70]. Researchers introduced to *descente infinie* by [57] (entitled "Lazy Generation of Induction Hypotheses") sometimes speak of "*lazy induction*" instead of *descente infinie*.

Inductionless Induction/Proof by Consistency. "Inductionless Induction" means that no induction can be observed explicitly. E.g. some Knuth–Bendix [35] or superposition calculus [25] completion procedure produces a huge number of irrelevant inferences under which the ones relevant for induction can hardly be made explicit in an automatic way. "Inductionless induction" has shown to be practically useless, mainly due to too many superfluous inferences, typically infinite runs, and too restrictive admissibility conditions. The approach in [25] is an interesting theoretical possibility but definitely useless for practical purposes. Roughly speaking, the conceptual flaw in "inductionless induction" seems to be that, instead of finding a sufficient set of reasonable inferences, the research follows the paradigm of ruling out as many irrelevant inferences as possible. A proof attempt is successful when the prover has drawn all necessary inferences and stops without having detected an inconsistency (empty clause).

Christoph Walther (cf. [67]) prefers to use the name "*proof by consistency*" instead of the *contradictio in eo ipso* "inductionless induction", which, however, is used again in the year 2001 in the title of [19]. Indeed, "proof by consistency" seems to be a better name for this aspect of implicit induction than "inductionless induction" because the former highlights on the fact that – roughly speaking – the derivation of the empty clause (or inconsistency) means disproof, not proof. E.g., in the case of the toy example proofs in [20] the number of irrelevant inferences is cut down to zero (unless they are hidden in the omitted parts of the proofs) and it is quite obvious where the induction takes place, so that "proof by consistency" is still appropriate, but "inductionless induction" is not.

While the opinion that the "inductionless induction" aspect (i.e. no induction explicitly observable) is the crucial one for "implicit induction" is now shared by several researchers who had worked on implicit induction in the past, at the Induction Workshop in 1996, only a minority held this opinion. Note that this view on the notion of implicit induction is held by Martin Protzen and places his important work [57] (which is implicit induction in the sense of *descente infinie*) on the side of explicit induction, where the practically useful ITP systems use to dwell, such as the powerful up-to-date systems INKA [3] and ACL2 [41].

The name "proof by consistency" was coined in [39], which is the forerunner of the earlier published improved version [38]. The name "inductionless induction" was coined already before [48] (to which it is erroneously attributed in [19, 62]), namely in [47] in the year 1980, when also the title of [32] included a similar phrase.

Implicit Induction Ordering. This means that there is no explicit induction ordering in the signature and the model. Instead of such semantical orderings, the induction is performed on syntactical term orderings that are not part of the logic language. The semantical orderings (cf. Definition 13.7 of [69]) cannot depend on the syntactical term structure of a weight w but only on the value of w under the evaluation function. In [69] we have rigorously investigated the price one has to pay for the possibility to have

induction orderings also depending on the syntax of weights. For powerful concrete inference systems this price turned out to be surprisingly high. Besides this, after improving the ordering information in *descente infinie* by our introduction of explicit weights (cf. [72]) the former necessity of sophisticated induction orderings that exploit the term structure (cf. e.g. [6, 64]) does not seem to exist anymore.
This view on the notion of "implicit induction" (i.e. no explicit induction ordering in the language) is held by a minority, e.g. by Peter Padawitz. Note that his view places his ITP system EXPANDER, [54, 55], on the explicit side, where the practically useful ITP systems use to dwell. Note that EXPANDER is also explicit in the sense that it does not perform "inductionless induction". EXPANDER is, however, implicit in the sense that it realizes *descente infinie*, and – to my knowledge – it is the only integration of *descente infinie* into a framework of refutational resolution (or – more precisely – inverse method, [49]) where – roughly speaking – the empty clause (or inconsistency) means proof, not disproof.
Moreover, this view on the notion of "implicit induction" (i.e. implicit induction ordering) is also the view found in [19], which, however, concentrates on "inductionless induction".

It does not seem to be completely superfluous to note that "*refutational completeness*" in itself cannot be crucial for implicitness of induction, simply because it is not a very important property:

- For practical purposes, it is not important that the invalidity of a theorem would be detected sometime (= refutational completeness), but that it is detected efficiently, cf. [56].
- In theoretical terms, refutational completeness used to be trivial for the very restricted logics in the scope of implicit induction.

Nevertheless, since Leo Bachmair's important paper [6] various authors in the field of implicit induction emphasized refutational completeness as if it were a major asset. So let us have a final look at it:

In general, for all significant notions of inductive validity, the set of inductively valid theorems is not enumerable, cf. [31, 52, 50]. Therefore, refutational completeness is an optimal theoretical quality of inference systems for ITP. In practice, however, refutational completeness by itself does not help in refuting invalid conjectures or in finding finite proofs for inductively valid formulas. Only theoreticians completely detached from reality can consider nonterminating proof attempts in refutationally complete inference systems to be successful proofs.

4.2 End of the Schism

In 1996, the theoretical aspects of "inductionless induction" were clearly understood by the experts. Moreover, the severe limitations of "inductionless induction" were admitted by all researchers in the field, at least by those who attended

the Induction Workshop. There was the general opinion that "inductionless induction" was *dead*.

The practitioners in automated theorem proving were also not too interested in the remaining aspects of implicit induction, i.e. "*descente infinie*" and "implicit induction orderings". My main interest, however, was and still is to convince the community of the practical importance of "*descente infinie*", cf. [71].

All participants agreed that the general challenge would be the development of practically more useful ITP systems where the separation of implicit vs. explicit would not play a splitting role because (contrary to "inductionless induction") "*descente infinie*" goes together well with the standard ideas of explicit induction. The general opinion was – roughly speaking – the following.

> To succeed in proving an inductive theorem in finite time, implicit inductive theorem provers have to solve the same problems as explicit inductive theorem provers, namely to find a finite cyclic representation for an infinite deductive proof as well as an induction ordering guaranteeing the termination of its cycles.

Thus, the hatchet between the tribe of explicit ITP and the (rather small) tribe of implicit ITP was buried at the Induction Workshop in 1996, at least between the attendees.

5 The Aftermath

Since the year 1996 there was only a small number of publications on implicit induction. The following seems to be a complete list of the research papers: "Inductionless induction" is only found in [20]. Besides this, *descente infinie* is treated in [10, 61, 69, 70, 45, 71, 65, 2]. Implicit induction orderings are found only in [10, 69, 20, 65, 2].

Recently, in the Handbook of Automated Reasoning [60], Hubert Comon published an article [19] on "inductionless induction" that gave a very biased account on the history of "implicit induction". It seems to me that Hubert Comon neglects the more practice-oriented research on implicit induction because this would invalidate his resurrection of "inductionless induction" in [20] as of merely theoretical interest. I think this to be very problematic because this was already misleading in the past (cf. [62], where [20] is mistaken to be "the cutting edge", [63]) and probably will further mislead newcomers who take the handbook [60] for a complete high-quality reference for entering a research area. Therefore, I would like to give the advice:

> *Do not to fumble around with the zombie of "inductionless induction"!* The experts in implicit induction have spent a lot of time with it, mostly to bury it.

Less problematic than the biased handbook article [19] is the technical report [20], although it does not represent a significant practical progress as compared to [25]. It is more of theoretical interest. Moreover, sentences like "The method of proofs by consistency has lost part of its popularity since it

was developed in the early 80s." are misleading because the method was never popular, not even among researchers in ITP, not in the 1980s, not even in the the early 1990s.

6 Conclusion

While there was reason to hope to develop "inductionless induction" into practical applicability in 1988, it is a theoretician's game for more than half a decade now. Also "implicit induction orderings" do not seem to be useful in practice.

The remaining aspect of implicit induction, however, "*descente infinie*", is going to play an important role in the practice of ITP when mathematical assistant systems are applied to proof problems that are beyond the scope of the recursion analysis and the eager hypothesis generation of explicit ITP.

7 A Word to the Wise

In communications on previous versions of this paper I was surprised that the confusion on the meaning of implicitness of induction is just as total today as it was in 1996; only the confidence in the justifiedness of the respective personal views seems to have increased with the time gone by.

A sinner myself, cf. [70], regarding future research, I therefore propose to finally bury the phrases "implicit induction" and "inductionless induction".

The notions of "*descente infinie*", "proof by consistency", and "implicit induction ordering" are the relays that can take us further than the old jade "implicit induction" and the zombie of "inductionless induction".

Caveat

This article depends on my personal possibly erroneous (but careful and honest) judgments and memories gathered over a dozen years of work on this very complex subject.

References

The following list contains publications on implicit induction with a special emphasis on those that should have been but are not cited in [19]. Additionally, it contains some outstanding or important papers on other kinds and aspects of induction and theorem proving.

[1] H. Ait-Kaci, M. Nivat (eds.) (1989). *Resolution of Equations in Algebraic Structures.* Academic Press.

[2] Alessandro Armando, Michaël Rusinowitch, Sorin Stratulat (2002). *Incorporating Decision Procedures in Implicit Induction.* J. Symbolic Computation **34**, pp. 241–258, Academic Press.

[3] Serge Autexier, Dieter Hutter, Heiko Mantel, Axel Schairer (1999). INKA *5.0 – A Logical Voyager.* 16th CADE 1999, LNAI 1632, pp. 207–211, Springer.
[4] Jürgen Avenhaus, Klaus Madlener (1995). *Theorem Proving in Hierarchical Clausal Specifications.* SEKI-Report SR–95–14 (SFB), Univ. Kaiserslautern. `http://www-madlener.informatik.uni-kl.de/seki/1995/Avenhaus.SR-95-14.ps.gz (March 07, 2002).`
[5] Matthias Baaz, Uwe Egly, Christian G. Fermüller (1997). *Lean Induction Principles for Tableaus.* 6th TABLEAU 1997, LNAI 1227, pp. 62–75, Springer.
[6] Leo Bachmair (1988). *Proof By Consistency in Equational Theories.* 3rd IEEE symposium on Logic In Computer Sci., pp. 228–233, IEEE Press.
[7] Leo Bachmair (1991). *Proof By Consistency.* Birkhauser, Boston.
[8] Klaus Becker (1993). *Proving Ground Confluence and Inductive Validity in Constructor-Based Equational Specifications.* TAPSOFT 1993, LNCS 668, pp. 46–60, Springer.
[9] Klaus Becker (1994). *Rewrite Operationalization of Clausal Specifications with Predefined Structures.* Ph.D. thesis, FB Informatik, Univ. Kaiserslautern.
[10] Klaus Becker (1996). *How to Prove Ground Confluence.* SEKI-Report SR–96–02, Univ. Kaiserslautern.
[11] Rudolf Berghammer (1993). *On the Characterization of the Integers: The Hidden Function Problem Revisited.* Acta Cybernetica **11**, pp. 85–96, Szeged.
[12] Eddy Bevers, Johan Lewi (1990). *Proof by Consistency in Conditional Equational Theories.* Report CW 102, rev. July 1990. Dept. Comp. Sci., K. U. Leuven. Short version in: 2nd CTRS 1990, LNCS 516, pp. 194–205, Springer.
[13] Adel Bouhoula, Michaël Rusinowitch (1995). *Implicit Induction in Conditional Theories.* J. Automated Reasoning **14**, pp. 189–235, Kluwer.
[14] Adel Bouhoula, Emmanuël Kounalis, Michaël Rusinowitch (1992). *Automated Mathematical Induction.* Technical Report 1636, INRIA.
[15] Robert S. Boyer, J Strother Moore (1979). *A Computational Logic.* Academic Press.
[16] Robert S. Boyer, J Strother Moore (1988). *A Computational Logic Handbook.* Academic Press.
[17] Robert S. Boyer, J Strother Moore (1989). *The Addition of Bounded Quantification and Partial Functions to A Computational Logic and Its Theorem Prover.* In: Manfred Broy (ed.), Constructive Methods in Computing Science, NATO ASI Series, Vol. F 55, pp. 95–145, Springer.
[18] Alan Bundy (1988). *The Use of Explicit Proof Plans to Guide Inductive Proofs.* 9th CADE 1988, LNAI 310, pp. 111–120, Springer.
[19] Hubert Comon (2001). *Inductionless induction.* In: [60], Vol. I, pp. 913–970.
[20] Hubert Comon, Robert Nieuwenhuis (1998). *Induction = I-Axiomatization + First-order Consistency.* Technical Report, ENS Cachan.
[21] Ulrich Fraus (1993). *A Calculus for Conditional Inductive Theorem Proving.* 3rd CTRS 1992, LNCS 656, pp. 357–362, Springer.
[22] Ulrich Fraus (1994). *Mechanizing Inductive Theorem Proving in Conditional Theories.* Ph.D. thesis, Univ. Passau.
[23] Laurent Fribourg (1986). *A Strong Restriction of the Inductive Completion Procedure.* 13th ICALP 1986, LNCS 226, pp. 105–116, Springer. Also in: J. Symbolic Computation **8**, pp. 253–276, Academic Press, 1989.
[24] Dov M. Gabbay, C. J. Hogger, J. Alan Robinson (eds.) (1993 ff.). *Handbook of Logic in Artificial Intelligence and Logic Programming.* Clarendon Press.

[25] Harald Ganzinger, Jürgen Stuber (1992). *Inductive Theorem Proving by Consistency for First-Order Clauses.* In: Informatik-Festschrift zum 60. Geburtstag von Günter Hotz. pp. 441–462. Teubner Verlag, Stuttgart. Also in: 3rd CTRS 1992, LNCS 656, pp. 226–241, Springer, 1993.

[26] Gerhard Gentzen (1938). *Die gegenwärtige Lage in der mathematischen Grundlagenforschung – Neue Fassung des Widerspruchsfreiheitsbeweises für die reine Zahlentheorie.* Forschungen zur Logik und zur Grundlegung der exakten Wissenschaften, Folge 4, Leipzig.

[27] Gerhard Gentzen (1943). *Beweisbarkeit und Unbeweisbarkeit von Anfangsfällen der transfiniten Induktion in der reinen Zahlentheorie.* Mathematische Annalen **119**, pp. 140–161.

[28] Alfons Geser (1995). *A Principle of Non-Wellfounded Induction.* In: Tiziana Margaria (ed.). Kolloquium Programmiersprachen und Grundlagen der Programmierung, MIP–9519, pp. 117–124, Univ. Passau.

[29] Martin Giese (1998). *Integriertes automatisches und interaktives Beweisen: die Kalkülebene.* Master's thesis, Univ. Karlsruhe.
`http://i11www.ira.uka.de/~giese/da.ps.gz(May 09, 2000)`.

[30] Richard Göbel (1985). *Completion of Globally Finite Term Rewriting Systems for Inductive Proofs.* 9th German Workshop on AI, IFB 118, Springer.

[31] Kurt Gödel (1931). *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.* Monatshefte für Mathematik und Physik **38**, pp. 173–198.

[32] Joseph Goguen (1980). *How to Prove Algebraic Inductive Hypotheses Without Induction.* 5th CADE 1980, LNCS 87, pp. 356–373, Springer.

[33] Bernhard Gramlich (1989). *Inductive Theorem Proving Using Refined Unfailing Completion Techniques.* SEKI-Report SR–89–14 (SFB), Univ. Kaiserslautern. Short version in: 9th ECAI 1990, pp. 314–319, Pitman Publ..

[34] Bernhard Gramlich (1990). *Completion Based Inductive Theorem Proving: A Case Study in Verifying Sorting Algorithms.* SEKI-Report SR–90–04, Univ. Kaiserslautern.

[35] Bernhard Gramlich, Wolfgang Lindner (1991). *A Guide to* UNICOM*, an Inductive Theorem Prover Based on Rewriting and Completion Techniques.* SEKI-Report SR-91–17 (SFB) Univ. Kaiserslautern.
`http://agent.informatik.uni-kl.de/seki/1991/Lindner.SR-91-17.ps.gz (May 09, 2000)`.

[36] Gérard Huet, Jean-Marie Hullot (1980). *Proofs by Induction in Equational Theories with Constructors.* 21st FOCS 1980, pp. 96–107. Also in: J. Computer and System Sci. **25**, pp. 239–266, Academic Press, 1982.

[37] Jean-Pierre Jouannaud, Emmanuël Kounalis (1986). *Automatic Proofs by Induction in Equational Theories Without Constructors.* 1st IEEE symposium on Logic In Computer Sci., pp. 358–366, IEEE Press. Also in: Information and Computation **82**, pp. 1–33, Academic Press, 1989.

[38] Deepak Kapur, David R. Musser (1986). *Inductive Reasoning with Incomplete Specifications.* 1st IEEE symposium on Logic In Computer Sci., pp. 367–377, IEEE Press.

[39] Deepak Kapur, David R. Musser (1987). *Proof by Consistency.* Artificial Intelligence **31**, pp. 125–157.

[40] Deepak Kapur, Hantao Zhang (1989). *An Overview of Rewrite Rule Laboratory (*RRL*).* 3rd RTA 1989, LNCS 355, pp. 559–563, Springer.

[41] Matt Kaufmann, Panagiotis Manolios, J Strother Moore (2000). *Computer-Aided Reasoning: An Approach.* Kluwer.

[42] Emmanuël Kounalis, Michaël Rusinowitch (1990). *Mechanizing Inductive Reasoning.* 8th AAAI 1990, pp. 240–245, MIT Press.

[43] Georg Kreisel (1965). *Mathematical Logic.* In: T. L. Saaty (ed.). Lectures on Modern Mathematics, Vol. III, pp. 95–195, John Wiley & Sons, New York.

[44] Wolfgang Küchlin (1987). *Inductive Completion by Ground Proof Transformation.* Colloquium on Resolution of Equations in Algebraic Structures (CREAS). Also in: [1], Vol. 2, pp. 211–244.

[45] Ulrich Kühler (2000). *A Tactic-Based Inductive Theorem Prover for Data Types with Partial Operations.* Ph.D. thesis, Infix, Sankt Augustin.

[46] Ulrich Kühler, Claus-Peter Wirth (1996). *Conditional Equational Specifications of Data Types with Partial Operations for Inductive Theorem Proving.* SEKI-Report SR–96–11, Univ. Kaiserslautern. Short version in: 8th RTA 1997, LNCS 1232, pp. 38–52, Springer.
`http://ags.uni-sb.de/~cp/p/rta97/welcome.html (Aug. 05, 2001)`.

[47] D. S. Lankford (1980). *Some Remarks on Inductionless Induction.* Memo MTP-11, Math. Dept., Louisiana Tech. Univ., Ruston.

[48] D. S. Lankford (1981). *A Simple Explanation of Inductionless Induction.* Memo MTP-14, Math. Dept., Louisiana Tech. Univ., Ruston.

[49] Vladimir A. Lifschitz (1989). *What Is the Inverse Method?.* J. Automated Reasoning **5**, pp. 1–23, Kluwer.

[50] David B. MacQueen, Donald T. Sannella (1985). *Completeness of Proof Systems for Equational Specifications.* IEEE Transactions on Software Engineering **11**, pp. 454–461, IEEE Press.

[51] David R. Musser (1980). *On Proving Inductive Properties of Abstract Data Types.* 7th POPL 1980, pp. 154–162, ACM Press.

[52] Cyrus F. Nourani (1994). *Types, Induction, and Incompleteness.* Bull. EATCS **53**, pp. 226–247.

[53] Peter Padawitz (1990). *Horn Logic and Rewriting for Functional and Logic Program Design.* MIP–9002, Univ. Passau.

[54] Peter Padawitz (1996). *Inductive Theorem Proving for Design Specifications.* J. Symbolic Computation **21**, pp. 41–99, Academic Press.

[55] Peter Padawitz (1998). EXPANDER. *A System for Testing and Verifying Functional Logic Programs.*
`http://LS5.cs.uni-dortmund.de/~peter/ExpaTex.ps.gz (Sept. 14, 1999)`.

[56] Martin Protzen (1992). *Disproving Conjectures.* 11th CADE 1992, LNAI 607, pp. 340–354, Springer.

[57] Martin Protzen (1994). *Lazy Generation of Induction Hypotheses.* 12th CADE 1994, LNAI 814, pp. 42–56, Springer. Long version in: [58].

[58] Martin Protzen (1995). *Lazy Generation of Induction Hypotheses and Patching Faulty Conjectures.* Ph.D. thesis, Infix, Sankt Augustin.

[59] Uday S. Reddy (1990). *Term Rewriting Induction.* 10th CADE 1990, LNAI 449, pp. 162–177, Springer.

[60] J. Alan Robinson, Andrei Voronkov (eds.) (2001). *Handbook of Automated Reasoning.* Elsevier.

[61] Christof Sprenger (1996). *Über die Beweissteuerung des induktiven Theorembeweisers* QUODLIBET *mit Taktiken.* Master's thesis, FB Informatik, Univ. Kaiserslautern.

[62] Graham Steel (1999). *Inductionless Induction (aka Implicit Induction or Proof by Consistency): A Literature Survey.*
`http://www.dai.ed.ac.uk/homes/grahams/papers/lit-survey.ps.gz (Apr. 28, 2002).`

[63] Graham Steel, Alan Bundy, Ewen Denney (2002). *Using Implicit Induction to Guide a Parallel Search for Inconsistency.*
`http://www.dai.ed.ac.uk/homes/grahams/papers/abstract.pdf (Apr. 28, 2002).`

[64] Joachim Steinbach (1995). *Simplification Orderings – History of Results.* Fundamenta Informaticae **24**, pp. 47–87.

[65] Sorin Stratulat (2001). *A General Framework to Build Contextual Cover Set Induction Provers.* J. Symbolic Computation **32**, pp. 403–445, Academic Press.

[66] Christoph Walther (1992). *Computing Induction Axioms.* 3rd LPAR 1992, LNAI 624, pp. 381–392, Springer.

[67] Christoph Walther (1994). *Mathematical Induction.* In: [24], Vol. 2, pp. 127–228.

[68] Claus-Peter Wirth (1991). *Inductive Theorem Proving in Theories Specified by Positive/Negative-Conditional Equations.* Master's thesis, FB Informatik, Univ. Kaiserslautern. Abstract in: 1st Workshop on Construction of Computational Logics, Val d'Ajol (France), 1992, Rapport Interne CRIN 93–R–023, p. 38, Villers-les-Nancy, 1993.

[69] Claus-Peter Wirth (1997). *Positive/Negative-Conditional Equations: A Constructor-Based Framework for Specification and Inductive Theorem Proving.* Ph.D. thesis, Verlag Dr. Kovač, Hamburg.

[70] Claus-Peter Wirth (1999). *Full First-Order Free Variable Sequents and Tableaus in Implicit Induction.* 8th TABLEAU 1999, LNAI 1617, pp. 293–307, Springer.
`http://ags.uni-sb.de/~cp/p/tab99/welcome.html (Aug. 05, 2001).`

[71] Claus-Peter Wirth (2000). *Descente Infinie + Deduction.* Report 737/2000, FB Informatik, Univ. Dortmund. Extd. version, Feb. 1, 2003
`http://ags.uni-sb.de/~cp/p/tab99/new.html (Feb. 01, 2003).`

[72] Claus-Peter Wirth, Klaus Becker (1995). *Abstract Notions and Inference Systems for Proofs by Mathematical Induction.* 4th CTRS 1994, LNCS 968, pp. 353–373, Springer.
`http://ags.uni-sb.de/~cp/p/ctrs94/welcome.html (Aug. 05, 2001).`

[73] Claus-Peter Wirth, Bernhard Gramlich (1994). *A Constructor-Based Approach for Positive/Negative-Conditional Equational Specifications.* J. Symbolic Computation **17**, pp. 51–90, Academic Press.
`http://ags.uni-sb.de/~cp/p/jsc94/welcome.html (Aug. 05, 2001).`

[74] Claus-Peter Wirth, Bernhard Gramlich (1994). *On Notions of Inductive Validity for First-Order Equational Clauses.* 12th CADE 1994, LNAI 814, pp. 162–176, Springer
`http://ags.uni-sb.de/~cp/p/cade94/welcome.html (Aug. 05, 2001).`

[75] Claus-Peter Wirth, Ulrich Kühler (1995). *Inductive Theorem Proving in Theories Specified by Positive/Negative-Conditional Equations.* SEKI-Report SR–95–15 (SFB), Univ. Kaiserslautern
`http://ags.uni-sb.de/~cp/p/sr9515/welcome.html (Aug. 05, 2001).`

[76] Hantao Zhang, Deepak Kapur, Mukkai S. Krishnamoorthy (1988). *A Mechanizable Induction Principle for Equational Specifications.* 9th CADE 1988, LNAI 310, pp. 162–181, Springer.

The Flowering of Automated Reasoning*

Larry Wos

Mathematics and Computer Science Division,
Argonne National Laboratory,
Argonne, IL 60439-4801
wos@mcs.anl.gov

Abstract. This article celebrates with obvious joy the role automated reasoning now plays for mathematics and logic. Simultaneously, this article evidences the realization of a dream thought impossible just four decades ago by almost all. But there were believers, including Joerg Siekmann to whom this article is dedicated in honor of his sixtieth birthday. Indeed, today (in the year 2001) a researcher can enlist the aid of an automated reasoning program often with the reward of a new proof or a better proof in some significant aspect. The contributions to mathematics and logic made with an automated reasoning assistant are many, diverse, often significant, and of the type Hilbert would indeed have found most pleasurable. The proofs discovered by W. McCune's OTTER (as well as other programs) are Hilbert-style axiomatic. Further, some of them address Hilbert's twenty-fourth problem (recently unearthed by Rudiger Thiele), which focuses on the completion of simpler proofs. In that regard, as well as others, I offer challenges and open questions, frequently providing appropriate clauses to provide a beginning.

1 From Minuscule to Monumental

The style of this article is narrative, interweaving elements of history with developments in automated reasoning that presage a stirring and rewarding future. Here one meets a new generation of researchers in the field and learns of incarnations of Hilbert's recently discovered (by Rudiger Thiele) twenty-fourth problem. By presenting diverse results from a wide spectrum focusing on mathematics and logic, the explicit intent is to interest new researchers, as well as the experienced, in experimentation and application. To further that objective, this article includes very short stories, featuring research nuggets, methodologies, and strategies to extend, modify, and experiment with. The number of topics covered and the length of this paper pay tribute to the long, long career of Joerg Siekmann. Although this article in no way comes close to being a thorough survey, it does offer some startling successes that would almost certainly have been out of reach were it not for the reliance on an automated reasoning program.

* This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 204–227, 2005.

I choose to mark the birth of the field now known to many as automated reasoning with J. A. Robinson's introduction of binary resolution. Of course, earlier work on geometry and logic in the context of proving theorems was indeed important, but the era of concern here begins with Robinson's contribution. The inference rule binary resolution beautifully generalizes both modus ponens and syllogism. I prefer (because of questions of effective encoding) the definition of binary resolution that focuses on a single literal in each of two premisses and, therefore, prefer the inclusion of the inference rule factoring as well.

The cited rule of reasoning is strongly connected with a particular language. Indeed, a simply brilliant stroke was the choice of the clause language for representing questions and problems to the type of reasoning program that would evolve. Although on the surface one might find this language quite poor in comparison with natural language, its poverty actually facilitates the formulation of effective means for conclusion drawing and, perhaps of greater importance, facilitates the formulation of powerful strategies to control the reasoning. Without strategy, it seems to me, what I discuss shortly as successes would most likely not have occurred.

Almost at the beginning, when the Argonne paradigm (with its heavy emphasis on the use of strategy and on experimentation) was in its infancy, a key test problem was a five-clause example, the Davis-Putnam problem. Finding a proof for this inconsistent set of clauses by a computer program in negligible time marked an achievement but – especially in retrospect – a minuscule one that in no way foretold what was to come.

A bit more significant was our success with a classroom exercise in group theory. The theorem (actually too strong a classification) asks for a proof of commutativity for groups of exponent 2 (those in which the square of x is the identity e). Its proof was completed in less than 3 CPU-seconds on what today would be called a very, very slow machine. What did count was the first instance of what would become the Argonne paradigm. In that paradigm, the use of strategy is indispensable, clause retention plays a key role, and experimentation is most heavily emphasized, where the targets are actual results from mathematics and logic as opposed to syntactic problems. When the question is open, far better. Regarding the retention of newly deduced information, sometimes the sought-after proof is discovered after the keeping of more than 1,000,000 clauses and the generation of tens of millions. As for the occasionally-voiced objection that an unaided master would not proceed in such a manner, note that the past few years have witnessed the discovery of proofs that had eluded experts for decades; see Section 4.

This section has, till now, focused on the minuscule; next in order is the monumental, a classification virtually demanding some justification and explanation before evidence is presented. In both mathematics and logic, at the simplest level, advances in the context of proof can be partitioned into first proofs and improved proofs. A beautiful example of the first class was the establishment by an automated reasoning program that every Robbins algebra is a Boolean algebra [McCune1997]. A charming example of the second class was the discovery

(by a different automated reasoning program) of a proof focusing on Meredith's 21-letter formula for two-valued sentential (or propositional) calculus [Meredith1953], the following expressed in clause notation.

```
P(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
```

The proof of concern shows that this formula is a single axiom, but, more important, the proof consists of 38 applications of condensed detachment in contrast to the (in effect) 41-step proof of Meredith himself. I shall return to these two examples, as well as others, when I present in Section 4 evidence that "monumental" is well deserved.

Of a different nature, and perhaps more significant, is that element focusing on the source of the implied question or that on the expert who has shown interest in such a question. No less a person than Alfred Tarski devoted energy to the Robbins algebra question, offering it in a book [Henkin1971], having his students attempt to answer it, and posing it repeatedly to researchers. Regarding Meredith, he clearly was concerned with proof length, and he was a master at such simplification, in part attested to by his publishing (with Prior) an abridgment of a result of Lukasiewicz. But I can offer (to many) a far more persuasive view, justification, and explanation for employing the term "monumental" – indeed, Hilbert himself enters the picture.

In his 1900 Paris talk, Hilbert offered twenty-three problems that have had and continue to have a profound effect on mathematics. This greatest mathematician of the twentieth century also had a dramatic effect on logic. His emphasis on axiomatic proofs is but one example. The type of program I prefer (of which OTTER is the finest example in my view) produces Hilbert-style axiomatic proofs. But far more gold was yielded by that most powerful mind.

Only recently, because of the scholarship of Rudiger Thiele, has the world learned that in fact a twenty-fourth problem was offered by Hilbert (in one of his notebooks) [Thiele2001]. As Hilbert himself said, the problem was not included in the famous Paris talk because he had not as yet adequately formulated it. The problem focuses on the finding of simpler proofs. As he notes, mathematicians should simplify mathematics, not complicate it. But what makes a proof simpler, and in which specific ways is one proof simpler than another?

Ceteris paribus, for the most obvious case, proof **P** is simpler than proof **Q** if **P** is strictly shorter than **Q**. A second case focuses on formula (or equation) complexity, where the *formula complexity* (*equation complexity*) of a proof is k if and only if one of its deduced steps consists of k symbols and all other deduced steps consist of k or fewer. A proof **P** is simpler than a proof **Q**, all things being equal, if the formula (equation) complexity of **P** is strictly less than that of **Q**. Where the *size* (gleaned from a conversation with D. Ulrich) of a proof equals the total number of symbols in its deduced steps, **P** is simpler than **Q** when **P** has strictly smaller size. A fourth and subtler case concerns lemma presence in a proof. The proof **P** is simpler than the proof **Q** when **P** avoids the use of some powerful lemma **L** whereas **Q** relies on the use of **L**. Finally, among the other cases, I find interesting that concerning the type of term present. In particular, for example, **P** is simpler than **Q** when **P** avoids the use of *double-negation terms*

whereas **Q** relies on such. A double-negation term t is a term of the form $n(n(s))$ for some term s, where the function n denotes negation (as occurs, for example, in two-valued sentential calculus).

Based on the preceding, the reader may have formulated a most natural conjecture concerning what is about to be written:

– Automated reasoning programs can and do answer open questions by producing a first proof and answers others by discovering simpler proofs.

With the evidence to be presented in Section 4, many – or perhaps many, many – will join me in the attribution of *monumental* to the field of automated reasoning. Before such a presentation, in order is the nature of a *marvelous dream* and some details concerning the journey culminating in reaching that dream.

2 An Impossible Dream

In 1963 at Argonne National Laboratory, I began my study of what was then called *mechanical theorem proving*, a rather distasteful name for theorem proving certainly was not attacked mechanically. I was not fresh from graduate school, having been at the laboratory for six years. As for background, in 1954 I received my masters from the University of Chicago and in 1957 my Ph.D. from the University of Illinois (in group theory, under R. Baer). So I was well schooled in mathematics. Perhaps because of that schooling, I would (I am almost certain) have conjectured in 1960 that an attempt to find proofs with a computer seemed totally out of reach. Indeed, my few years of computing would most likely have enabled me to conclude that no way existed to communicate concepts such as commutativity, associativity, group, ring, and the like. And, even if such was possible, how would a program set about to search for and then find a proof? One lesson to be learned: Do not attempt to estimate the power of the mind, especially that of a researcher.

The lesser of two dreams – which had to be realized if the second and impossible dream was to be pursued – was to provide the means for a computer program to accept the assignment of searching for a proof. The second and impossible dream was to rely on such means, embellish them, extend them, and then make significant contributions to mathematics and to logic. Indeed, could a computer emulate so well the mind of a researcher that new results would come into existence? Could the knowledge, experience, and intuition of a master be encoded? And how would a program recognize, among its drawn conclusions, those that would merit accolades?

To answer these questions, and others that merited answering, I embarked on an exciting journey – one that continues today. The stage will then be set for presenting the promised evidence.

3 A Lengthy Journey of Almost Four Decades

I shall in the main confine my treatment – from there (the early 1960s) to here (late 2001) – to items featuring the Argonne extended group of researchers. Of

course, the work of others counts substantially. For example, one merely glances at the Boyer-Moore incredible success with program verification to experience amazement. A successor to the original program is now being used by the chip manufacturer AMD. Many deep theorems have been proved by students and by researchers with reliance on one of the incarnations of the Boyer-Moore effort. But I am sketching only a fraction of an almost-four-decade journey, not a full history.

For reasons not to be covered here, the 1960s witnessed relatively little experimentation; few reasoning programs existed. That paucity, according to me, hindered the needed advances. In contrast, from the beginning, Argonne emphasized experiments, featuring theorems taken from the literature. At first, and for many years, the group focused on finding proofs of known results. The explanation that was given – and I think a reasonable one – was that, if our program could not find proofs of theorems already proven, then how could we expect to answer open questions. Yes, I think it accurate to say that we were pursuing the impossible dream of eventually making important contributions, adding new knowledge in the form of new proofs. Nevertheless, I would have predicted, and in effect did so, that my lifetime would see little or no such successes. How wrong I was!

A more than casual observer, examining the evidence of the early and mid-1960s, would also have made the prediction that little or nothing would ever come of the effort devoted to eventually proving significant theorems. For example, although we at Argonne were somewhat gratified when our program proved that groups of exponent 2 are commutative, the theorem is very easy to prove and lacks much depth. When we turned to ring theory and found that, for the program to prove that minus x times minus y equals xy, we were forced to include two lemmas concerning the product of 0 and x, we were not filled with optimism. Our observer would have most likely scoffed with justification at the weakness of our automated search for proofs. For a third bit of negative evidence, for the program to complete a proof that the square root of 2 is irrational, we were again forced to include a crucial lemma. Given this sampling of data, who would have predicted eventual success?

Sure, we had not yet automated any form of equality-oriented reasoning (paramodulation was our choice eventually), nor had demodulation been formulated. Equality was simply treated as just another relation, no different than the relation of containment, subset, or the like. Yes, hyperresolution had been formulated, but the program was still forced to rely on appropriate clauses to capture equality substitution and other properties. The three theorems just cited yield far more readily to treating equality as built in, as "understood". The set of support strategy – which had been formulated to conquer the exponent-2 theorem and which is still considered today to be the most powerful restriction strategy – was useful but not sufficient to crush the other two theorems. The unsatisfying study of the cited theorem from ring theory produced nothing of consequence. In contrast, the study of the number theory problem did lead to the formulation and introduction of demodulation; however, the power offered

by a reasoning program was still indeed inadequate. So why did we press on; what made us believe the far distant future was more than promising?

I never asked others in the Argonne group that included (before 1980) George Robinson, Dan Carson, Lee Shalla, Ross Overbeek, Ewing Lusk, Robert Veroff, Brian Smith, Steve Winker. Therefore, I can speak only for myself, and then not with certainty. Clearly, strategy fascinated me. The improved power offered by my introduction of the unit preference strategy and the set of support strategy was indeed gratifying. I was often heard to say from the 1960s throughout the 1980s that new strategies were needed, and I expressed puzzlement at the lack of intense research devoted to their formulation – not that I myself was making many contributions in this context.

Also, at least for me, there was the excellent companionship of fine minds, sharing the belief in experimentation. The almost exclusive emphasis was on proving theorems from mathematics and logic, the exception being the work of Smith and Veroff, which was primarily concerned with program verification. Finally, and perhaps accurately, we accepted the nature of basic research, understood that the main objectives would be difficult to reach and possibly years away.

The value and importance of Overbeek's joining the group in the early 1970s cannot be overestimated. He was, and still is, a master at systems design, one of the best in the entire world of computing. He brought with him his program for proving theorems, followed by many later and improved versions. By then the field had survived being called "automatic theorem proving", which it certainly was not, and had more accurately become known as "automated theorem proving". The mid-1970s witnessed the publication of a paper devoted to experiments in theorem proving (in part coauthored by John McCharen, a member of the extended group for but a short time). But even then (in my view) far, far too little experimentation was featured throughout the world of automated theorem proving.

With Overbeek as the motivating force, in the late 1970s our journey became most hazardous: We ventured into the dangerous and often unrewarding domain of *open questions*. As far as I know, with the exception of SAM's lemma proved in the mid-1960s, the field had provided nothing new to mathematics and to logic. We at Argonne were about to change that aspect to a small extent. Indeed, we first answered a set of small open questions from ternary Boolean algebra [Winker1978]. Prompted by that minor achievement and motivated by a conversation I had with I. J. Kaplansky in which I requested a reasonable target, we then answered a far more interesting question concerning involutions, antiautomorphisms, and semigroups [Winker1981]. Winker played the key role in both achievements.

Next, the logician John Kalman (expert in equivalential calculus) visited Argonne, bringing with him seven open questions concerning the status of certain formulas in the context of being single axioms for the cited area of logic. Winker and I answered six of the seven questions, with the status of the formula XCB even at this time (2001) still uncertain. Around the time Winker and I were

collaborating (1980), I had introduced the term "automated reasoning", in part because of our study of conjectures, program verification, puzzle solving, and, of course, theorem proving. History shows that the term has lived on and gained rather wide acceptance.

In the early and mid-1980s, a vigorous debate rose concerning the value of clause retention. The Argonne paradigm insisted on retaining ever-growing sets of conclusions in the form of clauses. Various other paradigms did not share our enthusiasm for this aspect. We still maintain that such retention is crucial if a program is to provide substantial assistance in answering deep questions and solving hard problems.

The early 1980s marked yet another significant development: William McCune joined the group. He has proved to be a master of systems design – his program OTTER is (in my view) currently the most versatile and powerful reasoning program in existence. His monumental success (with his program EQP) in answering the Robbins algebra question (that had remained open for more than six decades) [McCune1997] and his monograph on open questions answered (with his colleague R. Padmanabhan) [McCune1996] attest to the vital role he has played and continues to play in automated reasoning. He and I (in the late 1980s) answered open questions in combinatory logic posed by R. Smullyan [Wos1993;McCune1987]. Roughly at the same time, Lusk with his colleague McFadden answered some tough questions concerning semigroup order [Lusk1987]. All of the questions we have cited and all we will cite were attacked with indispensable assistance from one of our reasoning programs. Their diversity provides strong evidence that the field was and is moving forward with increasing speed.

These varied and, in some cases, impressive successes in no way silenced the skeptics. Indeed, in a 1992 article published in the *New York Times*, mathematicians from various fields expressed a negative view of automated reasoning. Some asserted that, among the flaws, one could not learn from proofs produced through automated means. Of course, such is clearly not the case. For a first example, OTTER has discovered proofs of significant theorems that avoid the reliance on thought-to-be-indispensable lemmas. For a second example, OTTER has found proofs in which so-called double-negation terms are absent, terms of the form $n(n(t))$ for some term t. A glance at the literature strongly suggests that logicians were unaware of such proofs and, further, may have believed such was not possible in many of the cases in which OTTER has succeeded. More generally, Michael Beeson has proved that, for certain axiom systems in certain areas of logic, one is guaranteed the existence of such double-negation free proofs when the theorem is free of double negation. A study of the proofs that avoid various lemmas or that avoid various classes of term can indeed be instructive and can teach much to both the student and the experienced researcher.

With the new millennium, the journey continues. New members have joined the extended Argonne group, among whom are Branden Fitelson, Kenneth Harris, and Zachary Ernst. Although not in the field, each alone as well as together have found the field of automated reasoning most engrossing and the use of OTTER so intriguing. Mathematicians are now assisted by a reasoning program, and chip designers have theorem-proving groups.

4 Recent Successes

Far more detail, including proofs and input files, relevant to the material of this section will be found in the planned book entitled *Automated Reasoning and the Discovery of Missing and Elegant Proofs*.

Still surprising to me, (from what it appears) not all researchers in automated reasoning consider applications the most important aspect. Although I clearly emphasize the proving of theorems in mathematics and in logic, I suspect that, from the viewpoint of science in general, program verification is the most important application. A close second is circuit and chip validation and design. However, as programs are continually enhanced to offer more and more power with the evidence of proving deeper and deeper theorems, the two cited applications will benefit. This section offers evidence of significant advances and is the pinnacle of this article in its demonstration of the power and value of today's automated reasoning program. In the main, I focus on successes that have occurred since McCune conquered the Robbins algebra problem and also answered numerous open questions discussed in his monograph. Challenges and open questions are offered in this section.

In both mathematics and logic, much energy has been devoted to finding axiom systems with appealing characteristics such as independence and smallness in number. Regarding the latter, often the limiting case has been reached, the discovery of a single axiom. In group theory, for example, McCune contributed (among others) the following single axiom [McCune1993], where the functions f and g respectively denote product and inverse.

```
f(x,g(f(y,f(f(f(z,g(z)),g(f(u,y))),x)))) = u.
```

Kunen then offered the following single axiom, an improvement in that its variable richness is three rather than four [Kunen1992].

```
f(g(f(y,g(y))),f(f(g(y),z),g(f(g(f(y,x)),z)))) = x.
```

Ideally, one might prefer a single axiom in which product, inverse, and the identity e were present, but such cannot be done, as proved by Tarski. However, researchers are offered an interesting open question. Does there exist a single axiom whose variable richness is three and whose length is equal to the McCune offering (shorter than that of Kunen)?

Lattice theory also admits single axioms. Until very recently, the shortest known (as far as I can determine) had length 79 with variable richness six, where "v" denotes union and "^" denotes intersection.

```
(((x ^ y) v (y ^ (x v y))) ^ z) v (((x ^ (((u ^ y) v
  (y ^ v)) v y)) v (((y ^ (((u v (y v v)) ^ (w v y)) ^ y)) v
  (v6 ^ (y v (((u v (y v v)) ^ (w v y)) ^ y)))) ^ (x v (((u ^ y) v
  (y ^ v)) v y)))) ^ (((x ^ y) v (y ^ (x v y))) v z)) = y.
```

McCune and Veroff, with our colleague R. Padmanabhan, laid the groundwork for seeking a far shorter single axiom. McCune has succeeded, finding the following 29-symbol axiom of variable richness eight.

```
(((y v x)^x) v (((z^ (x v x)) v (u^x))^x1) )^
  (w v ((v6 v x)^ (x v v7))) = x.
```

Immediately three open questions arise. First, does there exist a shorter single axiom? Second, does there exist a single axiom with less variable richness than eight whose length is, say, less than or equal to forty? Third, in the spirit of Hilbert's twenty-fourth problem focusing on proof simplification, does there exist for the McCune axiom a proof of length strictly less than 59?

The last question merits a bit more detail. In particular, I have a preference for proofs relying solely on forward reasoning, the denial being used to complete the proof. Such proofs have the appealing property of explicitly deducing the goal or goals of the theorem, or generalizations of such. I also prefer proofs in which demodulation is not present. To me (as well as other mathematicians with whom I have spoken) its absence often facilitates more insight into the nature of the proof. In contrast, a Knuth-Bendix bidirectional proof is usually easier to complete, sometimes far easier. Therefore, if one has the objective of producing a forward-reasoning demodulation-free proof, one might be wise to first seek a Knuth-Bendix bidirectional proof and then embark on some form of proof translation. The process of proof translation is often rather difficult, presenting various obstacles.

I have found with OTTER's assistance a 59-step proof of the type I prefer. That proof has variable richness nine, which leads to yet another challenge. Can one find a proof for the given single axiom for lattice theory such that the proof has variable richness that does not exceed eight?

Next in order is Boolean algebra, a field that is studied in terms of various operators. Among the studies is that focusing on disjunction coupled with negation and that focusing on the Sheffer stroke. Regarding the latter, S. Wolfram suggested as candidate axiom systems a pair of equations and twenty-five equations to be considered separately [Veroff2001b]. Robert Veroff, with an intense effort relying on his use of his ingenious methodology called *sketches*, showed that the pair is indeed a 2-basis, an axiom system for Boolean algebra when presented in terms of the Sheffer stroke. Still using OTTER, he then showed that, if commutativity was coupled with any of the twenty-five candidates, an axiom system resulted. McCune then showed that commutativity could be proved dependent for two of the twenty-five, yielding two different single axioms. McCune then proved that their mirror images are also single axioms. But the study did not stop there.

Indeed, two of the newer members of the extended Argonne group, Branden Fitelson and Kenneth Harris, showed that no shorter single axiom in terms of the Sheffer stroke existed, shorter than length 15, the length of the cited single axioms [McCune2001]. Further, with the contribution of a summer student Andrew Feist, members of the group showed with models that seven of the twenty-five are too weak. Summarizing, open questions remain, each regarding the possible axiomatic status of sixteen of the twenty-five. Of those proved to be single axioms, where the function f denotes the Sheffer stroke, my favorite is the following.

```
f(f(f(x,f(y,x)),x),f(y,f(z,x))) = y.
```

But Boolean algebra had additional treasure to mine.

In particular, McCune conducted a lengthy study of this algebra in terms of disjunction and negation. His effort was rewarded, and mathematics was enriched. He found various 22-symbol axioms, including the following, where + denotes **or** and ~ denotes **not**.

```
~(~(~(x+y)+z)+~(x+~(~z+~(z+u)))) = z.
```

Two open questions merit study. Does there exist a shorter single axiom than length 22 of this type? Does there exist a forward-reasoning, demodulation-free proof shorter than length 57 for the McCune axiom? The reader's conclusion is correct: I have such a 57-step step proof.

The preceding citations are generally placed under the province of mathematics rather than under logic. Consistent with my earlier and frequent references to mathematics *and* logic, the time has come for the latter to take center stage. Although not required, to provide more diverse evidence of the advances that have occurred, I shall feature theorems in which equality plays no role, in contrast to the preceding.

The researchers that played the main role in what is to be presented now are the newest generation, the newer members of the extended Argonne group. The first of these, Fitelson, entered the picture through an e-mail to me, asking my view of a rather intricate input file he had produced to prove a theorem of Lukasiewicz concerning the dependence of an axiom of Frege [Lukasiewicz1970]. Fitelson's (and OTTER's) proof were markedly different from that of Lukasiewicz. He had become interested in automated reasoning, so he informed me, because of reading my book entitled *The Automation of Reasoning: An Experimenter's Notebook and OTTER Tutorial*. I was so impressed by the nature of his included input file that I contacted him by phone, and we have conducted research together almost continually the past three years.

For a splendid example of the interworkings of our group, Veroff, learning of Fitelson's new 8-step proof of the cited Lukasiewicz theorem, made an important contribution. Specifically, Veroff showed that, quite likely, no shorter proof could be found and that there are a number of 8-step proofs, including the original Lukasiewicz proof [Veroff2001a]. The inference rule used is condensed detachment, encoded with a single clause of three literals and the use of hyperresolution. Veroff formulated a new use of linked inference rules to produce the cited result concerning different proofs and the likelihood that the shortest had been found.

Shortly thereafter, Fitelson answered an open question posed by Epstein. Quite different from the search for a single axiom, the question focused on possible axiom dependence among the following six axioms for two-valued sentential (or propositional) calculus, asking which (if any) of the six axioms is dependent on the remaining [Epstein1995]. The functions a, i, and n denote, respectively, conjunction, implication, and negation.

```
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(A(x,y),x)).
P(i(x,i(y,A(x,y)))).
P(i(x,i(y,x))).
P(i(n(x),i(x,y))).
P(i(i(x,y),i(i(n(x),y),y))).
```

To give the reader (unaided or aided by some reasoning program) time to attack the at-one-time-open Epstein question uninfluenced by its answer, I shall delay presentation of the facts for a short while.

Early in our collaboration, Fitelson provided proofs, obtained with OTTER, of the two halves of an associativity relation that holds in infinite-valued sentential calculus [Fitelson2001]. As far as I know, these proofs marked an advance for logic in that each was the first such relying solely on condensed detachment for drawing conclusions. From the viewpoint of automated reasoning, the pair of proofs illustrates the value of strategy. Specifically, when Fitelson was able to obtain one of the two proofs but unable to obtain the other even after many CPU-hours, he turned to the use of the resonance strategy. In particular, he used the proof steps of the new proof (of one half of the relation) to direct the program's attack on finding the proof of the other half – and OTTER quickly succeeded. When equality is not involved (as is the case in the typical axiomatic treatment of infinite-valued sentential calculus), proofs based solely on condensed detachment are preferred. From Hilbert's writings, one might surmise he would indeed have recommended that proofs remain within the theory. In an important sense, such proofs are simpler than those in which the author instead relies on the use of equality-oriented reasoning.

With this vignette completed, it is time now for the answer to the open question offered by Epstein as to which (if any) of the given six axioms are dependent on the remaining.

The fourth axiom is dependent, and I now have a 10-step proof solely in terms of condensed detachment of its dependency. Does there exist a shorter proof relying solely on condensed detachment? As for the other five axioms, they form (as Fitelson showed with automated model generation) an independent set, which answers another of Epstein's questions. For a challenge, one might attempt to prove that the independent five-axiom set indeed axiomatizes two-valued sentential calculus by deriving from that set some previously known axiomatization. I have a 12-step proof that derives the following Lukasiewicz three-axiom system; a shorter proof in that context might exist.

```
%  Lukasiewicz 1 2 3.
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).
```

The proof has added interest in the context of simpler proofs (which is relevant to the Hilbert twenty-fourth problem, but not regarding proof length). Indeed, although nine of the twelve steps rely on the presence of the function n for

negation, the proof is free of double-negation terms, terms of the form $n(n(t))$ for any term t. I shall later return to that aspect of proof simplification in part because of the Hilbert problem, in part because of the emphasis on evidence of field advancement, and in part to illustrate what can be done with an automated reasoning program in the context of term avoidance and also the context of lemma avoidance.

Having found condensed-detachment proofs for an associativity law, Fitelson turned his attention to the far more difficult task of finding a condensed-detachment proof of a distributivity law in infinite-valued sentential calculus. He enlisted the collaboration of his long-term colleague Kenneth Harris, who is now yet another member of the extended Argonne group. Based on the literature and their collective insight, they decided that their likelihood of success would be increased if they began their attack by relying on equality-oriented reasoning. Therefore, rather than using condensed detachment and hyperresolution, Fitelson and Harris turned to paramodulation. After substantial effort in part by hand and in part by program, they had their proof in terms of equality. However, much work remained before the desired condensed-detachment proof would appear, for their proof relied on heavy use of demodulation and on bidirectional reasoning.

The presence of demodulation and steps that assert that s does not equal t for terms s and t and that result from reasoning backward from the denial each presented a rather severe obstacle. Fortunately, the group contains a master at proof conversion, Robert Veroff. He took the Fitelson-Harris proof and applied his methodologies to their proof, using his extended version of OTTER, and produced a forward-reasoning, demodulation-free proof relying solely on paramodulation. But the goal had not yet been reached, clearly.

McCune supplied the next key piece to the puzzle: He provided an algorithm to convert the paramodulation proof to a condensed-detachment proof. And the game was essentially won [Harris2001]. What remained was my role, which was to apply various methodologies to simplify the resulting proof in the context of length and term structure. In particular, regarding the latter, the final proof is free of double-negation terms, resulting from a move not often endorsed by all the members of the group. Indeed, our group can trace much of its success to the ability to disagree sharply about various aspects of the field and still perform as a well-integrated team. Such disagreements have in fact played a key role in the contributions in the areas of inference rule, strategy, implementation, and the like for which we have been credited.

Possibly because of McCune's success in finding single axioms for Boolean algebra in terms of disjunction and negation, Fitelson with Harris made a similar study for the implicational fragment of infinite-valued sentential calculus. Harris produced the following 69-symbol single axiom (not counting the predicate symbol).

```
P(i(i(i(x,i(y,x)),i(i(i(i(i(i(i(i(i(z,u),i(i(v,z),i(v,u))),
  i(i(w,i(v6,w)),v7)),v7),i(i(i(i(v8,v9),v9),
  i(i(v9,v8),v8)),v10)),v10),i(i(i(i(v11,v12),i(v12,v11)),
  i(v12,v11)),v13)),v13),i(i(v14,i(v15,v14)),v16))),v16)).
```

Immediately two open questions come to mind. Is there a shorter single axiom? Is there a single axiom in strictly fewer than seventeen variables? Given my 15-step proof that uses the Harris axiom to derive the standard four-axiom system for the implicational fragment, one wonders if a shorter proof in that context exists.

```
%  the four-axiom system.
P(i(x,i(y,x))).
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(i(x,y),y),i(i(y,x),x))).
P(i(i(i(x,y),i(y,x)),i(y,x))).
```

Some months before this success, Fitelson and Harris had found new single axioms for two-valued sentential calculus in terms of the Sheffer stroke, among which is the following.

```
P((D(D(x,D(y,z)),D(D(x,D(y,z)),D(D(u,z),D(D(z,u),D(x,u)))))).
```

One might enjoy accepting the challenge of proving that the given formula is in fact a single axiom by deriving Nicod's single axiom, the following negated.

```
-P(D(D(a,D(b,c)),D(D(e,D(e,e)),D(D(f,b),D(D(a,f),D(a,f)))))) |
  $ANS(NICOD).
```

They proved that no shorter single axiom in terms of the Sheffer stroke exists.

The Fitelson-Harris studies naturally included attempts to prove that no shorter single axiom existed for various areas of logic, such as propositional logic. That area is axiomatized by the following single formula of Meredith.

```
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
```

Meredith's finding of this single axiom answered a question posed by Lukasiewicz when he offered in the mid-1930s his 23-letter single axiom for propositional calculus. Specifically, Lukasiewicz noted that three years had been devoted to finding the 23-letter formula, and he would leave it to others to find, if such existed, a shorter single axiom [Lukasiewicz1970]. Whether an axiom with strictly fewer than 21 letters (the length of the Meredith axiom) exists for this area of logic is still open. Fitelson and Harris have made substantial progress on this problem, but the work is not complete.

However, in their efforts, they found most valuable the addition of another colleague, Zachary Ernst, who is now yet one more member of the Argonne extended group. Ernst took to the type of problem under discussion beautifully. He also became enthralled with the use of OTTER. One of the areas Fitelson, Harris, and Ernst attacked was $C5$, the implicational fragment of $S5$. The logic known as $S5$ is a modal logic, in part formulated to capture more closely the widely accepted notion of implication, in contrast to that typically featured in other areas of logic. Meredith and others had found axiom systems for $C5$, among which is the following Meredith single axiom.

```
P(i(i(i(i(i(x,x),y),z),i(u,v)),i(i(v,y),i(w,i(u,y))))).
```

Ernst, through diligence, thoroughness, and brilliance, found six additional single axioms [Ernst2001b], among which is the following that exhibits a most interesting property.

```
P(i(i(i(i(x,y),z),i(i(u,u),y)),i(i(y,v),i(w,i(x,v))))).
```

The reader might enjoy comparing the two axioms with the objective of identifying the property I am about to discuss.

Before stating explicitly what that property is, I shall set the stage by quickly touching on the *resonance strategy* introduced in the early 1990s. My formulation of that strategy can be traced directly to Dana Scott's offering 68 theses (theorems of Lukasiewicz) for consideration by OTTER. A *resonator* is a formula or equation whose variables are treated as indistinguishable; its functional pattern is what matters. The researcher assigns a value to each included resonator. The program's reasoning is then directed accordingly, assigning the corresponding priority to any formula or equation that matches a resonator. Low assigned values give high priority to matching conclusions. The Nicod and Fitelson single axioms for propositional calculus in terms of the Sheffer stroke are in the same equivalence class, that class defined by taking either and treating it as a resonator. And the stage is set for answering the posed question concerning the two given single axioms (Meredith's and Ernst's) for $C5$.

The cited Ernst formula is the fourth of six new single axioms he found. The first three as well as that of Meredith all are in the same resonator-equivalence class. The fourth is not; its functional pattern differs from that of Meredith – indeed most gratifying!

But, where the *size* of a basis (axiom system) is measured in terms of the total number of symbols present, the Meredith 1-basis and each of the Ernst 1-bases have size 21, no difference. Naturally, one wonders about the existence of a smaller-sized basis, when no restriction is placed on the number of elements (axioms) in the basis. Ernst examined that question with the assistance of OTTER and produced a startling result. A basis of size less than 21 (the size of, for example, the Meredith basis) does exist, the following.

```
P(i(i(x,y),i(i(i(i(y,z),w),z),i(x,z)))).
P(i(x,x)).
```

This basis has size 18 (not counting predicate symbol occurrences). And, ensuring that no loose ends remained, the trio of Ernst, Fitelson, and Harris finished the game well, proving that no smaller in size (total symbol count) basis for $C5$ can exist. They then turned their attention to $C4$.

Apparently Meredith made a serious attempt to find a single axiom for $C4$ to no avail. Clearly, the problem was extremely difficult, if indeed such an axiom did exist. But, as it turned out, Ernst, Fitelson, and Harris conquered the problem [Ernst2001b], finding this elusive single axiom, the following.

```
P(i(i(x,i(i(y,i(z,z)),i(x,u))),i(i(u,v),i(w,i(x,v))))).
```

To prove that the given formula suffices, one might accept the challenge of attempting to deduce from it the following two-axiom system for $C4$.

```
P(i(i(x,i(y,z)),i(i(x,y),i(u,i(x,z))))).
P(i(x,i(y,y))).
```

The shortest proof I have found at this point has length 33. Their success leads naturally to an intriguing open question. Does there exist another single axiom for $C4$, or have my colleagues found the only such?

Inspired by the $C4$ success, Ernst turned to another area of logic, conducting an intense study of the implicational fragment of Dunn's classical relevance logic RM, which is called $RM \rightarrow$. Meyer and Parks provided the following impressive and independent basis for $RM \rightarrow$.

```
P(i(i(x,y),i(i(y,z),i(x,z)))).  % suffixing
P(i(x,i(i(x,y),y))).            % assertion
P(i(i(x,i(x,y)),i(x,y))).       % contraction
% Following is Parke's axiom -- GOAL
P(i(i(i(i(i(x,y),y),x),z),i(i(i(i(i(y,x),x),y),z),z))).
```

The system $RM \rightarrow$ is equivalent to the implicational fragment of Sobocinski's three-valued logic S. Perhaps a more satisfying basis exists, possibly of the same cardinality, of smaller size. In particular, perhaps the fourth and most complex axiom could be replaced by a less complex axiom.

The Ernst study did just that, yielding the following four-axiom system.

```
P(i(i(x,y),i(i(y,z),i(x,z)))).  % suffixing
P(i(x,i(i(x,y),y))).            % assertion
P(i(i(x,i(x,y)),i(x,y))).       % contraction
P(i(i(i(i(i(x,y),z),i(y,x)),z),z)).
```

He and OTTER found a replacement for the 21-symbol axiom, namely, an axiom of complexity thirteen. His research yielded even more, a second 4-basis, where the last given formula is replaced by the following.

```
P(i(i(i(x,i(i(i(y,x),z),y)),z),z)).
```

Quite interesting, the two 13-symbol axioms are not in the same resonator-equivalence class. But the story is not yet complete.

Indeed, Fitelson, with his continual curiosity concerning axiom dependence, considered the two new Ernst bases for $RM \rightarrow$. And the world of logic was treated to a marvelous result: Fitelson showed that the axiom of contraction is dependent on the remaining three basis elements, for each of the Ernst bases [Ernst2001a]. Ernst, Fitelson, and Harris presented to logic a 3-basis of size 31 to replace the well-known 4-basis of size 48.

Immediately, two open questions offer themselves. Does there exist a basis of smaller size than 31? Does there exist a basis of two or fewer members? If the reader enjoys challenges regarding proof length, I note that I have in hand a 38-step proof that relies on the first Ernst basis and deduces the contraction axiom and the Parks axiom. For the second Ernst basis, I have found a 37-step proof.

At this point, I now turn to results directly pertinent to the Hilbert twenty-fourth problem, various types of proof simplification. In the context of proof length, Meredith and Prior were clearly interested [Meredith1963]. Indeed, they published an "abridgement" of a proof of Lukasiewicz, for the Lukasiewicz shortest single axiom for the implicational fragment f two-valued logic. The Meredith-Prior proof has length 33 (applications of condensed detachment), and the Lukasiewicz proof has length 34. One might, before reading any further, enjoy the challenge of using as hypothesis the following 13-symbol Lukasiewicz formula and attempting to deduce the Tarski-Bernays system.

```
%  following is shortest single axiom
%  for the implicational fragment
P(i(i(i(x,y),z),i(i(z,x),i(u,x)))).
-P(i(p,i(q,p))) | -P(i(i(i(p,q),p),p)) |
  -P(i(i(p,q),i(i(q,r),i(p,r)))) |
  $ANS(TARSKI_BERNAYS).
```

Because Fitelson had witnessed a number of successes on my part in proof refinement with regard to length (some of which I report here), he brought the Meredith-Prior result to my attention. He posed for me the problem of finding a further abridgment. As one justifiably would predict, I thought the chances small in view of the mastery of logic evident in Meredith's and Prior's works. Notwithstanding, the problem was most intriguing, and I had OTTER for a companion and a powerful assistant.

The sought-after proof of length 32 or less did indeed prove elusive. The use of the resonance strategy, of Veroff's hints strategy, of a methodology that blocks steps of a given proof one at a time – none of these won the game. However, by using as resonators the Meredith-Prior proof and the Lukasiewicz proof and various methodologies, OTTER did complete another 33-step proof. That proof contained as a subproof a 30-step proof of the third member of the cited Tarski-Bernays system. And an idea was born, a possible new strategy, one that would be called *cramming*.

Intuitively, in the first incarnation, the object of the strategy is to reduce the length of the proof of a target conjunction by focusing on the proof of one member and attempting to "cram" as many of its steps into the needed remaining proofs. In the case of the target three-axiom Tarski-Bernays system, the plan (if successful) was to focus on the 30-step proof of the third member and cram so many of its steps into the desired proofs of the other two members that only two additional formulas would be needed. For this to occur, there must exist two pairs of clauses among the thirty such that condensed detachment applied to the pairs yields, respectively, the first and second members of the three-axiom system – highly unlikely, but possible. Fortunately, OTTER offers just what is needed to examine all pairs, including pairs in which the two elements are identical. In particular, with the command set(sos_queue), the program conducts a breadth-first search. Therefore, one merely places the (in this case) thirty clauses that prove the third member of the Tarski-Bernays system in the initial set of support. One additional move must be made to attempt to prevent the

program from retaining unwanted conclusions, conclusions other than the first and second members of the three-axiom system. The move consists of placing their correspondents in a hints list and assigning a very small maximum on the complexity of newly retained conclusions.

If the preceding had failed, clearly with almost certainty this episode would not be included here. Indeed, the story as expected ends with success, with the completion of a 32-step proof that is an abridgment of the Meredith-Prior abridgment. Especially for the thorough historian who enjoys the vagaries of science, I note that later experiments yielded different 30-step proofs of the third member, none of which permitted completion of the desired 32-step proof. Because of the charm of this result and its significance to both automated reasoning and logic, an open question is virtually demanded. Does there exist a proof that uses the Lukasiewicz shortest single axiom as hypothesis and that has as target the Tarski-Bernays system with strictly fewer than thirty-two applications of condensed detachment?

But reduction in proof length is just one refinement reflecting Hilbert's intest in simpler proofs. Quite different is that concerning *variable richness*, where the variable richness of a proof is k if and only if one of its deduced steps relies on exactly k distinct variables and all other deduced steps rely on k or fewer. If one has in hand a proof of variable richness, say, k and wishes a refinement of that proof whose richness is strictly less than k, OTTER offers precisely what is needed. One can include assign(max_distinct_vars,j), and the program will retain a newly deduced conclusion only if it relies on j or fewer distinct variables. Deepak Kapur conducted an investigation with the objective of finding a proof for Meredith's single axiom for two-valued logic that completes with the deduction of the cited Lukasiewicz three-axiom system such that its variable richness is six. Meredith's proof has richness seven, containing two deduced steps out of forty-one in which seven distinct variables are present. Kapur in fact succeeded in this context of proof refinement, producing with OTTER a 63-step proof of richness six. In answer to an anticipated question, I now have in hand a 49-step proof of variable richness six, only one of whose steps has that richness. As for answers to the next probable series of questions, no proof exists with richness strictly less than five, which can be seen by noting that the first condensed detachment step has that richness and is the result of applying condensed detachment to two copies of the Meredith single axiom. Yes, there does exist a proof of variable richness five, and I have in hand (because of OTTER) one of length 68.

Of still a different nature is proof refinement with respect to *term structure*. For example, all things being equal, a second proof is simpler than the first when the second avoids the use of double-negation terms, whereas the first relies on them heavily. Meredith's (in effect) 41-step proof for his single axiom for two-valued logic includes seventeen steps relying on double negation. If a researcher wishes to avoid such terms or to avoid some other class, OTTER again comes to the rescue through the use of demodulation or weighting. For example, by including the demodulator of the form (n(n(x)) = junk) with others to propagate

the corresponding rewrite, double-negation terms can be avoided among retained clauses. I have in hand many, many proofs that the literature suggests require the use of double negation but that avoid it entirely. In fact, almost never did I fail to find such a proof for the theorem under consideration. I was thus curious about the conditions that guaranteed the existence of a double-negation-free proof.

D. Ulrich beautifully refined my concern by asking whether there existed an axiom system for two-valued logic such that, whenever the theorem to be proved was free of double negation, a proof could be found that was also free of double negation. Thus began a collaboration that first included the newest member of the extended Argonne automated reasoning group, Michael Beeson; later Veroff joined in the collaboration. Beeson answered the Ulrich question (which astounded me) by providing most of the details for a proof showing that the cited three-axiom system of Lukasiewicz has the desired property. Veroff and I supplied proofs of some needed lemmas, through (of course) OTTER's cooperation. Perhaps more astounding, Beeson (with some assistance by Veroff and me) then proved the corresponding theorem for infinite-valued sentential calculus with the following axiom system.

```
P(i(x,i(y,x))).
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(i(x,y),y),i(i(y,x),x))).
P(i(i(n(x),n(y)),i(y,x))).
```

One might quickly conjecture that proof simplification in one aspect comes at the expense of simplification in another. For example, the removal of double-negation terms most likely results in a rather longer proof than that in hand. Similarly, the avoidance of thought-to-be-indispensable lemmas, a nice refinement, quite likely lengthens the resulting proof. I can report with satisfaction and with some amount of awe that, often, such is not the case. For but one example, infinite-valued sentential calculus takes center stage, with the preceding four axioms in focus. A fifth axiom (the following) once thought necessary but later proved by that master Meredith dependent was the target.

```
P(i(i(i(x,y),i(y,x)),i(y,x))).
```

Before OTTER and I attacked the problem, the literature offered a 37-step proof, one that relied upon double negation. Further, a review of the literature suggested that the following three lemmas might be indispensable.

```
P(i(i(i(x,y),i(z,y)),i(i(y,x),i(z,x)))).
P(i(i(x,y),i(n(y),n(x)))).
P(i(i(i(x,y),i(x,z)),i(i(y,x),i(y,z)))).
```

After years of study, frequently interrupted by other investigations, I finally have in hand a 30-step proof, free of double negation, and avoiding the use of all three cited lemmas. I know of no shorter proof than that of length 30, regardless of its other properties. In other words – and this example is by no means isolated – a refinement in one aspect does not necessitate a cost in one or more other aspects.

The student, the experienced researcher, the simply curious – each might wonder why such proofs were missing from the literature and, apparently in many cases, perhaps out of reach of the unaided. Perhaps today's powerful and fast computers are essentially the answer. In my view (shared by others), such is indeed not the case. Rather, the answer lies in the use of strategy and methodology that permits and even encourages the program to search in the vast space of conclusions where no one had searched before. Indeed, some of the approaches that have led to many of the cited successes (and others not presented here) could be classed as counterintuitive.

5 Highlights and Perspective

This article is written in honor of Joerg Siekmann's sixtieth birthday; Joerg and I have known each other for decades. To put all in perspective, I sample a bit of history (at one end of the spectrum), and (at the other) I focus on recent significant contributions to mathematics and to logic that resulted directly from the use of an automated reasoning program. The results would have been out of reach, so it appears, were it not for the introduction of various strategies and methodologies that depend on them. Heavy experimentation was the key as well as the ability to retain hundreds of thousands of new conclusions.

The article is intended to stimulate a wide audience, including those not primarily concerned with automated reasoning. Perhaps not obvious, that audience includes those mainly interested in circuit design and validation and those interested in program synthesis and verification. Indeed, for a small hint as to why such may be the case, I note that the reduction in the length of a constructive proof typically corresponds to a reduction in the complexity of the object being constructed, a circuit or a bit of computer code, for example. Proof simplification in the context of length, as well as in other contexts such as variable richness, are directly pertinent to the newly discovered Hilbert's twenty-fourth problem. To add to the possible usefulness of this article, I include open questions, challenges, some detail concerning strategy, some concerning methodology, and more.

I mark the birth of the current incarnation of automated reasoning with J. A. Robinson's introduction of binary resolution. In its beginning, the field could boast only of conquering the minuscule. However – and I say this in part directly to Joerg, who clearly shared with me and others the dream of automation – here in the year 2001, automated reasoning in the Olympics of science merits a gold medal for its achievements. They are many, and they are diverse.

At one end of the spectrum is the use of reasoning programs by chip manufacturers that include Intel and AMD; Robert Boyer and J Moore pioneered this important subfield of program verification. At the other end of the spectrum – but of equal significance in the long run – reasoning programs are making important contributions to both mathematics and logic. For a taste of the fecundity of the field, one need only turn to the Web site of Johan Belinfante, finding there hundreds and hundreds of proofs for theorems from set theory [Belinfante2001], proofs essentially out of reach but a decade ago. In that regard, for the researcher seeking a deep problem on which to work, (as suggested by Boyer) one might

seek a small set of axioms, say, sixty or fewer, that captures most of the set-theoretic reasoning needed for mathematics and logic outside set theory. For a taste of the breadth of automated reasoning's successes, one need only sample the material in this article or browse among the following highlights.

5.1 Axiom Systems with Various Properties

In both mathematics and logic, fine minds have devoted substantial research to the discovery (if such exists) of single axioms. In that context, automated reasoning programs (specifically McCune's OTTER) have proved to be a powerful reasoning assistant, and sometimes even a colleague, as the following highlights from the work of the Argonne extended group indicate.

- Boolean algebra, in terms of the Sheffer stroke (Veroff, McCune, Fitelson, Harris, Feist)
  - from 25 candidate equations (supplied by Wolfram), proof that 2 are single axioms, and then proved that their mirror images are also
  - proof that 7 of the 25 are too weak to be single axioms
  - proof that there exists no shorter axiom than length 15
- Boolean algebra, in terms of disjunction and negation (McCune)
  - discovery of ten 22-symbol single axioms
- Lattice theory (McCune)
  - discovery of a 29-symbol single axiom for lattices: far more attractive than the 40,000,000-symbol single axiom found algorithmically

In the preceding successes, equality is featured. As the experienced experimenter knows, equality presents various problems because of its nature. Paramodulation, for example, is a term-oriented inference rule in contrast to, say, hyperresolution, which is a literal-oriented rule. In logic, condensed detachment also is literal oriented, and its study therefore involves fewer obstacles than when equality is dominant. Nevertheless, problems in which condensed detachment is the sole rule of inference are most challenging. The following examples highlight some of our recent achievements.

- C5 (Ernst, Fitelson, Harris)
  - six new single axioms, three in the same resonator-equivalence class as Meredith's single axiom
  - shortest possible 2-basis
  - proof that no single axiom shorter than 21 symbols exists
  - proof that a basis with 18 symbols exists, the shortest possible
- C4 (Ernst, Fitelson, and Harris)
  - first single axiom
  - proof that a single axiom with 21 symbols exists, the shortest possible
  - proof that a 20-symbol 2-basis exists, the shortest possible
- RM$\rightarrow$ (Ernst, Fitelson, Harris)
  - smaller 4-basis than that of Meyer and Parks
  - 3-basis of size 31 (replacing 4-basis of size 48 of Meyer and Parks)
- Propositional logic, in terms of the Sheffer stroke (Fitelson and Harris)
  - new single axioms all of length 23, and no shorter exists

5.2 Proofs with Various Properties

Of an apparently different cast from the pursuit of axiom systems with various properties is the pursuit of proofs with various properties. I say "apparent" because size of basis (among other properties) and cardinality of basis have their obvious counterparts in proof, namely, the total number of symbols in the deduced steps of a proof and the precise number of such steps. For the past decade, I have devoted much research to such proof properties and to proof refinement (in the spirit of Hilbert's twenty-fourth problem). Here I highlight a few of my successes.

- Proof length
  - first proof (length 200) and shortest proof (length 50) deducing Lukasiewicz's three-axiom system from his 23-letter formula
  - 38-step proof, improving on Meredith's 41-step proof of his single axiom for two-valued logic
  - 30-step proof of the dependence of one of Lukasiewicz's five axioms for infinite-valued sentential calculus
- Term and lemma avoidance
  - cited shortest known proof (length 30) of axiom dependence in infinite-valued sentential calculus (1) avoids three seemingly crucial lemmas and (2) avoids double-negation terms

Intrigued by my success with the avoidance of double negation terms, M. Beeson (with assistance from Veroff and me) proved a charming metatheorem for two-valued sentential calculus: If the theorem **T** to be proved is free of double negation, then there must exist a proof **P** of **T** in which double negation is totally absent that relies on condensed detachment alone and that uses as hypotheses the Lukasiewicz three-axiom system. Beeson (with assistance from Veroff and from me) also showed that the independent four-axiom system for infinite-valued sentential calculus presented earlier in this article has the corresponding property to that of the Lukasiewicz three-axiom system.

All of these successes are directly related to Hilbert's twenty-fourth problem (as discovered by R. Thiele in his examination of Hilbert's files). The focus of that newly discovered problem is *simpler proofs* – whether of proof length or size, of term avoidance, of formula complexity, or of variable richness. Imagine my delight when I learned that my decade-long fascination with proof simplification had been a subject of considerable interest also to one of the masters of mathematics!

5.3 Techniques for Refining Proofs

The key to proof simplification is not, as some might suppose, the CPU speed of today's computer; rather, it is the availability of powerful strategies and effective methodologies. The past few years of my research have resulted in several new approaches that often aid in proof refinement – particularly with respect to proof length (applications of condensed detachment). The approaches also are effective where equality is the sole or dominant relation, and they are effective often in

finding a first proof or settling a conjecture. These approaches include *blocking* proof steps one at a time (through demodulation or weighting), directing the program's reasoning with the McCune's *ratio strategy* (which blends breadth first with choosing the initiator of an inference rule application by conclusion complexity), and *cramming* the steps of a proof of one member of a conjunction into the needed proofs of the other members.

Interesting – and indeed counterintuitive – is the fact that a reduction in proof length may result from the study of proof refinement in some other aspect, such as lemma avoidance or term avoidance. The avoidance of thought-to-be-indispensable lemmas, when successful, is a sometimes overlooked aspect of proof simplification, as is the avoidance of some class of term. For example, the cited proof of axiom dependence in infinite-valued sentential calculus (the one that avoids the use of three lemmas and double-negation terms) possesses one additional remarkable property: It has length 30 (applications of condensed detachment), the shortest proof I know of.

Is the reduction in length the result of the successful refinements in the two other aspects? I strongly suspect so. Indeed, were it not for instructing OTTER to avoid the so-called key three lemmas and avoid double-negation terms (each through the use of demodulation), I believe that the 30-step proof may have lay hidden forever. Its discovery (so it appears) was because of forcing the program to explore within the space of conclusions where it would ordinarily not spend the majority of its time. That part of the space would typically not be explored by the unaided master, I believe, because of its counterintuitive nature, avoiding key lemmas and avoiding double negation. I suspect that the density of proofs of interest is greater in the restricted space of conclusions than in the entire space of conclusions.

For a glimpse of what I think is the case, one might imagine spending one's research life in algebras in which associativity is present and then being asked to (intuitively) study a nonassociative algebra. One's experiences would almost certainly interfere with sought-after proofs. Put a slightly different way, one of the marvelous features of using a program such as OTTER is the capability to *explore where no researcher has gone before.*

5.4 Evolution and Revolution

The field has come a long way. Even its name has evolved, from mechanical theorem proving to automatic theorem proving to automated theorem proving and, most accurately today, to automated reasoning. At first, some satisfaction was drawn from finding proofs for essentially syntactic examples; now research is often rewarded with a significant contribution to mathematics, logic, or some other discipline. For but one example of the latter, McCune's work on ortholattices [McCune1998] has served physicists well. He was asked to find a small ortholattice in which the following equation fails, where v denotes join, ^ denotes meet, and ′ denotes complement.

```
((x v y') ^ ((( x ^ y) v (x' ^ y)) v (x' ^ y'))) =
  ((x ^ y) v (x' ^ y'))
```

MACE (a model generation program of McCune) found a size-12 example. This example was very useful to Pavicic and Megill [Pavicic1999]. More generally, the Argonne paradigm that emphasizes the use of strategy, conclusion retention, reliance on the clause language, and vast amounts of experimentation is triumphing.

I think one can say with certainty that Hilbert would have found great pleasure in the successes of our field, not the least of which are those concerning proof simplification. Hilbert's recently discovered twenty-fourth problem may stimulate experimentation and research, *within* and *without* automated reasoning. More open questions are needed, more proofs to refine, more challenges to address. I for one would enjoy receiving such, preferably from mathematics or logic.

Joerg, the field is winning and, perhaps even more important, the second derivative of progress is positive. As for the future, the field will never lose sight of the goal, that of proving ever deeper theorems and contributing to design and verification in other disciplines. Finally, researchers have access to programs that often function as automated reasoning colleagues.

References

[Belinfante2001] Belinfante, J., Computer Assisted Proofs in Set Theory, Web site http://www.math.gatech.edu/ belinfan/research/autoreas/index.html, 2001.

[Epstein1995] Epstein, R., *Propositional Logics: The Semantical Foundations of Logic*, Oxford University Press, Oxford, 1995.

[Ernst2001a] Ernst, Z., "A Concise Axiomatization of RM→", *Bulletin of the Section of Logic*, to appear.

[Ernst2001b] Ernst, Z., Fitelson, B., Harris, K., and Wos, L., "Shortest Axiomatizations of Implicational S4 and S5", Preprint ANL/MCS-P919-1201, December 2001.

[Fitelson2001] Fitelson, B., and Wos, L. " Missing Proofs Found", *J. Automated Reasoning* **27**, no. 2 (August 2001) 201–225.

[Harris2001] Harris, K., and Fitelson, B. "Distributivity in L-Aleph-0 and other Sentential Logics", *J. Automated Reasoning* **27** (2001) 141–156.

[Henkin1971] Henkin, L., Monk, J. D., and Tarski, A., *Cylindric Algebras I*, North-Holland, Amsterdam, 1971.

[Kunen1992] Kunen, K., "Single Axioms for Groups", *J. Automated Reasoning* **9**, no. 3 (December 1992) 291–308.

[Lukasiewicz1970] Lukasiewicz, J., *Selected Works*, edited by L. Borowski, North Holland, Amsterdam, 1970.

[Lusk1987] Lusk, E., and McFadden, R., "Using Automated Reasoning Tools: A Study of the Semigroup F_2B_2", *Semigroup Forum* **36**, no. 1 (1987) 75–88.

[McCune1987] McCune, W., and Wos, L., "A Case Study in Automated Theorem Proving: Finding Sages in Combinatory Logic", *J. Automated Reasoning* **3**, no. 1 (February 1987) 91–108.

[McCune1993] McCune, W., "Single Axioms for Groups and Abelian Groups with Various Operations", *J. Automated Reasoning* **10**, no. 1 (1993) 1–13.

[McCune1996] McCune, M., and Padmanabhan, R., *Automated Deduction in Equational Logic and Cubic Curves*, *Lectures Notes in Computer Science 1095*, Springer-Verlag, New York, 1996.

[McCune1997] McCune, W., "Solution of the Robbins Problem", *J. Automated Reasoning* **19**, no. 3 (1997) 263–276.

[McCune1998] McCune, W., "Automatic Proofs and Counterexamples for Some Ortholattice Identities", *Information Processing Letters* **65** (1998) 285–291.

[McCune2001] McCune, W., Veroff, R., Fitelson, B., Harris, K., Feist, A., and Wos, L., "Short Single Axioms for Boolean Algebra", *J. Automated Reasoning* (accepted for publication).

[Meredith1953] Meredith, C. A., "Single Axioms for the Systems ⟨C,N⟩, ⟨C,O⟩, and ⟨A,N⟩ of the Two-Valued Propositional Calculus", *J. Computing Systems* **1**, no. 3 (1953), 155–164.

[Meredith1963] Meredith, C. A., and Prior, A., "Notes on the Axiomatics of the Propositional Calculus", *Notre Dame J. Formal Logic* **4**, no. 3 (1963) 171–187.

[Pavicic1999] Pavicic, M., and Megill, N., "Non-Orthomodular Models for Both Standard Quantum Logic and Standard Classical Logic: Repercussions for Quantum Computers", *Helv. Phys. Acta* **72**, no. 3 (1999) 189–210.

[Thiele2001] Thiele, R., and Wos, L., "Hilbert's Twenty-Fourth Problem", Preprint ANL/MCS-P899-0801, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 2001.

[Veroff2001a] Veroff, R., "Finding Shortest Proofs: An Application of Linked Inference Rules", *J. Automated Reasoning* **27**, no. 2 (August 2001) 123–139.

[Veroff2001b] Veroff., "Solving Open Questions and Other Challenge Problems Using Proof Sketches", *J. Automated Reasoning* **27**, no. 2 (August 2001) 157–174.

[Winker1978] Winker, S., and Wos, L., "Automated Generation of Models and Counterexamples and Its Application to Open Questions in Ternary Boolean Algebra", in *Proceedings of the Eighth International Symposium on Multiple-Valued Logic, Rosemont, Illinois,* IEEE and ACM, May 1978, pp. 251–256; reprinted in [Wos2000, pp. 286–297].

[Winker1981] Winker, S., Wos, L., and Lusk, E., "Semigroups, Antiautomorphisms, and Involutions: A Computer Solution to an Open Problem, I" *Mathematics of Computation* **37**, no. 156 (October 1981) 533–545 (October 1981); reprinted in [Wos2000, pp. 315–329].

[Wos1993] Wos, L., "The Kernel Strategy and Its Use for the Study of Combinatory Logic", *J. Automated Reasoning* **10**, no. 3 (1993) 287–343; reprinted in [Wos2000, pp. 1221–1287].

[Wos1999] Wos, L., and Pieper, G. W., *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning,* World Scientific, Singapore, 1999.

[Wos2000] Wos, L., and Pieper, G. W., *Collected Works of Larry Wos*, 2 vols., World Scientific, 2000.

Description Logics as Ontology Languages for the Semantic Web

Franz Baader[1], Ian Horrocks[2], and Ulrike Sattler[1]

[1] Theoretical Computer Science, RWTH Aachen, Germany
{baader,sattler}@cs.rwth-aachen.de
[2] Department of Computer Science, University of Manchester, UK
horrocks@cs.man.ac.uk

Abstract. The vision of a Semantic Web has recently drawn considerable attention, both from academia and industry. Description logics are often named as one of the tools that can support the Semantic Web and thus help to make this vision reality.
In this paper, we describe what description logics are and what they can do for the Semantic Web. Descriptions logics are very useful for defining, integrating, and maintaining ontologies, which provide the Semantic Web with a common understanding of the basic semantic concepts used to annotate Web pages. We also argue that, without the last decade of basic research in this area, description logics could not play such an important rôle in this domain.

1 Introduction

The goal of this introduction is to sketch, on an informal level, what the Semantic Web is, why it needs ontologies, and where description logics come into play. Regarding the last point, we will first give a brief introduction to description logics, and then argue why they are well-suited as ontology languages. The remainder of this paper will then put some flesh on this skeleton by providing more technical details.

The Semantic Web and Ontologies

For many people, the World Wide Web has become an indispensable means of providing and searching for information. Searching the Web in its current form is, however, often an infuriating experience since today's search engines usually provide a huge number of answers, many of which are completely irrelevant, whereas some of the more interesting answers are not found. One of the reasons for this unsatisfactory state of affairs is that existing Web resources are usually only human understandable: the mark-up (HTML) only provides rendering information for textual and graphical information intended for human consumption.

The Semantic Web [15] aims for machine-understandable Web resources, whose information can then be shared and processed both by automated tools, such as search engines, and by human users. In the following we will refer to consumers of Web resources, whether automated tools or human users, as agents.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 228–248, 2005.

This sharing of information between different agents requires semantic mark-up, i.e., an annotation of the Web page with information on its content that is understood by the agents searching the Web. Such an annotation will be given in some standardized, expressive language (which, e.g., provides Boolean operators and some form of quantification) and make use of certain terms (like "Human", "Plant", etc.). To make sure that different agents have a common understanding of these terms, one needs *ontologies* in which these terms are described, and which thus establish a joint terminology between the agents. Basically, an ontology [44, 43] is a collection of definitions of concepts and the shared understanding comes from the fact that all the agents interpret the concepts w.r.t. the same ontology.

The use of ontologies in this context requires a well-designed, well-defined, and Web-compatible ontology language with supporting reasoning tools. The syntax of this language should be both intuitive to human users and compatible with existing Web standards (such as XML, RDF, and RDFS). Its semantics should be formally specified since otherwise it could not provide a shared understanding. Finally, its expressive power should be adequate, i.e., the language should be expressive enough for defining the relevant concepts in enough detail, but not too expressive to make reasoning infeasible.

Reasoning is important to ensure the quality of an ontology. It can be employed in different development phases. During ontology design, it can be used to test whether concepts are non-contradictory and to derive implied relations. In particular, one usually wants to compute the concept hierarchy. Information on which concept is a specialization of another and which concepts are synonyms can be used in the design phase to test whether the concept definitions in the ontology have the intended consequences or not. Moreover, this information is also useful when searching Web pages annotated with such concepts. Since it is not reasonable to assume that there will be a single ontology for the whole Web, interoperability and integration of different ontologies is also an important issue. Integration can, for example, be supported by asserting inter-ontology relationships and testing for consistency and computing the integrated concept hierarchy. Finally, reasoning may also be used when the ontology is deployed, i.e., when a Web page is already annotated with its concepts. One can, for example, determine the consistency of facts stated in the annotation with the ontology or infer instance relationships. However, in the deployment phase, the requirements on the efficiency of reasoning are much more stringent than in the design and integration phases.

Before arguing why description logics are good candidates for such an ontology language, we provide a brief introduction to and history of description logics.

Description Logics

Description logics (DLs) [7, 24] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. The name *description logics* is

motivated by the fact that, on the one hand, the important notions of the domain are described by concept *descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. On the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, *logic*-based semantics.

In this introduction, we only illustrate some typical constructors by an example. Formal definitions are given in Section 2. Assume that we want to define the concept of "A man that is married to a doctor and has at least five children, all of whom are professors." This concept can be described with the following concept description:

$$\mathsf{Human} \sqcap \neg\mathsf{Female} \sqcap \exists\mathsf{married}.\mathsf{Doctor} \sqcap (\geq 5\,\mathsf{hasChild}) \sqcap \forall\mathsf{hasChild}.\mathsf{Professor}$$

This description employs the Boolean constructors *conjunction* ($\sqcap$), which is interpreted as set intersection, and *negation* ($\neg$), which is interpreted as set complement, as well as the *existential restriction* constructor ($\exists R.C$), the *value restriction* constructor ($\forall R.C$), and the *number restriction* constructor ($\geq n\,R$). An individual, say Bob, belongs to $\exists\mathsf{married}.\mathsf{Doctor}$ iff there exists an individual that is married to Bob (i.e., is related to Bob via the married role) and is a doctor (i.e., belongs to the concept Doctor). Similarly, Bob belongs to $(\geq 5\,\mathsf{hasChild})$ iff he has at least five children, and he belongs to $\forall\mathsf{hasChild}.\mathsf{Professor}$ iff all his children (i.e., all individuals related to Bob via the hasChild role) are professors.

In addition to this description formalism, DLs are usually equipped with a terminological and an assertional formalism. In its simplest form, *terminological axioms* can be used to introduce names (abbreviations) for complex descriptions. For example, we could introduce the abbreviation HappyMan for the concept description from above. More expressive terminological formalisms allow the statement of constraints such as

$$\exists\mathsf{hasChild}.\mathsf{Human} \sqsubseteq \mathsf{Human},$$

which says that only humans can have human children. The *assertional formalism* can be used to state properties of individuals. For example, the assertions

$$\mathsf{HappyMan}(\mathsf{BOB}), \quad \mathsf{hasChild}(\mathsf{BOB}, \mathsf{MARY})$$

state that Bob belongs to the concept HappyMan and that Mary is one of his children.

Description logic systems provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. The *subsumption* algorithm determines subconcept-superconcept relationships: C is subsumed by D iff all instances of C are necessarily instances of D, i.e., the first description is always interpreted as a subset of the second description. For example, given the definition of HappyMan from above, HappyMan is subsumed by $\exists\mathsf{hasChild}.\mathsf{Professor}$ – since instances of HappyMan have at least five children, all of whom are professors, they also have a child that is a professor.

The *instance* algorithm determines instance relationships: the individual i is an instance of the concept description C iff i is always interpreted as an element of C. For example, given the assertions from above and the definition of HappyMan, MARY is an instance of Professor. The *consistency* algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is non-contradictory. For example, if we add ¬Professor(MARY) to the two assertions from above, then the knowledge base containing these assertions together with the definition of HappyMan from above is inconsistent.

In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable for the DL employed by the system, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues in DL research. Roughly, the research related to this issue can be classified into the following four phases.

Phase 1 (1980–1990) was mainly concerned with implementation of systems, such as KLONE, K-REP, BACK, and LOOM [19, 61, 70, 60]. These systems employed so-called *structural subsumption algorithms*, which first normalize the concept descriptions, and then recursively compare the syntactic structure of the normalized descriptions [62]. These algorithms are usually very efficient (polynomial), but they have the disadvantage that they are complete only for very inexpressive DLs, i.e., for more expressive DLs they cannot detect all the existing subsumption/instance relationships. At the end of this phase, early formal investigations into the complexity of reasoning in DLs showed that most DLs do not have polynomial-time inference problems [18, 63]. As a reaction, the implementors of the CLASSIC system (the first industrial-strength DL system) carefully restricted the expressive power of their DL [69, 17].

Phase 2 (1990–1995) started with the introduction of a new algorithmic paradigm into DLs, so-called *tableau-based algorithms* [75, 32, 48]. They work on propositionally closed DLs (i.e., DLs with full Boolean operators) and are complete also for expressive DLs. To decide the consistency of a knowledge base, a tableau-based algorithm tries to construct a model of it by breaking down the concepts in the knowledge base, thus inferring new constraints on the elements of this model. The algorithm either stops because all attempts to build a model failed with obvious contradictions, or it stops with a "canonical" model. Since in propositionally closed DLs subsumption and satisfiability can be reduced to consistency, a consistency algorithm can solve all inference problems mentioned above. The first systems employing such algorithms (KRIS and CRACK) demonstrated that optimized implementations of these algorithms lead to an acceptable behavior of the system, though the worst-case complexity of the corresponding inference problem is no longer in polynomial time [6, 20]. This phase also saw a thorough analysis of the complexity of reasoning in various DLs [32–34]. Another important observation was that DLs are very closely related to modal logics [73].

Phase 3 (1995–2000) is characterized by the development of inference procedures for very expressive DLs, either based on the tableau-approach [56, 57] or on a translation into modal logics [29, 30, 28, 31]. Highly optimized systems (FaCT, Race, and Dlp [55, 45, 68]) showed that tableau-based algorithm for expressive DLs lead to a good practical behavior of the system even on (some) large knowledge bases. In this phase, the relationship to modal logics [29, 74] and to decidable fragments of first-order logic was also studied in more detail [16, 66, 42, 40, 41], and applications in databases (like schema reasoning, query optimization, and DB integration) were investigated [21, 22, 25, 26].

We are now at the beginning of *Phase 4*, where industrial strength DL systems employing very expressive DLs and tableau-based algorithms are being developed, with applications like the Semantic Web or knowledge representation and integration in bio-informatics in mind.

Description Logics as Ontology Languages

As already mentioned above, high quality ontologies are crucial for the Semantic Web, and their construction, integration, and evolution greatly depends on the availability of a well-defined semantics and powerful reasoning tools. Since DLs provide for both, they should be ideal candidates for ontology languages. That much was already clear ten years ago, but at that time, there was a fundamental mismatch between the expressive power and the efficiency of reasoning that DL systems provided, and the expressivity and the large knowledge bases that ontologists needed [35]. Through the basic research in DLs of the last 10–15 years that we have summarized above, this gap between the needs of ontologist and the systems that DL researchers provide has finally become narrow enough to build stable bridges.

Regarding an ontology language for the Semantic Web, there is a joint US/EU initiative for a W3C ontology standard, for historical reasons called DAML+OIL [52, 27]. This language has a syntax based on RDF Schema (and thus is Web compatible), and it is based on common ontological primitives from Frame Languages (which supports human understandability). Its semantics can be defined by a translation into the expressive DL $\mathcal{SHIQ}$ [54][1], and the developers have tried to find a good compromise between expressiveness and the complexity of reasoning. Although reasoning in $\mathcal{SHIQ}$ is decidable, it has a rather high worst-case complexity (ExpTime). Nevertheless, there is a highly optimized $\mathcal{SHIQ}$ reasoner (FaCT) available, which behaves quite well in practice.

Let us point out some of the features of $\mathcal{SHIQ}$ that make this DL expressive enough to be used as an ontology language. Firstly, $\mathcal{SHIQ}$ provides number restrictions that are more expressive than the ones introduced above (and employed be earlier DL systems). With the *qualified number restrictions* available in $\mathcal{SHIQ}$, as well as being able to say that a person has at most two children (without mentioning the properties of these children):

$$(\leq 2\,\mathsf{hasChild}),$$

[1] To be exact, the translation is into an extension of $\mathcal{SHIQ}$.

one can also specify that there is at most one son and at most one daughter:

$$(\leq 1\,\mathsf{hasChild}.\neg\mathsf{Female}) \sqcap (\leq 1\,\mathsf{hasChild}.\mathsf{Female})$$

Secondly, $\mathcal{SHIQ}$ allows the formulation of complex terminological axioms like "humans have human parents":

$$\mathsf{Human} \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Human}.$$

Thirdly, $\mathcal{SHIQ}$ also allows for *inverse roles*, *transitive roles*, and *subroles*. For example, in addition to hasChild one can also use its inverse hasParent, one can specify that hasAncestor is transitive, and that hasParent is a subrole of hasAncestor.

It has been argued in the DL and the ontology community that these features play a central role when describing properties of aggregated objects and when building ontologies [72, 76, 37]. The actual use of DLs providing these features as the underlying logical formalism of the web ontology languages OIL and DAML+OIL [36, 52] substantiates this claim [76].

2 The Expressive Description Logic $\mathcal{SHIQ}$

In contrast to most of the DLs considered in the literature, which concentrate on constructors for defining concepts, the DL $\mathcal{SHIQ}$ [53] also allows for rather expressive roles. Of course, these roles can then be used in the definition of concepts. We start with the definition of $\mathcal{SHIQ}$-roles, and then continue with the definition of $\mathcal{SHIQ}$-concepts.

Definition 1 (Syntax and semantics of $\mathcal{SHIQ}$-roles). *Let* $\mathbf{R}$ *be a set of* role names*, which is partitioned into a set* $\mathbf{R}_+$ *of transitive roles and a set* $\mathbf{R}_\mathsf{P}$ *of normal roles. The set of all* $\mathcal{SHIQ}$-roles *is* $\mathbf{R} \cup \{r^- \mid r \in \mathbf{R}\}$*, where* r^- *is called the* inverse *of the role* r*. A* role inclusion axiom *is of the form* $r \sqsubseteq s$*, where* r, s *are* $\mathcal{SHIQ}$*-roles. A* role hierarchy *is a finite set of role inclusion axioms.*

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a set* $\Delta^\mathcal{I}$*, called the* domain *of* $\mathcal{I}$*, and a function* $\cdot^\mathcal{I}$ *that maps every role to a subset of* $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ *such that, for all* $p \in \mathbf{R}$ *and* $r \in \mathbf{R}_+$*,*

$$\langle x, y\rangle \in p^\mathcal{I} \quad \textit{iff} \quad \langle y, x\rangle \in (p^-)^\mathcal{I},$$
$$\textit{if } \langle x, y\rangle \in r^\mathcal{I} \textit{ and } \langle y, z\rangle \in r^\mathcal{I} \textit{ then } \langle x, z\rangle \in r^\mathcal{I}.$$

An interpretation $\mathcal{I}$ satisfies *a role hierarchy* $\mathcal{R}$ *iff* $r^\mathcal{I} \subseteq s^\mathcal{I}$ *for each* $r \sqsubseteq s \in \mathcal{R}$*; such an interpretation is called a* model *of* $\mathcal{R}$*.*

The unrestricted use of these roles in all of the concept constructors of $\mathcal{SHIQ}$ (to be defined below) would lead to an undecidable DL [53]. Therefore, we must first define an appropriate subset of all $\mathcal{SHIQ}$-roles. This requires some more notation.

1. The inverse relation on binary relations is symmetric, i.e., the inverse of r^- is again r. To avoid writing role expressions such as r^{--}, r^{---}, etc., we define a function Inv, which returns the inverse of a role:
$$\mathsf{Inv}(r) := \begin{cases} r^- & \text{if } r \text{ is a role name,} \\ s & \text{if } r = s^- \text{ for a role name } s. \end{cases}$$
2. Since set inclusion is transitive and an inclusion relation between two roles transfers to their inverses, a given role hierarchy $\mathcal{R}$ implies additional inclusion relationships. To account for this fact, we define $\sqsubseteq^*_{\mathcal{R}}$ as the reflexive-transitive closure of
$$\sqsubseteq_{\mathcal{R}} := \mathcal{R} \cup \{\mathsf{Inv}(r) \sqsubseteq \mathsf{Inv}(s) \mid r \sqsubseteq s \in \mathcal{R}\}.$$
We use $r \equiv_{\mathcal{R}} s$ as an abbreviation for $r \sqsubseteq^*_{\mathcal{R}} s$ and $s \sqsubseteq^*_{\mathcal{R}} r$. In this case, every model of $\mathcal{R}$ interprets these roles as the same binary relation.
3. Obviously, a binary relation is transitive iff its inverse is transitive. Thus, if $r \equiv_{\mathcal{R}} s$ and r or $\mathsf{Inv}(r)$ is transitive, then any model of $\mathcal{R}$ interprets s as a transitive binary relation. To account for such implied transitive roles, we define the following function Trans:
$$\mathsf{Trans}(s, \mathcal{R}) := \begin{cases} \text{true} & \text{if } r \in \mathbf{R}_+ \text{ or } \mathsf{Inv}(r) \in \mathbf{R}_+ \text{ for some } r \text{ with } r \equiv_{\mathcal{R}} s \\ \text{false} & \text{otherwise.} \end{cases}$$
4. A role r is called *simple* w.r.t. $\mathcal{R}$ iff $\mathsf{Trans}(s, \mathcal{R}) = \text{false}$ for all $s \sqsubseteq^*_{\mathcal{R}} r$.

Definition 2 (Syntax and semantics of $\mathcal{SHIQ}$-concepts). *Let N_C be a set of* concept names. *The set of $\mathcal{SHIQ}$-*concepts *is the smallest set such that*

1. *every concept name $A \in N_C$ is a $\mathcal{SHIQ}$-concept,*
2. *if C and D are $\mathcal{SHIQ}$-concepts and r is a $\mathcal{SHIQ}$-role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, and $\exists r.C$ are $\mathcal{SHIQ}$-concepts,*
3. *if C is a $\mathcal{SHIQ}$-concept, r is a* simple *$\mathcal{SHIQ}$-role, and $n \in \mathbb{N}$, then $(\leqslant n\, r.C)$ and $(\geqslant n\, r.C)$ are $\mathcal{SHIQ}$-concepts.*

The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \qquad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},$$
$$(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \textit{There is some } y \in \Delta^{\mathcal{I}} \textit{ with } \langle x, y\rangle \in r^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\},$$
$$(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \textit{For all } y \in \Delta^{\mathcal{I}}, \textit{ if } \langle x, y\rangle \in r^{\mathcal{I}}, \textit{then } y \in C^{\mathcal{I}}\},$$
$$(\leqslant n\, r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp r^{\mathcal{I}}(x, C) \leqslant n\},$$
$$(\geqslant n\, r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp r^{\mathcal{I}}(x, C) \geqslant n\},$$

where $\sharp M$ denotes the cardinality of the set M, and $r^{\mathcal{I}}(x, C) := \{y \mid \langle x, y\rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$. If $x \in C^{\mathcal{I}}$, then we say that x is an instance *of C in $\mathcal{I}$, and if $\langle x, y\rangle \in r^{\mathcal{I}}$, then y is called an* r-successor *of x in $\mathcal{I}$.*

Concepts can be used to describe the relevant notions of an application domain. The terminology (TBox) introduces abbreviations (names) for complex concepts. In $\mathcal{SHIQ}$, the TBox allows one to state also more complex constraints.

Definition 3. *A* general concept inclusion *(GCI) is of the form $C \sqsubseteq D$, where C, D are $\mathcal{SHIQ}$-concepts. A finite set of GCIs is called a* TBox. *An interpretation $\mathcal{I}$ is a* model *of a TBox $\mathcal{T}$ iff it satisfies all GCIs in $\mathcal{T}$, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each $C \sqsubseteq D \in \mathcal{T}$.*

A concept definition is of the form $A \equiv C$, where A is a concept name. It can be seen as an abbreviation for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$.

Inference problems are defined w.r.t. a TBox and a role hierarchy.

Definition 4. *The concept C is called* satisfiable with respect to the role hierarchy $\mathcal{R}$ and the TBox $\mathcal{T}$ *iff there is a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model of C *w.r.t. $\mathcal{R}$ and $\mathcal{T}$. The concept D* subsumes *the concept C w.r.t. $\langle \mathcal{R}, \mathcal{T} \rangle$ (written $C \sqsubseteq_{\langle \mathcal{R}, \mathcal{T} \rangle} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$. Two concepts C, D are* equivalent *w.r.t. $\mathcal{R}$ (written $C \equiv_{\langle \mathcal{R}, \mathcal{T} \rangle} D$) iff they subsume each other.*

By definition, equivalence can be reduced to subsumption. In addition, subsumption can be reduced to satisfiability since $C \sqsubseteq_{\langle \mathcal{R}, \mathcal{T} \rangle} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$. Before sketching how to solve the satisfiability problem in $\mathcal{SHIQ}$, we try to give an intuition on how $\mathcal{SHIQ}$ can be used to define ontologies.

3 Describing Ontologies in $\mathcal{SHIQ}$

In general, an ontology can be formalised in a TBox as follows. Firstly, we restrict the possible worlds by introducing restrictions on the allowed interpretations. For example, to express that, in our world, we want to consider humans, which are either muggles or sorcerers, we can use the GCIs

$$\mathsf{Human} \sqsubseteq \mathsf{Muggle} \sqcup \mathsf{Sorcerer} \quad \text{and} \quad \mathsf{Muggle} \sqsubseteq \neg \mathsf{Sorcerer}.$$

Next, to express that humans have exactly two parents and that all parents and children of humans are human, we can use the following GCI:

$$\mathsf{Human} \sqsubseteq \forall \mathsf{hasParent}.\mathsf{Human} \sqcap (\leqslant 2\ \mathsf{hasParent}.\top) \sqcap (\geqslant 2\ \mathsf{hasParent}.\top) \sqcap \forall \mathsf{hasParent}^{-}.\mathsf{Human},$$

where $\top$ is an abbreviation for the top concept $A \sqcup \neg A$.

In addition, we consider the *transitive* role hasAncestor, and the role inclusion

$$\mathsf{hasParent} \sqsubseteq \mathsf{hasAncestor}.$$

The next GCI expresses that humans having an ancestor that is a sorcerer are themselves sorcerers:

$$\mathsf{Human} \sqcap \exists \mathsf{hasAncestor}.\mathsf{Sorcerer} \sqsubseteq \mathsf{Sorcerer}.$$

Secondly, we can define the relevant notions of our application domain using concept definitions. Recall that the concept definition $A \equiv C$ stands for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. A concept name is called *defined* if it occurs on the left-hand side of a definition, and *primitive* otherwise.

We want our concept definitions to have definitional impact, i.e., the interpretation of the primitive concept and role names should uniquely determine the interpretation of the defined concept names. For this, the set of concept definitions together with the additional GCIs must satisfy three conditions:

1. There are no multiple definitions, i.e., each defined concept name must occur at most once as a left-hand side of a concept definition.
2. There are no cyclic definitions, i.e., no cyclic dependencies between the defined names in the set of concept definitions[2].
3. The defined names do not occur in any of the additional GCIs.

In contrast to concept definitions, the GCIs in $\mathcal{SHIQ}$ may well have cyclic dependencies between concept names. An example are the above GCIs describing humans.

As a simple example of a set of concept definitions satisfying the restrictions from above, we define the concepts grandparent and parent[3]:

$$\mathsf{Parent} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasParent}^{-}.\top,$$
$$\mathsf{Grandparent} \equiv \exists \mathsf{hasParent}^{-}.\mathsf{Parent},$$

The TBox consisting of the above concept definitions and GCIs, together with the fact that hasAncestor is a transitive superrole of hasParent, implies the following subsumption relationship:

$$\mathsf{Grandparent} \sqcap \mathsf{Sorcerer} \sqsubseteq \exists \mathsf{hasParent}^{-}.\exists \mathsf{hasParent}^{-}.\mathsf{Sorcerer},$$

i.e., grandparents that are sorcerers have a grandchild that is a sorcerer. Though this conclusion may sound reasonable given the assumptions, it requires quite some reasoning to obtain it. In particular, one must use the fact that hasAncestor (and thus also $\mathsf{hasAncestor}^{-}$) is transitive, that $\mathsf{hasParent}^{-}$ is the inverse of hasParent, and that we have a GCI that says that children of humans are again humans.

To sum up, a $\mathcal{SHIQ}$-TBox can, on the one hand, axiomatize the basic notions of an application domain (the primitive concepts) by GCIs, transitivity statements, and role inclusions, in the sense that these statements restrict the possible interpretations of the basic notions. On the other hand, more complex notions (the defined concepts) can be introduced by concept definitions. Given an interpretation of the basic notions, the concept definitions uniquely determine the interpretation of the defined notions.

[2] In order to give cyclic definitions definitional impact, one would need to use fixpoint semantics for them [64, 2].

[3] In addition to the role hasParent, which relates children to their parents, we use the concept Parent, which describes all humans having children.

The *taxonomy* of such a TBox is then given by the subsumption hierarchy of the defined concepts. It can be computed using a subsumption algorithm for $\mathcal{SHIQ}$ (see Section 5 below). The knowledge engineer can test whether the TBox captures her intuition by checking the satisfiability of the defined concepts (since it does not make sense to give a complex definition for the empty concept), and by checking whether their place in the taxonomy corresponds to their intuitive place. The expressive power of $\mathcal{SHIQ}$ together with the fact that one can "verify" the TBox in the sense mentioned above is the main reason for $\mathcal{SHIQ}$ being well-suited as an ontology language [72, 37, 76].

4 $\mathcal{SHIQ}$ and DAML+OIL

As already discussed, DAML+OIL is a semantic web ontology language whose semantics can be defined via a translation into an expressive DL. This is not a coincidence – it was a design goal. The mapping allows DAML+OIL to exploit formal results from DL research (e.g., regarding the decidability and complexity of key inference problems) and use implemented DL reasoners (e.g., FaCT [50] and Racer [46]) in order to provide reasoning services for DAML+OIL applications.

DAML+OIL uses a syntax that is based on RDF (the *Resource Description Framework*), and thus suitable for the Semantic Web. The underlying model for RDF is a labelled directed graph where nodes are either *resources* or *literals* (currently literals are just strings, but it is planed to extend the language to support type data values, e.g., "integer 5"). The graph is defined by a set of *triples*, statements of the form $\langle \mathsf{Subject}, \mathsf{Property}, \mathsf{Object} \rangle$, where Subject is a resource, Property is the edge label and Object is either a resource or a literal.

Everything describable by RDF is a resource; a resource may be named by a URI, but some resources (we will call them *anonymous resources*) may not be so named. A resource may be an entire Web page (identified by its URL), a part of a Web page (identified by its URL and an anchor), but also an object not accessible through the Web. A property is an attribute or relation used to describe a resource, and is also named by a URI. In practice, triples are written using a standard XML serialisation of RDF triples (see http://www.w3.org/RDF/ for more details).

A DAML+OIL ontology can be seen to correspond to a DL TBox together with a role hierarchy, describing the domain in terms of *classes* (corresponding to concepts) and *properties* (corresponding to roles). An ontology consists of a set of *axioms* that assert, e.g., subsumption relationships between classes or properties. Asserting that an individual resource (a pair of resources) is an instance of a DAML+OIL class (property) is left to RDF, a task for which it is well suited.

As in a standard DLs, DAML+OIL classes may be names or expressions built up from simpler classes and properties using a variety of constructors. The set of constructors supported by DAML+OIL, along with the equivalent DL abstract syntax, is summarised in Figure 1[4]. The full XML serialisation of the

[4] In fact, there are a few additional constructors provided as "syntactic sugar", but all are trivially reducible to the ones described in Figure 1.

RDF syntax is not shown as it is rather verbose, e.g., Human ⊓ Male would be written as

```
<daml:Class>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Human"/>
    <daml:Class rdf:about="#Male"/>
  </daml:intersectionOf>
</daml:Class>
```

while ($\geqslant 2$ hasChild.Lawyer) would be written as

```
<daml:Restriction daml:minCardinalityQ="2">
  <daml:onProperty rdf:resource="#hasChild"/>
  <daml:hasClassQ rdf:resource="#Lawyer"/>
</daml:Restriction>
```

Prefixes such as `daml:` specify XML namespaces for resources, while `rdf:parseType="daml:collection"` is a DAML+OIL extension to RDF that provides a "shorthand" notation for lisp style lists defined using triples with the properties first and rest (it can be eliminated, but with a consequent increase in verbosity). E.g., the first example above consists of the triples $\langle \mathsf{r}_1, \mathsf{daml:intersectionOf}, \mathsf{r}_2\rangle$, $\langle \mathsf{r}_2, \mathsf{daml:first}, \mathsf{Human}\rangle$, $\langle \mathsf{r}_2, \mathsf{rdfs:type}, \mathsf{Class}\rangle$, $\langle \mathsf{r}_2, \mathsf{daml:rest}, \mathsf{r}_3\rangle$, etc., where r_i is an anonymous resource, Human stands for a URI naming the resource "Human", and daml : intersectionOf, daml : first, daml : rest and rdfs : type stand for URIs naming the properties in question.

Constructor	DL Syntax	Example
`intersectionOf`	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
`unionOf`	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
`complementOf`	$\neg C$	¬Male
`oneOf`	$\{x_1 \ldots x_n\}$	{john, mary}
`toClass`	$\forall P.C$	∀hasChild.Doctor
`hasClass`	$\exists r.C$	∃hasChild.Lawyer
`hasValue`	$\exists r.\{x\}$	∃citizenOf.{USA}
`minCardinalityQ`	$(\geqslant n\ r.C)$	($\geqslant 2$ hasChild.Lawyer)
`maxCardinalityQ`	$(\leqslant n\ r.C)$	($\leqslant 1$ hasChild.Male)
`inverseOf`	r^-	hasChild$^-$

Fig. 1. DAML+OIL constructors.

An important feature of DAML+OIL is that, besides "abstract" classes defined by the ontology, one can also use XML Schema *datatypes* (e.g., so called primitive datatypes such as string, decimal or float, as well as more complex derived datatypes such as integer sub-ranges) in `hasClass`, `hasValue`, and `cardinality`. E.g., the class Adult could be asserted to be equivalent to Person ⊓ ∃age.over17, where over17 is an XML Schema datatype based on decimal, but with the added restriction that values must be at least 18. Using a combination of XML Schema and RDF this could be written as:

```
<xsd:simpleType name="over17">
  <xsd:restriction base="xsd:positiveInteger">
  <xsd:minInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>

<daml:Class rdf:ID="Adult">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#age"/>
      <daml:hasClass rdf:resource="#over17"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

As already mentioned, a DAML+OIL ontology consists of a set of axioms. Figure 2 summarises the axioms supported by DAML+OIL. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties, the disjointness of classes, the equivalence or non-equivalence of individuals (resources), and various properties of properties. DAML+OIL also allows properties of properties (i.e., DL roles) to be asserted. In particular, it is possible to assert that a property is unique (i.e., functional), unambiguous (i.e., its inverse is functional) or transitive.

Axiom	DL Syntax	Example
`subClassOf`	$C_1 \sqsubseteq C_2$	$\mathsf{Human} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{Biped}$
`sameClassAs`	$C_1 \equiv C_2$	$\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male}$
`subPropertyOf`	$P_1 \sqsubseteq P_2$	$\mathsf{hasDaughter} \sqsubseteq \mathsf{hasChild}$
`samePropertyAs`	$P_1 \equiv P_2$	$\mathsf{cost} \equiv \mathsf{price}$
`disjointWith`	$C_1 \sqsubseteq \neg C_2$	$\mathsf{Male} \sqsubseteq \neg\mathsf{Female}$
`sameIndividualAs`	$\{x_1\} \equiv \{x_2\}$	$\{\mathsf{President\_Bush}\} \equiv \{\mathsf{G\_W\_Bush}\}$
`differentIndividualFrom`	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$\{\mathsf{john}\} \sqsubseteq \neg\{\mathsf{peter}\}$
`transitiveProperty`	$P \in \mathbf{R}_+$	$\mathsf{hasAncestor}^+ \in \mathbf{R}_+$
`uniqueProperty`	$\top \sqsubseteq (\leqslant 1\ P.\top)$	$\top \sqsubseteq (\leqslant 1\ \mathsf{hasMother}.\top)$
`unambiguousProperty`	$\top \sqsubseteq (\leqslant 1\ P^-.\top)$	$\top \sqsubseteq (\leqslant 1\ \mathsf{isMotherOf}^-.\top)$

Fig. 2. DAML+OIL axioms.

This shows that, except for individuals and datatypes, the constructors and axioms of DAML+OIL can be translated into $\mathcal{SHIQ}$. In fact, DAML+OIL is equivalent to the extension of $\mathcal{SHIQ}$ with nominals (i.e., individuals) and a simple form of so-called concrete domains [5]. This extension will be discussed in Section 6.

5 Reasoning in $\mathcal{SHIQ}$

Reasoning in $\mathcal{SHIQ}$ means deciding satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. TBoxes (i.e., sets of general concept inclusions) and role hierarchies. As shown in Section 2, subsumption can be reduced (in linear time) to satisfiability. In addition, since $\mathcal{SHIQ}$ allows for both subroles and transitive roles, TBoxes can be internalized, i.e., satisfiability w.r.t. a TBox and a role hierarchy can be reduced to satisfiability w.r.t. the empty TBox and a role hierarchy. In principle, this is achieved by introducing a (new) transitive superrole u of all roles occurring in the TBox $\mathcal{T}$ and the concept C_0 to be tested for satisfiability. Then we extend C_0 to the concept

$$\widehat{C}_0 := C_0 \sqcap \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D) \sqcap \forall u.(\neg C \sqcup D).$$

We can then show that $\widehat{C}_0$ is satisfiable w.r.t. the extended role hierarchy iff the original concept C_0 is satisfiable w.r.t. the TBox $\mathcal{T}$ and the original role hierarchy [1, 73, 3, 53].

Consequently, it is sufficient to design an algorithm that can decide satisfiability of $\mathcal{SHIQ}$-concepts w.r.t. role hierarchies and transitive roles. This problem is known to be ExpTime-complete [77]. In fact, ExpTime-hardness can be shown by an easy adaptation of the ExpTime-hardness proof for satisfiability in propositional dynamic logic [38]. Using automata-based techniques, Tobies [77] shows that satisfiability of $\mathcal{SHIQ}$-concepts w.r.t. role hierarchies is indeed decidable within exponential time.

In the remainder of this section, we sketch a tableau-based decision procedure for this problem. This procedure, which is described in more detail in [53], runs in worst case *nondeterministic* double exponential time. However, according to the current state of the art, this procedures is more practical than the ExpTime automata-based procedure in [77]. In fact, it is the basis for the highly optimised implementation of the DL system FaCT [51].

When started with a $\mathcal{SHIQ}$-concept C_0, a role hierarchy $\mathcal{R}$, and information on which roles are transitive, this algorithm tries to construct a model of C_0 w.r.t. $\mathcal{R}$. Since $\mathcal{SHIQ}$ has a so-called tree model property, we can assume that this model has the form of an infinite tree. If we want to obtain a decision procedure, we can only construct a finite tree representing the infinite one (if a (tree) model exists at all). This can be done such that the finite representation can be *unravelled* into an infinite tree model $\mathcal{I}$ of C_0 w.r.t. $\mathcal{R}$. In the finite tree representing this model, a node x corresponds to an individual $\pi(x) \in \Delta^{\mathcal{I}}$, and we label each node with the set of concepts $\mathcal{L}(x)$ that $\pi(x)$ is supposed to be an instance of. Similary, edges represent role-successor relationships, and an edge between x and y is labelled with the roles supposed to connect x and y. The algorithm either stops with a finite representation of a tree model, or with a *clash*, i.e., an obvious inconsistency, such as $\{C, \neg C\} \subseteq \mathcal{L}(x)$. It answers "$C_0$ is satisfiable w.r.t. $\mathcal{R}$" in the former case, and "C_0 is unsatisfiable w.r.t. $\mathcal{R}$" in the latter.

The algorithm is initialised with the tree consisting of a single node x labelled with $\mathcal{L}(x) = \{C_0\}$. Then it applies so-called *completion rules*, which break down the concepts in the node labels syntactically, thus inferring new constraints for the given node, and then extend the tree according to these constraints. For example, if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, then the $\sqcap$-rule adds both C_1 and C_2 to $\mathcal{L}(x)$. The $\geq$-rule generates n new r-successor nodes $y_1, \ldots, y_n$ of x with $\mathcal{L}(y_i) = \{C\}$ if $(\geqslant n\ r.C) \in \mathcal{L}(x)$ and x does not yet have n distinct r-successors with C in their label. In addition, it asserts that these new successors must remain distinct (i.e., cannot be identified in later steps of the algorithm). Other rules are more complicated, and a complete description of this algorithm goes beyond the scope of this paper. However, we would like to point out two issues that make reasoning in $\mathcal{SHIQ}$ considerably harder than in less expressive DLs.

First, *qualified number restriction* are harder to handle than the unqualified ones used in most early DL systems. Let us illustrate this by an example. Assume that the algorithm has generated a node x with $(\leqslant 1\ \mathsf{hasChild}.\top) \in \mathcal{L}(x)$, and that this node has two $\mathsf{hasChild}$-successors y_1, y_2 (i.e., two edges labeled with $\mathsf{hasChild}$ leading to the nodes y_1, y_2). In order to satisfy the number restriction $(\leqslant 1\ \mathsf{hasChild}.\top)$ for x, the algorithm identifies node y_1 with node y_2 (unless these nodes were asserted to be distinct, in which case we have a clash). Now assume that we still have a node x with two $\mathsf{hasChild}$-successors y_1, y_2, but the label of x contains a qualified number restriction like $(\leqslant 2\ \mathsf{hasChild}.\mathsf{Parent})$. The naive idea [78] would be to check the labels of y_1 and y_2 whether they contain Parent, and identify y_1 and y_2 only if both contain this concept. However, this is not correct since, in the model $\mathcal{I}$ constructed from the tree, $\pi(y_i)$ may well belong to $\mathsf{Parent}^{\mathcal{I}}$ even if this concept does not belong to the label of x. The first correct algorithm that can handle qualified number restrictions was proposed in [49]. The main idea is to introduce a so-called *choose-rule*. In our example, this rule would (nondeterministically) choose whether y_i is supposed to belong to Parent or $\neg\mathsf{Parent}$, and correspondingly extend its label. Together with the choose rule, the above naive identification rule is in fact correct.

Second, in the presence of *transitive roles*, guaranteeing termination of the algorithm is a non-trivial task [47, 71]. If $\forall r.C \in \mathcal{L}(x)$ for a transitive role r, then not only must we add C to the label of any r-successor y of x, but also $\forall r.C$. This ensures that, even over an "r-chain"

$$x \xrightarrow{r} y \xrightarrow{r} y_1 \xrightarrow{r} y_2 \xrightarrow{r} \ldots \xrightarrow{r} y_n$$

we get indeed $C \in \mathcal{L}(y_n)$. This is necessary since, in the model constructed from the tree generated by the algorithm, have

$$(\pi(x), \pi(y)),\ (\pi(y), \pi(y_1)),\ \ldots,\ (\pi(y_{n-1}), \pi(y_n)) \in r^{\mathcal{I}},$$

and thus the transitivity of $r^{\mathcal{I}}$ requires that also $(\pi(x), \pi(y_n)) \in r^{\mathcal{I}}$, and thus the value restriction on x applies to y_n as well. Propagating $\forall r.C$ over r-edges makes sure that this is taken care of. However, it also might lead to nontermination. For example, consider the concept $\exists r.A \sqcap \forall r.\exists r.A$ where r is a transitive role. It is easy to see that the algorithm then generates an infinite chain of nodes

with label $\{A, \forall r.\exists r.A, \exists r.A\}$. To prevent this looping and ensure termination, we use a cycle-detection mechanism called *blocking*: if the labels of a node x and one of its ancestors coincide, we "block" the application of rules to x. The blocking condition must be formulated such that, whenever blocking occurs, we can "unravel" the blocked (finite) path into an infinite path in the model to be constructed. In description logics, blocking was first employed in [8] in the context of an algorithm that can handle GCIs, and was the improved on in [4, 23, 9]. In $\mathcal{SHIQ}$, the blocking condition is rather complicated since the combination of transitive and inverse roles r^- with number restrictions requires a rather advanced form of unravelling [53]. In fact, this combination of constructors is responsible for the fact that, unlike most DLs considered in the literature, $\mathcal{SHIQ}$ does not have the finite model property, i.e., there are satisfiable $\mathcal{SHIQ}$-concepts that are only satisfiable in infinite interpretations.

6 Extensions and Variants of $\mathcal{SHIQ}$

As mentioned in Section 4, the ontology language DAML+OIL is a syntactic variant of $\mathcal{SHIQ}$ extended with nominals (i.e., concepts $\{x_1\}$ representing a singleton set consisting of one individual) and concrete datatypes (like a concept representing all integers between 4 and 17). In this section, we discuss the consequences of these extensions on the reasoning problems in $\mathcal{SHIQ}$.

Concrete datatypes, as available in DAML+OIL, are a very restricted form of so-called concrete domains [5]. For example, using the concrete domain of all nonnegative integers equipped with the $<$ predicate, a (functional) role age relating (abstract) individuals to their (concrete) age, and a (functional) subrole father of $\mathsf{hasParent}$, the following axiom states that children are younger than their fathers:

$$\mathsf{Animal} \sqsubseteq (\mathsf{age} < \mathsf{father} \circ \mathsf{age}).$$

Extending expressive DLs with concrete domains may easily lead to undecidability [10, 59]. However, DAML+OIL provides only a very limited form of concrete domains. In particular, the concrete domain must not allow for predicates of arity greater than 1 (like $<$ in our example), and the predicate restrictions must not contain role chains (like $\mathsf{father} \circ \mathsf{age}$ in our example). In [67], decidability of $\mathcal{SHIQ}$ extended with a slightly more general type of concrete domains is shown.

Concerning nominals, things become a bit more complicated. Firstly, it can be shown that $\mathcal{SHIQ}$ extended with nominals is a fragment of C2, the two-variable fragment of first order logic with counting quantifiers [39, 65, 77]. Thus, satisfiability and subsumption are decidable in NExpTime. This is optimal since the problem is also NExpTime-hard [77]. Roughly speaking, the combination of GCIs (or transitive roles and role hierarchies), inverse roles, and number restrictions with nominals is responsible for this leap in complexity (from ExpTime for $\mathcal{SHIQ}$ to NExpTime). To the best of our knowledge, no "practicable" decision procedure for $\mathcal{SHIQ}$ with nominals has been described until now. With "practicable" we mean an algorithm that can be implemented with reasonable effort and can be optimized such that it behaves well in practice (which is the case for the algorithm for $\mathcal{SHIQ}$ implemented in FaCT).

7 Conclusion

The emphasis in DL research on a formal, logic-based semantics and a thorough investigation of the basic reasoning problems, together with the availability of highly optimized systems for very expressive DLs, makes this family of knowledge representation formalisms an ideal starting point for defining ontology languages for the Semantic Web. The reasoning services required to support the construction, integration, and evolution of high quality ontologies are provided by state-of-the-art DL systems for very expressive languages.

To be used in practice, these languages will, however, also need DL-based tools that further support knowledge acquisition (i.e., building ontologies), maintenance (i.e., evolution of ontologies), and integration and inter-operation of ontologies. First steps in this direction have already been taken. For example, OilEd [14] is a tool that supports the development of OIL[5] and DAML+OIL ontologies, and ICom is a tool that supports the design and integration of entity-relationship and UML diagrams. On a more fundamental level, so-called non-standard inferences that support building and maintaining knowledge bases (like computing least common subsumers, unification, and matching) are now an important topic of DL research [12, 13, 11, 58]. All these efforts aim at supporting users that are not DL-experts in building and maintaining DL knowledge bases.

References

1. F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 1991.
2. F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.
3. F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.
4. F. Baader, H.-J. Bürkert, B. Hollunder, W. Nutt, and J. H. Siekmann. Concept logics. In John W. Lloyd, editor, *Computational Logics, Symposium Proceedings*, pages 177–201. Springer-Verlag, 1990.
5. F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.
6. F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, volume 567 of *Lecture Notes In Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.
7. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 2001. To appear. An abridged version appeared in *Tableaux 2000*, volume 1847 of LNAI, 2000. Springer-Verlag.

[5] OIL is a fragment of DAML+OIL.

8. F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 1991.
9. F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence Journal*, 88(1–2):195–213, 1996.
10. F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German AI-Conference, GWAI-92*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143, Bonn, Germany, 1992. Springer-Verlag.
11. F. Baader, R. Küsters, A. Borgida, and D. L. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.
12. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 96–101, 1999.
13. F. Baader and P. Narendran. Unification of concepts terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.
14. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the semantic web. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 1–9. CEUR (http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/), 2001.
15. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, 2001.
16. A. Borgida. On the relative expressive power of Description Logics and Predicate Calculus. To appear in *Artificial Intelligence*, 1996.
17. R. J. Brachman. "reducing" CLASSIC to practice: Knowledge representation meets reality. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, pages 247–258. Morgan Kaufmann, Los Altos, 1992.
18. R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI-84)*, pages 34–37, 1984.
19. R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
20. P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: Preliminary report. In *Proc. of the 1995 Description Logic Workshop (DL'95)*, pages 131–139, 1995.
21. M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. Terminological systems revisited: Terminology = schema + views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 199–204, Seattle (USA), 1994.
22. M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. A refined architecture for terminological systems: Terminology = schema + views. *Artificial Intelligence Journal*, 99(2):209–260, 1998.
23. M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
24. D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers (North-Holland), Amsterdam, 1999.

25. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the Seventeenth ACM SIGACT SIGMOD Sym. on Principles of Database Systems (PODS-98)*, pages 149–158, 1998.
26. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 2–13, 1998.
27. DAML language home page (http://www.daml.org/language/).
28. G. De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.
29. G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 205–212. AAAI Press/The MIT Press, 1994.
30. G. De Giacomo and M. Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of the 11th European Conf. on Artificial Intelligence (ECAI-94)*, pages 411–415, 1994.
31. G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.
32. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, Boston, MA, USA, 1991.
33. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 458–463, Sydney, 1991.
34. F. M. Donini, B. Hollunder, M. Lenzerini, A. M. Spaccamela, D. Nardi, and W. Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence Journal*, 2–3:309–327, 1992.
35. J. Doyle and R. S. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence Journal*, 48:261–297, 1991.
36. D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
37. D. Fensel, F. van Harmelen, M. Klein, H. Akkermans, J. Broekstra, C. Fluit, J. van der Meer, H.-P. Schnurr, R. Studer, J. Hughes, U. Krohn, J. Davies, R. Engels, B. Bremdal, F. Ygge, T. Lau, B. Novotny, U. Reimer, and I. Horrocks. On-To-Knowledge: Ontology-based tools for knowledge management. In *Proceedings of the eBusiness and eWork 2000 (eBeW'00) Conference*, 2000.
38. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.
39. E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proc. of the 12th Ann. IEEE Symp. on Logic in Computer Science (LICS-97)*, 1997. Available via http://speedy.informatik.rwth-aachen.de/WWW/papers.html.
40. E. Grädel. Guarded fragments of first-order logic: A perspective for new description logics? In *Proc. of the 1998 Description Logic Workshop (DL'98)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-11/, 1998.
41. E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

42. E. Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.
43. T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.
44. N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *Int. Journal of Human-Computer Studies*, 43(5/6):625–640, 1995.
45. V. Haarslev and R. Möller. RACE system description. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics*, Linköping, Sweden, 1999. CEUR.
46. V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-01)*, volume 2083 of *Lecture Notes In Artificial Intelligence*. Springer-Verlag, 2001.
47. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
48. B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *ECAI-90*, Pitman Publishing, London, 1990.
49. B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 335–346, 1991.
50. I. Horrocks. The FaCT system. In Harrie de Swart, editor, *Proc. of the Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98)*, volume 1397 of *Lecture Notes In Artificial Intelligence*, pages 307–312. Springer-Verlag, 1998.
51. I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, 1998.
52. I. Horrocks and P. Patel-Schneider. The generation of DAML+OIL. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 30–35. CEUR (http://ceur-ws.org/), volume 49, 2001.
53. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes In Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.
54. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic shiq. In D. MacAllester, editor, *Proc. of the 17th Conf. on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes in Computer Science, Germany, 2000. Springer-Verlag.
55. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 636–647, 1998.
56. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
57. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes In Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

58. R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes In Artificial Intelligence*. Springer-Verlag, 2001.
59. C. Lutz. NExpTime-complete description logics with concrete domains. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-01)*, number 2083 in Lecture Notes In Artificial Intelligence, pages 45–60. Springer-Verlag, 2001.
60. R. MacGregor. The evolving technology of classification-based knowledge representation systems. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, Los Altos, 1991.
61. E. Mays, R. Dionne, and R. Weida. K-REP system overview. *SIGART Bulletin*, 2(3), 1991.
62. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes In Artificial Intelligence. Springer-Verlag, 1990.
63. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence Journal*, 43:235–249, 1990.
64. B. Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.
65. L. Pacholski, W. Szwast, and L. Tendera. Complexity of two-variable logic with counting. In *Proc. of the 12th Ann. IEEE Symp. on Logic in Computer Science (LICS-97)*, 1997.
66. L. Pacholski, W. Szwast, and L. Tendera. Complexity of two-variable logic with counting. In *Proc. of the 12th Ann. IEEE Symp. on Logic in Computer Science (LICS-97)*, pages 318–327. IEEE Computer Society Press, 1997.
67. J. Z. Pan. Web ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Workshop on Methods for Modalities 2001 (M4M-2001)*, Amsterdam, 2001. ILLC.
68. P. F. Patel-Schneider. DLP. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 9–13. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-22/, 1999.
69. P. F. Patel-Schneider, D. L. McGuiness, R. J. Brachman, L. A. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3):108–113, 1991.
70. C. Peltason. The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119, 1991.
71. U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, volume 1137 of *Lecture Notes In Artificial Intelligence*. Springer-Verlag, 1996.
72. U. Sattler. Description logics for the representation of aggregated objects. In W.Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press, Amsterdam, 2000.
73. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 466–471, Sydney, 1991.
74. K. Schild. *Querying Knowledge and Data Bases by a Universal Description Logic with Recursion*. PhD thesis, Universität des Saarlandes, Germany, 1995.
75. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence Journal*, 48(1):1–26, 1991.

76. R. Stevens, I. Horrocks, C. Goble, and S. Bechhofer. Building a reason-able bioinformatics ontology using OIL. In *Proceedings of the IJCAI-2001 Workshop on Ontologies and Information Sharing*, pages 81–90, 2001.
77. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001. electronically available at http://www.bth.rwth-aachen.de/ediss/ediss.html.
78. W. van der Hoek and M. De Rijke. Counting objects. *Journal of Logic and Computation*, 5(3):325–345, 1995.

Living Books, Automated Deduction and Other Strange Things*

Peter Baumgartner and Ulrich Furbach

Institut für Informatik
Universität Koblenz-Landau, Koblenz, Germany
{peter,uli}@uni-koblenz.de

Abstract. This work is about a "real-world" application of automated deduction. The application is the management of documents (such as mathematical textbooks) as they occur in a readily available tool. These documents are "living", in the sense, that they can be modified and extended by the reader, who can interact via Living Books with external systems.
A particular task is to assemble a new document from such units in a selective way, based on the user's current interest and knowledge.
It is argued that this task can be naturally expressed through logic, and that automated deduction technology can be exploited for solving it. More precisely, we rely on first-order clausal logic with some default negation principle, and we propose a model computation theorem prover as a suitable deduction mechanism.

1 Introduction

In this paper we describe an approach for the development of personalized interactive electronic publications[1]. Personalized electronic books have been described elsewhere (e.g. in [3]). The term "Living Book" is chosen to express the additional feature of interaction. This is meant not just as a means of combining a text with some additional software-tools in order to enable the user to interactively experiment in the domain of the book. Moreover we understand Living Books as electronic books which can be modified by the user. To this end the user can introduce examples, she can decide which systems, such as theorem provers, are to be applied to the examples. Finally, the result of the interaction is included within the book. Thus each user gets an individual version of her book, which is maintained by the central book-server.

Automated deduction plays a role in two aspect within this approach: an automated reasoning system, based on model generation for first order predicate logic, is used as a reasoning machinery to compute and to compose those parts of

* This work is supported by an EU grant TRIAL-SOLUTION and by a BMBF grant In2Math.

[1] This work is an extended version of [3]; the concept of Living Book, i.e. the interaction with external systems is new in this version.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 249–267, 2005.

the electronic book which meets the query resp. the specification of the user. We discuss the application of a model generation theorem prover, based on hyper tableau, for this task and we finally motivate and discuss the extension of this classical first oder prover by non-monotonic reasoning.

In addition to this meta-level usage, automated deduction is a kind of object, since the book we are developing as a Living Book is on logics and automated deduction. For this, various deduction systems and tools are used as interaction facilities in order to produce an interactive personalized electronic book.

The concept of Living Book is already used in our university education for computer science students. A discussion of some experiences is included below.

2 Living Book

In this section we introduce the concept of Living Book. Since this is based on the concept of Slicing Book, which was developed as a technology for personalized intelligent books, we briefly recall Slicing Book in the following. In the subsequent subsection we discuss the extension towards living books.

2.1 Slicing Book Technology

Slicing Information Technology (SIT) is a tool for the management of personalized docu-ments. With SIT, a document, say, a book on logics for computer scientists, is separated once as a preparatory step into a number of small units, such as definitions, theorems, proofs, and so on. The purpose of the sliced book then is to enable authors, teachers and students to produce personalized teaching or learning materials based on a selective assembly of units. Once a reader is entering the portal of the book in the web, she can login with her account and gets the entry page of the book; this is depicted in Figure 1 for http://www.slicing.de/en/sbooks.php of [10].

The reader can browse through the TOC and specify which parts of the book she wants to read.

Besides of this, she can search the document by taking into account the meta-data which is stored together with the slices of the document. These meta-data contains the type of a slice (e.g. theorem, definition, example and others) or relations which enable the systems to deduce all the material necessary for the understanding of the units presented until now. Such a search is specified in Figure 3.

The reader also can choose to include all material which uses the slices marked until then, e.g. to understand how and where a certain property is used in the rest of the chapter or the entire book. She furthermore can store information, which parts of the book she do not want to get presented anymore, because she knows it very well. All such reasoning and computations are only possible, because besides the LaTeX sources of the units there is also a lot of meta-data stored on the server. This includes besides a glossary and keywords also information about relations of a unit to other units. Hence, we have not only the text of the book, we have an entire knowledge base about the material, which can be used by the reader.

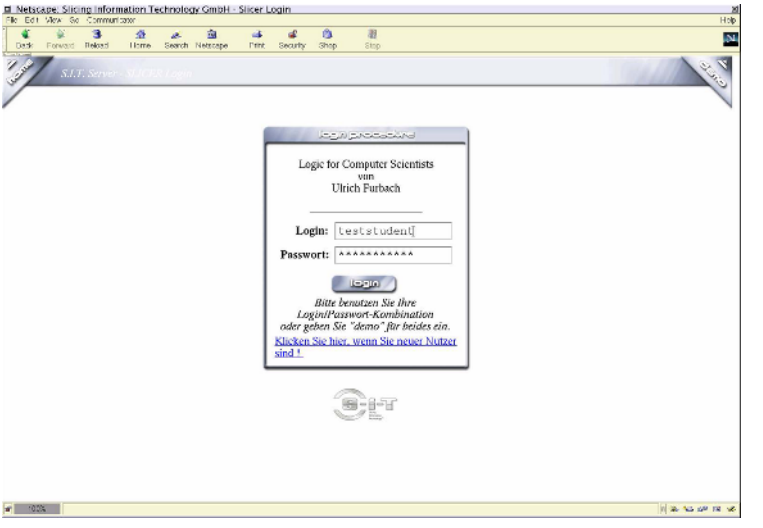


Fig. 1. SIT Reader – Login Page.

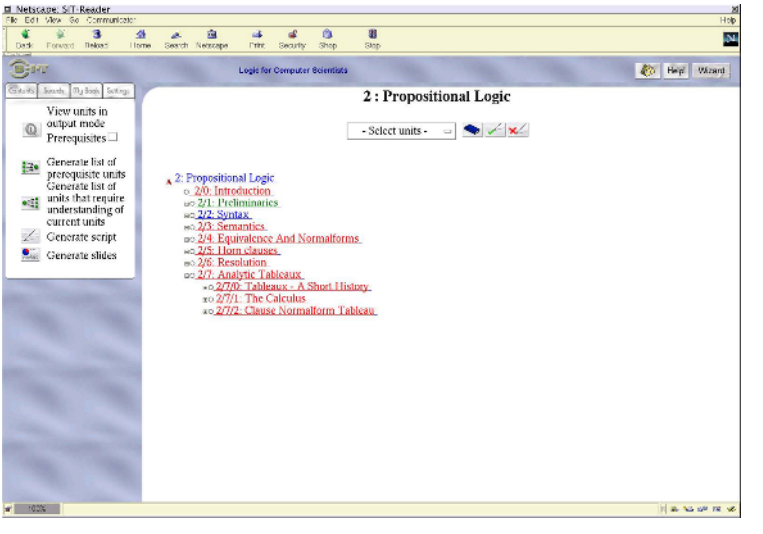


Fig. 2. Browsing the TOC.

Finally the reader can choose to get a print version of all the results of his query. This is produced by the system by putting together all LaTeX parts of the specified slices, processing the entire code and presenting it through a PDF-reader plug-in of the web browser.

Since SIT is applied in the "real world", this knowledge base together with its reasoning mechanism has to be robust, reliable and of course efficient. SIT is applied until know to several mathematics text book, which are explored commercially by the Springer Verlag. Furthermore, SIT is the technical basis of the

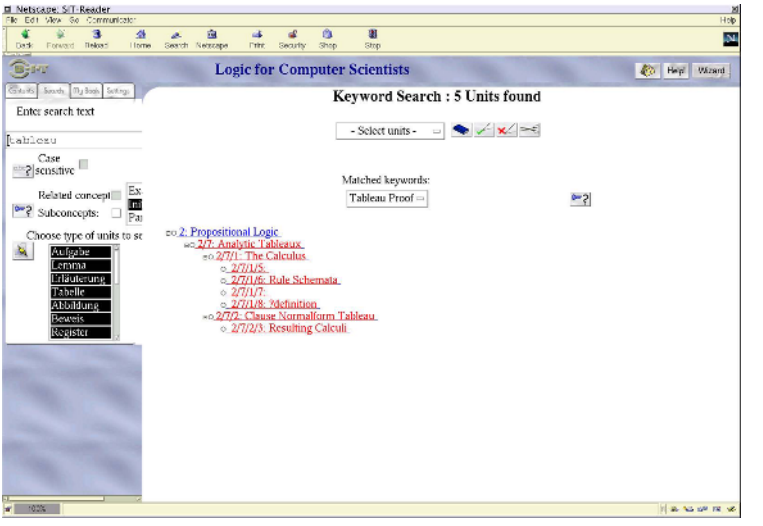


Fig. 3. Searching the document.

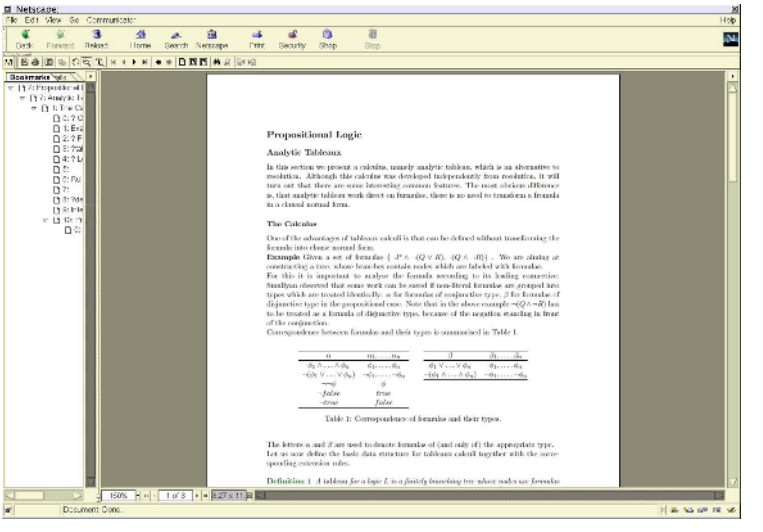


Fig. 4. Displaying the Document.

TRIAL-SOLUTION project. This aims to develop a technology for the generation of personalized teaching materials – notably in the field of mathematics – from existing documents (cf. www.trial-solution.de for more details).

Current work on SIT within the TRIAL-SOLUTION context is concerned with techniques to extend the capabilities by handling knowledge coming from various sources. In our previous example from Figure 1, e.g. it could be the case that the reader decides that she wants to get examples related to the material she

is studying, however from another book, that she knows to be much more of the introductory kind. By this, she could have a much better chance to understand the material. This approach is motivated by the imagination of a reader standing in front of a shelf of books in a library and searching the material she needs for her work: she is using different sources for her search, like books, table of contents, catalogues or book reviews. In our case of SIT, these sources include (i) different sliced books, (ii) a knowledge base of meta data on content (e.g. by keywords), didactic features and interoperability interfacing, (iii) the user profile, including e.g. information about units known to her, and (iv) thesauri that help to categorize and connect knowledge across different books.

All these sources are to be taken into account when generating a personalized document. So far, no language was available to us to formulate in a "nice" way (convenient, friendly to change and adapt to new needs, efficient, ...) the computation of the personalized documents. Our approach based on logic was heavily motivated to come to a solution here.

2.2 Living Book

On the basis of the Slicing Book Technology we are currently developing the concept of Living Books. Living Books are books in Slicing Book Technology which additionally have the property to allow interaction by the user, which may result in a modification of the book. The user can store the interaction together with its results in her version of the personalized book. This is exemplified by a book on logics for computer science([10]), where various deduction systems are usable via interaction.

As described in the previous section the reader is querying a specific part of her book. Figure 5 shows a part on the computation of the resolution closure for propositional clauses. The book is presented in a PDF-format, in so called screen-mode. In this mode the right hand side of each screen gives some navigation and interaction buttons. In this screen-shot we see some clauses which are the result of a previous interaction, where we previously transformed a specific formula into its clause normal form. In this screen the reader can modify this clause set. In our example she decided to add some new clauses in order to turn the depicted current set of clauses into a unsatisfiable one. Figure 6 shows the result of inserting two additional clauses $\neg c$ and $\neg b$ into the input window. Note, that these clauses are now part of the PDF-version of the book. They will remain there even after exiting from the session; if the user is login in again, she can find it again, unless she does not delete or modify them by other interactions.

As a next step the user decides to apply the resolution closure to the current set of clauses as it is depicted in Figure 6. For this she has to click the corresponding button (which is outside the the scope of this screen); this has the effect, that the clause set is transmitted to a resolution system, which is running on the book-server, which computes all the resolvents of this clause set. The result of this computation is than included in the current personalized LaTeX file, which then is process by PDF-Latex with the result depicted in Figure 7.

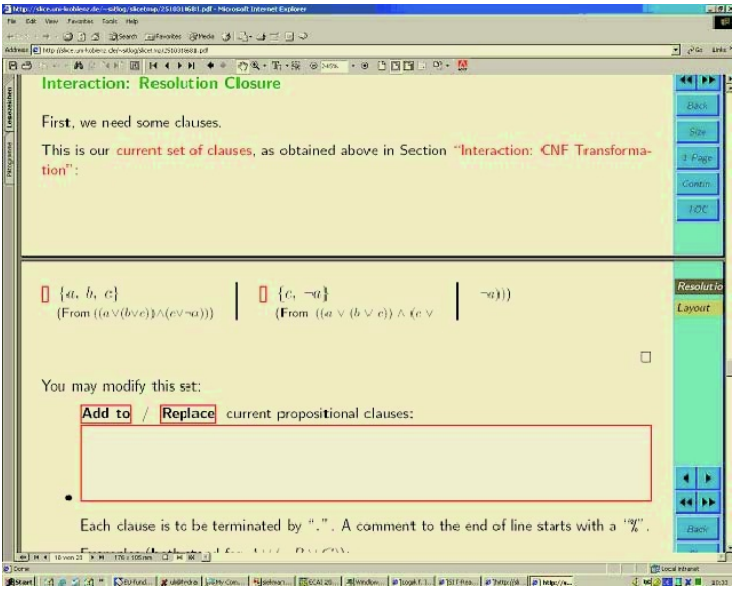


Fig. 5. Interaction in Living Book.

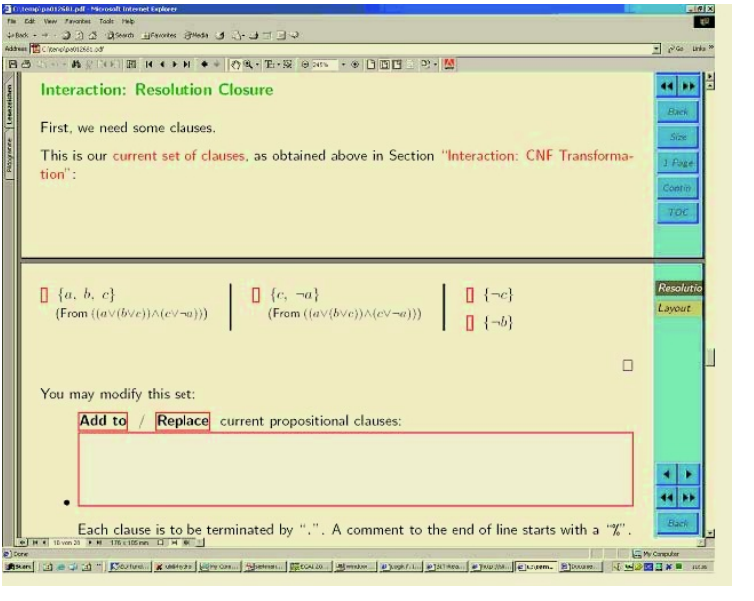


Fig. 6. Adding new clauses to the book.

Until now we included various different systems into Living Book to interact with them. Figure 8 shows the result of a truth-table generation and furthermore, in the navigation part of the frame various systems are listed as buttons for their invocation:

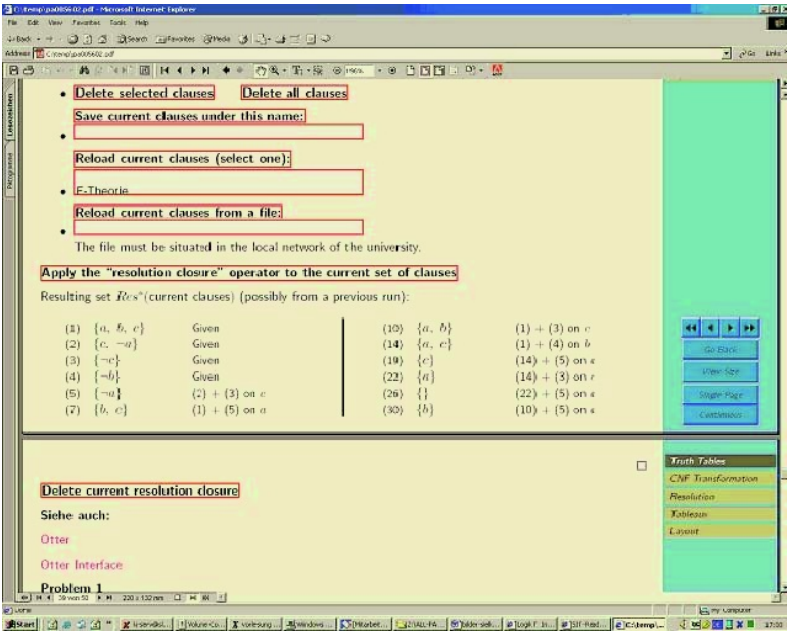


Fig. 7. Adding the result of an external system.

- CNF-Transformation.
- Resolution: a naive resolution prover.
- Tableau: an analytic tableau prover.

Note this is work in progress and other systems will be included. For instance, we will include the well-known Otter resolution prover, so that students can compare the naive resolution algorithm as presented in class with an optimized version.

One point to emphasize is, that we offer a unique input syntax for all systems already included and to be included in the future. So, a user does not have to learn the individual syntax of the systems offered.

3 The Reasoning Mechanism

In our approach for handling the personalization of Living Books, the document to be generated is computed by a model generating theorem prover. In Figure 9 the entire knowledge representation and reasoning task is depicted. On the left hand side we see several books, which can be used as sources for the query given by the user on the right hand side of the picture. Besides the various books, the system takes into account a number of different knowledge sources: keywords, an ontology giving relations between the keywords and terms and dependencies between units with respect of a refers and requires relation. All this

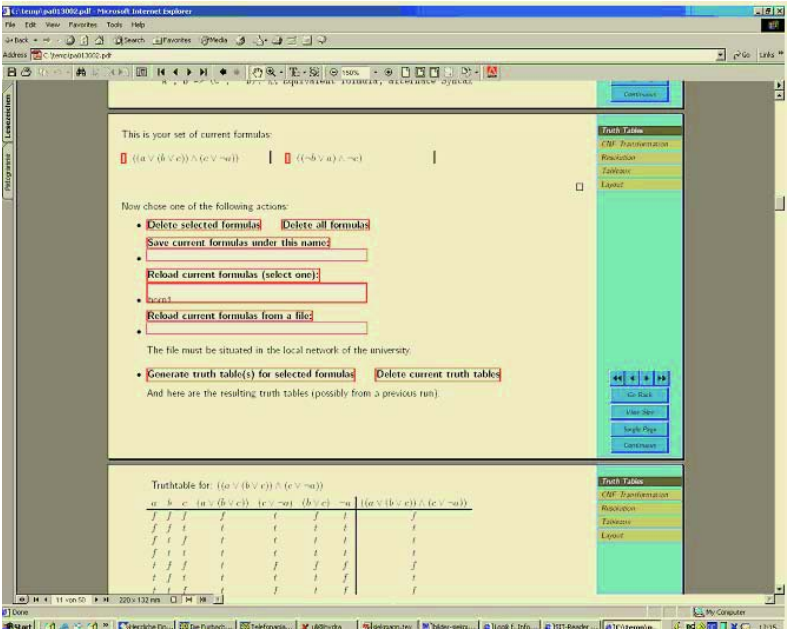


Fig. 8. Truth-table generation.

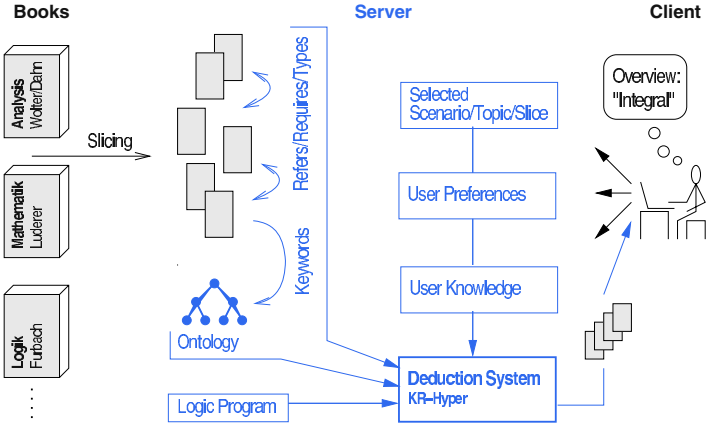


Fig. 9. The reasoning part of SBT.

knowledgeis given as meta-data of the books and can be used by the deduction system together with additional data about the user to answer the query.

The computation is triggered by marking some unit U as a "selected unit". The back-ground theory is essentially a specification of units to be included into the generated document. Such a specification is from now on called a query, and the task to generate the document from the *query* and the selected unit is referred to as *solving the query*.

Here is a sample query:

(i) For each keyword K attached to the selected unit U, include in the generated document some unit D that is categorized as a definition of K; in case there is more than such unit, prefer one from book A to one from book B.
(ii) Include all the units that have at least one keyword in common with the keywords of U and that are of explanatory type (examples, figures, etc).
(iii) Include all the units that are required by U.
(iv) Include U.

In our experiments we use sliced versions of two mathematics text books. Here are two sample units, one from each book:

```
    Ident: 0/1/2/1
     Text: \item Mengentheoretische
           Grundbegriffe ...
     Book: Wolter/Dahn
     Type: Merksatz
   Refers: 0/1/0/2, 0/1/0/3, ...
Requires:
     Keys: set, set intersection,
           set equality, ...
```

```
    Ident: 1/1/1/1/0
     Text: \definition{
           \textbf{Definition}:
           Eine \textbf{Menge} ...
     Book: Gellrich/Gellrich
     Type: Definition
   Refers:
Requires:
     Keys: set
```

The *Ident* field contains a unique name of the unit in the form of a Unix file system sub path, matching the hierarchically organization of the units according to the books sectioning structure. The *Text* field contains the unit's text. The *Book* field contains the name of the book's authors. The Type field denotes the class the unit belongs to. The *Refers* and *Requires* fields contain dependencies from other units. The *Keys* field contains a set of keywords describing the contents of the unit.

Now suppose that the unit with Ident `0/1/2/1` is selected, and that the query from above is to be solved. Some aspects in a logical formalization of this task are straightforward, like the representation of the units and the representation of the selected unit identifier as a fact (as in `selected_unit(0/1/2/1)`). The full formalization of the query proper is too long to be included here[2]. In order to motivate the whole approach, highlighting some parts of it might be helpful, though. Each of the following four parts demonstrates a different aspect of the formalization[3]. In the field of knowledge representation it is common practice to identify a clause with the set of its ground instances. Reasoning mechanisms often suppose that these sets are finite, so that essentially propositional logic results. Such a restriction should not be made in our case. Consider the following clauses:

```
wolter_dahn_unit(0 / _).
gellrich_gellrich_unit(1 / _).
```

```
equal(X, X).
```

[2] It consists of 18 clauses.
[3] Written in a Prolog-style notation.

The first two facts test if a given unit identifier denotes a unit from the respective book. Due to the /-function symbol (and probably others) the Herbrand-Base is infinite. Certainly it is sufficient to take the set of ground instances of these facts up to a certain depth imposed by the books. However, having thus exponentially many facts this option is not really a viable one.

Observe that *range restriction* (cf.[16]) does not apply to facts. The work-around, which is taken by some calculi, to enumerate the Herbrand-Base during proof search does not look too prospective either. In our system full first order predicate logic specification are used.

3.1 Non-classical Negation

Another feature of our reasoning mechanism is that of non-classical negation. Consider the following clauses:

```
computed_unit(UnitId) :-
    candidate_def(UnitId,Key),
    not multiple_def(Key).

multiple_def(Key) :-
    candidate_def(UnitId1,Key),
    candidate_def(UnitId2,Key),
    not equal(UnitId1,UnitId2).
```

The `computed_unit` relation shall contain those units (named by unit identifiers `UnitId`) that go into the generated document. According to the left clause, this applies to any candidate definition unit for some keyword Key (derived by some other other clauses not described here), provided there is not more than one such candidate definition of `Key`. The second clause states the definition of the `multiple_def`-relation.

What is the intended semantics of `not`? *Classical* semantics is not appropriate, as it allows, for instance, arbitrary different terms to hold in the equal-relation. The correct intention, however, of `equal` is to mean syntactical equality. Likewise, classical semantics allows for counter-intuitive interpretations of `multiple_def`.

We define a special case of the *supported model* semantics[4] as the intended meaning of our programs, and solving the query means to compute such a model for the given program. This point is worth emphasizing: we are thus *not* working in a classical theorem-proving (i.e. refutational) setting; solving the query is, to our intuition, more naturally expressed as a model-generation task.

Fortunately, model computation for stratified programs is much easier than for non-stratified programs, both conceptually and in a complexity-theoretic sense. There is little dispute about the intended meaning of stratified programs, at least for *normal* programs (i.e. programs without disjunctions in the head), and the two major semantics coincide, which are the stable model semantics [11] and the well-founded model semantics [24]. For propositional stratified normal programs, a polynomial time decision procedure for the model existence problem exists, which does not exist for the stable model semantics for non-stratified

[4] A model I of a (stratified and ground) clause set M is a *supported model* of M iff for every $A \in i$ there is a clause $A \vee A_1 \vee \ldots A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge \not B_{m+1} \wedge \ldots \wedge B_n$ in M such that $M \models B_1 \wedge \ldots \wedge B_m \wedge \not B_{m+1} \wedge \ldots \wedge B_n$

normal programs. Being confronted with large sets of data (stemming from tens of thousands of units) was the main motivation to strive for a tractable semantics.

As mentioned above, we do not restrict ourselves to normal programs. Instead, we found it convenient to allow disjunctions in the head of clauses in order to express degrees of freedom for the assembly of the final documents. Also for the disjunctive case (i.e. non-normal programs), stable model semantics and well-founded model semantics have been defined (see e.g. [6]). Both semantics agree to assign a minimal model semantics to disjunction. For instance, the program

$$A \vee B \leftarrow$$

admits two minimal models, which are $\{A\}$ and $\{B\}$. An equivalent characterization of minimal models is to insist that for each atom true in the intended model, there is a head of a true clause where only this atom is true. For our task, however, this preferred exclusive reading of disjunctions seems not necessarily appropriate. When assembling documents, redundancies (i.e. non-minimal models) should not be prohibited unless explicitly stated. In order not having to restrict to the minimal model semantics, we find the possible model semantics to be appropriate [20]. With it, the above program admits all the obvious models $\{A\}$, $\{B\}$ and $\{A, B\}$. If the inclusive reading is to be avoided, one would have to add the integrity constraint

$$\leftarrow A \wedge B\,.$$

Unfortunately, the possible model semantics is costly to implement. At the current state of our developments, our calculus is *sound* wrt. the possible model semantics (i.e. any computed model is a possible model) but *complete* only in a weak sense (if a possible model exists at all, some possible model will be computed). In our current setting, solving the query means to compute *any* model of the program, so the lack of completeness is not really harmful. However, future experiments will have to show if this approach is really feasible.

3.2 The Calculus

One of the big challenges in both classical logic and nonmonotonic logics is to design calculi and efficient procedures to compute models for *first-order* specifications. Some attempts have been made for classical first-order logic, thereby specializing on decidable cases of first-order logic and/or clever attempts to discover loops in derivations of standard calculi (see e.g. [9, 18, 2, 23]).

In the field of logic programming, a common viewpoint is to identify a program with the set of all its ground instances and to apply propositional methods then. Notable exceptions are described [5, 8, 12, 7]. Of course, the "grounding" approach is feasible only in restricted cases, when reasoning can be guaranteedly restricted to a finite subset of the possibly infinite set of ground instances. Even the best systems following this approach, like the S-models system [17], quite often arrive at their limits when confronted with real data.

In our application we are confronted with data sets coming from tens of thousands of units. Due to this mass, grounding of the programs before the computation starts seems not to be a viable option. Therefore, our calculus directly computes models, starting from the given program, and without grounding it beforehand. In order to make this work for the case of programs with default negation, a novel technique for the representation of and reasoning with non-ground representations of interpretations is developed.

The calculus developed here is obtained by combining features of two calculi readily developed – *hyper tableaux* [4] and FDPLL [2] – and some further adaptations for default negation reasoning. These two calculi were developed for *classical* first-order reasoning, and the new calculus can be seen to bring in "little monotonicity" to hyper tableaux.

The calculus is called *hyper tableau* because it combines two characteristics: the idea of clustering certain basic inference rules into a single one, as it is used in hyper-resolution [19], and the overall calculus as it was developed for clause normal form tableau (see [15]). Instead of defining the hyper tableau calculus formally we will illustrate it with the following example.

Consider the following set of clauses, where clauses are given in implication form, such that $B_1 \vee \ldots \vee B_m \leftarrow A_1 \wedge \ldots \wedge A_n$ stands for the clause $B_1 \vee \ldots \vee B_m \vee \neg A_1 \vee \ldots \vee \neg A_n$.

$$A \leftarrow A \tag{1}$$

$$B \vee C \leftarrow A \tag{2}$$

$$A \vee C \leftarrow C \tag{3}$$

$$\leftarrow A \wedge B \tag{4}$$

In order to construct a hyper tableau for this clause set, we start with the empty tableau ε, which is given in the left part of Figure 10. We will discuss this derivation from left to right: If we consider clause (1), which we can understand as "in any model A has to hold", hence we extend our single (empty) branch of the tableau ε by the new leaf A. We arrive at a tableau with a branch which contains the (possibly) partial model A. Obviously clause (2) does not hold in this model, because (2) is stating "if in a model A holds, than B or C has to hold as well". Let's repair this, by extending our tableau by these two possibilities; we extend it by the disjunction $B \vee C$, which is expressed in the tableau by a new branching. The left branch $\{A, B\}$ of this new tableau, again is a (possibly) partial model, but now we observe that there is a contradiction to clause (4), which is stating that A and B cannot be true together in any model; hence we know that this branch does not correspond to a partial model – we mark it as closed with an asterisk. The right branch of the tableau, however, although it could be further extended, is a model of the entire clause set (1)–(4).

We demonstrated the calculus only in the propositional case, but it can be extended to a complete and correct calculus for full first order clausal logic, and there are various improvements of its basic variant as introduced in [4].

Hyper tableau calculi are tableau calculi in the tradition of SATCHMO [16]. In essence, interpretations as candidates for models of the given clause set are

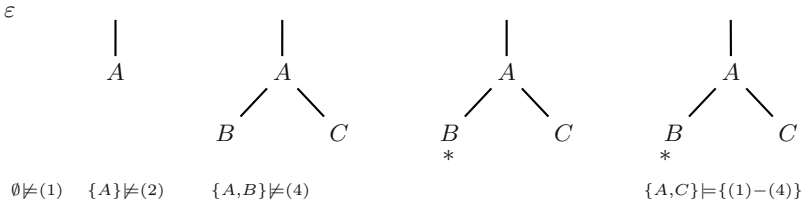


Fig. 10. A sample hyper tableau derivation.

generated one after another, and the search stops as soon as a model is found, or each candidate is provably not a model (refutation). A distinguishing feature of the hyper tableau calculi [4, 1] to SATCHMO and related procedures is the representation of interpretations at the first-order level. For instance, given the very simple clause set consisting of the single clause

$$\text{equal}(X, X) \leftarrow$$

the calculus stops after one step with the model described by the set {equal (X, X)}, which stands for the model that assigns true to each ground instance of equal(X, X).

The hyper tableau calculi developed so far do not allow for default negation. In the present work we therefore extend the calculus correspondingly. At the heart is a modified representation of interpretations. The central idea is to replace atoms – which stand for the set of all their ground instances – by pairs $A - \{E_1, \ldots, E_n\}$, where A is an atom as before, and E is a set of atoms ("Exceptions") that describes which ground instances of A are *excluded* from being true by virtue of A. For instance, if the clause

$$\text{different}(X, Y) \leftarrow \text{notequal}(X, Y)$$

is added, then the search stops with the set

$$\{\text{equal}(X, X) - \{\}, \text{different}(X, Y) - \{\text{different}(X, X)\}\} .$$

It represents the model where all instances of equal(X, X) are true, and all instances of different(X, Y) are true, except the reflexive ones. There are several non-monotonic variants (e.g. for minimal model reasoning or for abduction) and for modal logics.

To see how we use the calculus in the context of SBT, we take an excerpt from the knowledge base, and use it to demonstrate the working of the hyper tableau calculus modified for nonmonotonic reasoning. Figure 11 depicts this excerpt; it is from the "user model", and its purpose is to find out by means of the `known_unit_inferred` relation whether the user knows some unit in question. The clauses in the knowledge base in Figure 11 are stratified, i.e. they are organized in a hierarchical way. On the base layer, e.g., clause (2) is expressing the fact that the current user does not know the unit `1/2/1` from the `analysis`-book. Clause (1)

is a good example demonstrating the representational advantage of our system over other systems that require grounding of the clauses before computation (this is discussed in Section 3.2): in clause (1), `_ALL_` is a universally quantified variable, i.e., intentionally, all subunits of `analysis/1/2` are declared to be known. It is the purpose of clause (6) to resolve the apparent inconsistency behind the just given explanation of clauses (1) and (2) (cf. below).

Clauses (4) and (5) are expressing knowledge about the meta data relations "known unit" and "unknown unit". Clause (6) says that Book/Unit is contained in the `knows_unit_inf`-relation (the relation we are interested in), if it is "known" (by means of the `known_unit(Book/Unit)` declaration) *and* this circumstance is not overridden by a an explicit "unknown"-declaration of the same unit (by means of `not unknown_unit(Book/Unit)`}). By the use of the default negation technique we realize that "unknown"-declarations should be stronger than "known"-declarations. We think that this is appropriate modeling, as "unknown" units are never withhold from the user.

Now, the calculus derives from the clauses in Figure 11 in three steps the hyper tableau in Figure 12.

The topmost three lines stem from the clauses (1), (3) and (2), respectively. By combining clauses (1), (3) and (4) we conclude that the unit `analysis/1/0/4` should be "known". This is realized by the hyper tableau derivation in the fourth line. The concluding line in the derivation is obtained by clauses (1), (2) and (6). It says that in the `known_unit_inf`-relation are all subunits of `analysis/1/2` – more technically: all ground instances of `analysis/1/2/_ALL_` – except for the unit `analysis/1/2/1` (the technique of representing models this

```
%% User knowledge:
known unit(analysis/1/2/ ALL ).                                    (1)
unknown unit(analysis/1/2/1).                                      (2)

%% Book metadata:
refers(analysis/1/2/3, analysis/1/0/4).                            (3)

%% ``known unit'' transitive closure:
known unit(Book B/Unit B) :-                                       (4)
    known unit(Book A/Unit A),
    refers(Book A/Unit A, Book B/Unit B).

%% ``unknown unit'' transitive closure
unknown unit(Book B/Unit B) :-                                     (5)
   unknown unit(Book A/Unit A),
   refers(Book A/Unit A, Book B/Unit B).

%% Derived:
known unit inf(Book/Unit) :-                                       (6)
   known unit(Book/Unit),
   not unknown unit(Book/Unit).
```

Fig. 11. Excerpt from the knowledge base.

known_unit(analysis/1/2/_ALL_)
refers(analysis/1/2/3, analysis/1/0/4)
unknown_unit(analysis/1/2/1)

known_unit(analysis/1/0/4)

known_unit_inf(analysis/1/2/_ALL_)
- { known_unit_inf(analysis/1/2/1) }

Fig. 12. Hyper tableau derivation from the clauses in Figure 11.

way is described above). Observe that the mentioned, apparent contradiction between what clauses (1) and (2) say is eliminated as explained.

3.3 Other Approaches

In the previous sections a non-monotonic extension of first-order theorem proving was advocated as an appropriate formalism to model the task at hand. Undoubtedly, there are other candidate formalisms that seem well-suited, too. In the following we comment on these.

Prolog. Certainly, one could write a Prolog program to solve a query. When doing so, it seems natural to rely on the findall built-in to compute the extension of the `computed_unit` predicate, i.e. the solution to a query. Essentially, this means to enumerate and collect all solutions of the goal `computed_unit(U)` by means of the Prolog built-in backtracking mechanism. In order to make this work, some precautions have to be taken. In particular explicit loop checks would have to be programmed in order to let `findall` terminate. Because otherwise, for instance, alone the presence of a transitivity clause causes `findall` not to terminate.

XSB-Prolog. One of the few programming languages that works top-down (as Prolog) and that has built-in loop checking capabilities (as bottom-up model generation procedures) is XSB-Prolog [22]. XSB-Prolog supports query answering wrt. the well-founded semantics for normal logic programs [24]. At the heart of XSB-Prolog is the so-called `tabling` device that stores solutions (instantiations) of goals as soon as computed. Based on tabling, it is even possible to compute extensions of predicates (such as `computed_unit`) and return them to the user.

The only problem with XSB-Prolog for our application is the restriction to `normal` programs, i.e. disjunctions in the head of program clauses are not allowed. Certainly, this problem could be circumvented by explicitly coding disjunctions in the program, but possibly at the cost of far less intuitive solution. Description Logics. Description logics (DL) are a formalism for the representation of hierarchically structured knowledge about individuals and classes of individuals. Nowadays, numerous descendants of the original $\mathcal{ALC}$ formalism and calculus

[21], e.g. with greatly enhanced expressive power exist, and efficient respective systems to reason about DL specifications have been developed [14].

We can see that a good deal of the information represented by our first-order specifications, could be accessible to a DL formalization. The concrete units would form the so-called assertional part (A-Box), and general "is-a" or "has-a" knowledge would form the terminological part (T-Box). The T-Box would contain, for instance, the knowledge that a unit with type "example" *is-a* "explanatory unit", and also that a unit with type "figure" *is-a* "explanatory unit". Also, transitive relations like "requires" should be accessible to DL formalisms containing transitive roles.

At the current state of our work, however, it is not yet clear to us if and how the nonmonotonic features that we found useful would map to a DL formalism. Certainly, much more work has to be spent here. Presumably, one would arrive at a combined DL and logic programming approach. This is left here as future work.

4 Application of Living Book

The concept of Living Books is developed in a project, called *In2Math*, which is carried out by a consortium consisting of several German universities, a publisher and several companies (http://www.uni-koblenz.de/ag-ki/PROJECTS/in2math/). The goal is to provide material as it is described in the previous sections, to be used in different scenarios of University teaching and learning.

There are several different books under development: three books in the area of mathematics, where computer algebra systems are used for interaction and a course on logics for computer science, where various theorem provers and formulae manipulation systems are accessible from the system.

In the rest of this section we will shortly discuss, how Living Books can be used for teaching. Until know, we identified three different kinds of applications:

Class Room Teaching: For the course "Logics for Computer Scientists" we currently have 80 students, which are attending the lectures. For this the second author uses wireless LAN access to the book server and by overhead projection he is presenting the material on the screen. For this the server can provide a slides-version of the book, where larger fonts and a partition of the material into slides is done automatically. The advantage of this setting is, that during lecturing, the interactive systems can be used exactly at those points, where it was considered meaningful by the author of the book. There is no need for the lecturer to switch applications on the screen – he just interacts with the Living Book. The students, while listening to the lecture, have access to exactly the same material, by using their laptops and Internet access via wireless LAN. They can take notes directly by inserting their remarks and extension into the book, which of course, are stored in the personalized way on the server. The student can work on these notes again, if she is accessing the book later on from elsewhere.

It should be mentioned, that besides these services provided by the technology, there is a big challenge for the lecturer: different phases of a lecture have

to be mixed, such that it results to a meaningful and efficient teaching methods. Our experience is, that there should be intervals during the lecture, where spontaneously the blackboard is used for discussing questions or examples. Such discussions should be alternated with interactions via the Living Book.

Group Work: Besides the class room teaching we organized small groups of up to dozen students to work together on exercises with the help of a tutor. These groups are using laptops and the Living Book concept to work on a set of given exercises. For this, as well as for the overall organization of the courses we us an e-learning system, by which the students can access the exercises and the living book. In order to provide a laptop for each student, we have laptop-pools, which can be booked for those class room exercises by the tutor. A feature which is missing in our concept until now is that of collaborative learning. It would be very appropriate, if a group of students could work together on the solution of a single exercise by decomposing the task and by solving the various parts independently on their systems and then combining things together via the web.

Individual Learning: Of course, this is the classical application of e-learning concepts. The student has access to the material from everywhere and can work with living books individually from her home. This can be done online via the server, or by using a print version of the material, which is providing by the server in a pdf-format of the specified parts of the material.

5 Conclusion

This work is not yet finished. Open ends concern all parts described previously.

For the reasoning part, we have to concider some design decisions and practical experience with real data on a large scale. Concerning design decisions, for instance, it is not quite clear what semantics suits our application best. It is clear that a supportedness principle is needed, but there is some room left for further strengthenings. Further experiments (i.e. the formulation of queries) will guide us here.

The calculus and its implementation are developed far enough, so that meaningful experiments are possible. The implementation is carried out in Eclipse Prolog. For faster access to base relations, i.e. the currently computed model candidate, the discrimination tree indexing package from the ACID term indexing library [13] is coupled. Without indexing, even our moderately sized experiments seem not to be doable. With indexing, the response time for a typical query applied to a full book with about 4000 units takes less than five seconds, which seems almost acceptable.

Another serious topic is how to design the interface between the reasoning system and the user, i.e. the reader of the document. The user cannot be expected to be acquainted with logic programming or any other formal language (typically, this is what students should learn by reading the sliced books). Our system therefore offers a predefined set of queries for some typical learning scenarios, like “I want material to prepare for a written exam”, which are parametrized

by a topic of interest and a user-model (our user model consists of declarations what units are known/unknown to the student).

For Living Books, we have to extend the use of interactions; we have to add additional interactions and we have to use additional external systems to be combined with our books. For this we are experimenting with web interfaces to combine remote systems which are developed and maintained elsewhere. The teaching methods for applications of Living Books in university educations are only in an experimental stage until now; we need much more experiments and formative evaluations.

References

1. Peter Baumgartner. Hyper Tableaux – The Next Generation. In Harry de Swaart, editor, Automated Reasoning with Analytic Tableaux and Related Methods, volume 1397 of Lecture Notes in Artificial Intelligence, pages 60–76. Springer, 1998.
2. Peter Baumgartner. FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In David McAllester, editor, CADE-17 – The 17th International Conference on Auto-mated Deduction, volume 1831 of Lecture Notes in Artificial Intelligence, pages 200–219. Springer, 2000.
3. Peter Baumgartner and Ulrich Furbach. Automated Deduction Techniques for the Management of Personalized Documents. Annals of Mathematics and Artificial Intelligence – Special Issue on Mathematical Knowledge Management, Kluwer Academic Publishers, 2002. Accepted for Publication.
4. Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In Proc. JELIA 96, number 1126 in Lecture Notes in Artificial Intelligence. European Workshop on Logic in AI, Springer, 1996.
5. Sven-Erik Bornscheuer. Rational models of normal logic programs. In Steffen Hölldobler Günther Görz, editor, KI-96: Advances in Artificial Intelligence, volume 1137 of Lecture Notes in Artificial Intelligence, pages 1–4. Springer Verlag, Berlin, Heidelberg, New-York, 1996.
6. Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Andrei Voronkov and Alan Robinson, editors, Handbook of Automated Reasoning, pages 1121–1234. Elsevier-Science-Press, 2001.
7. Jürgen Dix and Frieder Stolzenburg. A framework to incorporate non-monotonic reasoning into constraint logic programming. Journal of Logic Programming, 37(1-3):47–76, 1998. Special Issue on Constraint Logic Programming. Guest editors: Kim Marriott and Peter J. Stuckey.
8. Thomas Eiter, James Lu, and V. S. Subrahmanian. Computing Non-Ground Representations of Stable Models. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97), number 1265 in Lecture Notes in Computer Science, pages 198–217. Springer-Verlag, 1997.
9. Christian Fermüller and Alexander Leitsch. Hyperresolution and automated model building. Journal of Logic and Computation, 6(2):173–230, 1996.
10. Ulrich Furbach. Logic for computer scientists. To appear as Living Book.

11. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, Proceedings of the 5th International Conference on Logic Programming, Seattle, pages 1070–1080, 1988.
12. Georg Gottlob, Sherry Marcus, Anil Nerode, Gernot Salzer, and V. S. Subrahmanian. A non-ground realization of the stable and well-founded semantics. Theoretical Computer Science, 166(1-2):221–262, 1996.
13. P. Graf. ACID User Manual – version 1.0. Technical Report MPI-I-94-DRAFT, Max-Planck- Institut, Saarbrücken, Germany, June 1994.
14. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. Logic Journal of the IGPL, 8(3):239–263, 2000.
15. Reinhold Letz. Clausal Tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, Automated Deduction. A Basis for Applications. Kluwer Academic Publishers, 1998.
16. Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, Proceedings of the 9 th Conference on Automated Deduction, Argonne, Illinois, May 1988, volume 310 of Lecture Notes in Computer Science, pages 415–434. Springer, 1988.
17. Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In Proceedings of the Joint International Conference and Symposium on Logic Programming, Bonn, Germany, 1996. The MITPress.
18. N. Peltier. Pruning the search space and extracting more models in tableaux. Logic Journal of the IGPL, 7(2):217–251, 1999.
19. J. A. Robinson. Automated deduction with hyperresolution. Internat. J. Comput. Math., 1:227–234, 1965.
20. C. Sakama. Possible Model Semantics for Disjunctive Databases. In W. Kim, J.-M. Nicholas, and S. Nishio, editors, Proceedings First International Conference on Deductive and Object-Oriented Databases (DOOD-89), pages 337–351. Elsevier Science Publishers B.V. (North–Holland) Amsterdam, 1990.
21. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1–26, 1991.
22. K. Sagonas, T. Swift, and D. S. Warren. An abstract machine for computing the well-founded semantics. Journal of Logic Programming, 2000. To Appear.
23. Frieder Stolzenburg. Loop-detection in hyper-tableaux by powerful model generation. Journal of Universal Computer Science, 5(3):135–155, 1999. Special Issue on Integration of Deduction Systems. Guest editors: Reiner Hähnle, Wolfram Menzel, Peter H. Schmitt and Wolfgang Reif. Springer, Berlin, Heidelberg, New York.
24. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. Journal of the ACM, 38:620–650, 1991.

An Essay on Sabotage and Obstruction

Johan van Benthem

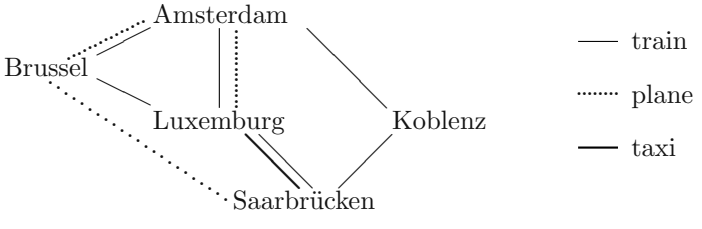
ILLC Amsterdam & CSLI Stanford

To Jörg Siekmann on the occasion of his 60th Birthday

Abstract. This is a light 'divertissement' about the problems of modern transportation, and the beckoning pleasures of meeting with Jörg.

Getting Nowhere

Traveling between Saarbrücken and Amsterdam is easy, as Jörg well knows. A convenient network of connections exists between these two capitals of logic and computation. In the phantasy world of this essay, the network is as pictured below. But what if the transportation system breaks down, and a malevolent demon starts canceling connections, anywhere in the network? Of course, a veteran AI researcher adapts, and changes to the next optimal plan. But what if, at every stage of his trip, the demon first takes out one connection? From Saarbrücken to Amsterdam, Jörg still has a winning strategy. The Demon's opening move may block Brussel or Koblenz, but then Jörg goes to Luxemburg in the first round, and gets to Amsterdam in the next. The Demon may also cut a connection between Amsterdam and some city in the middle – but Jörg can then go to at least one place offering him still two intact roads. My own Dutch situation is less rosy, however. This time, Demon has the winning strategy. It starts by cutting a link between Saarbrücken and Luxemburg. If I now go to any city in the middle, Demon always has time in the next rounds to cut my beckoning link to Saarbrücken. Oh, that fair city on the Saar, and yet so inaccessible!

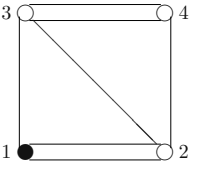


This story may sound like a bad fairy tale – but users of the Dutch Railway System NS will recognize the prevalent situation since one year. Connections disappear in malevolent patterns, and what used to be simple travel has become a game of high complexity. This essay reflects the thoughts of a traveling logician stuck at strange stations...

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 268–276, 2005.

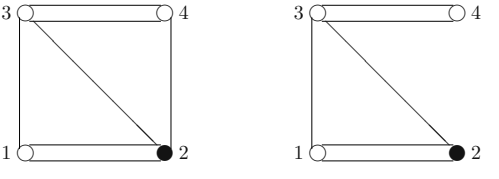
From Algorithms to Obstruction Games

The preceding example is the well-known task of *Graph Reachability* 'sabotaged'. More generally, any algorithmic task over graphs can be turned into a two-player game, with one player 'Runner' trying to do the original job, and another player 'Blocker' taking out edges at each stage. Different schedulings and winning conditions are possible. For instance, consider a game which sabotages *Traveling Salesman*. An undirected graph is given, and Runner must complete a circuit. This time, Blocker lets him start in each round, and then takes out a link. Players must move as long as they can: the game stops the first time a player cannot move. Runner wins if the end situation contains a completed circuit; otherwise, Blocker wins. Which player has a winning strategy in the following game, which starts with Runner at the black dot?



It may take a moment: but then you will see that Blocker has the winning strategy, first cutting one upper link. (Complex games of this sort are a nice pastime for winter evenings.) But, why this implicit trust that one of the two players must have a winning strategy – i.e., that the game is *determined*? Recall *Zermelo's Theorem* from the dawn of game theory. It says that each finite two-player zero-sum game is determined. Of course, the great German set theorist was thinking about Chess when he proved his result – as the Reichsbahn was still running according to schedule in his days – but the mathematics applies equally well to our modern transport plight. The reason is that the sabotage games still satisfy Zermelo's three conditions.

One can see this as a *game transformation*. The original algorithmic task is a graph game with just one player, where that player must perform a sequence of moves creating a path with some desired property ('ending in a specified location', 'forming a circuit', etc.). Now we get a new game, whose local states are subtrees of the original game tree, with a current position for Runner indicated, whose moves are of two kinds. Runner can follow an edge from his current position in the given graph, but Blocker can choose a new graph missing one edge. For instance, the Traveling Salesman game has the given diagram as an initial node. We display one opening move for Runner, followed by one for Blocker:



This blow up of the original search space to a game tree involves an exponential factor – though Blocker's moves also simplify the game as the graph gets simpler.

Complexity Bounds and Model Checking

Intuitively, solving my travel problems, if possible, against sabotage seems more complex than the old task itself. Graph Reachability is in **P**, Traveling Salesman is **NP**-complete. We will not determine the exact complexity of their sabotaged versions, but some quick general observations can be made. First, note that both problems amount to *model-checking* with *first-order* formulas. Given any finite graph **G**, they state the existence of a sequence of points satisfying some first-order definable condition, i.e., both check an *existential* formula of the form:

$$\exists x_1 \ldots \exists x_k \ \alpha(x_1, \ldots, x_k) \qquad \text{with } \alpha \text{ quantifier-free}$$

where k is of the order of the size of **G**. More conveniently, think of the graph as a domain of edges, and let the existential quantifiers run over these. What is the effect of Blocker's activities? At each stage, one edge is removed arbitrarily. The result of this progressive impoverishment is that Runner has a winning strategy if and only if the following first-order formula over edges is true in the graph **G**:

$$\exists x_1 \forall y_1 \exists x_2 \neq y_1 \ \forall y_2 \exists x_3 \neq y_1, y_2 \quad \ldots \quad \alpha'(x_1, \ldots, x_k)$$

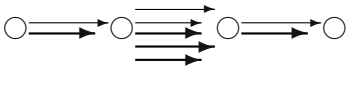
where α' is the old condition suitably adapted in terms of endpoints of edges. The length of this second formula is still linear in the size of the graph, but we have quantifier interchanges now. Thus, an upper bound for the complexity is that of uniform model checking of first-order formulas over finite models, which takes polynomial space in the size of the model plus that of the formula.

Fact. Solving a sabotage game takes at most **PSPACE** in the graph size.

Probably, one cannot do better in general. Here is one lower bound. Consider the **PSPACE**-complete problem of *Quantified Boolean Formulas*. This is akin to a sabotaged Reachability task. Consider this special case:

$$\forall p_1 \exists p_2 \forall p_3 \ \ \alpha(p_1, p_2, p_3) \qquad \text{with } \alpha \text{ some propositional formula}$$

Now look at a graph with 3 nodes, and edges distributed as follows:



thick edges : label *true*
thin edges : label *false*

Here is a sabotage game over this graph. Blocker starts each round by taking away an edge, after which Runner chooses an edge to cross. The game ends when Runner can no longer move – and such a situation is counted as a win for Blocker `iff` it is an end point and the formula $\alpha(p_1, p_2, p_3)$ evaluated according to the labels *true* or *false* of the successive edges chosen by Runner is false. Now, let us analyze the strategic situation in this game. Blocker must let Runner move

three times: otherwise he loses. But by taking away links, he can force the truth value at the first step, and then at the third – or perhaps just at the third. For Runner to succesfully counter all possible actions by Blocker requires the truth of the following formulas:

$$\begin{array}{ll} \forall p_1 \exists p_2 \forall p_3 & \alpha(p_1, p_2, p_3) \\ \forall p_1 \forall p_3 \exists p_2 & \alpha(p_1, p_2, p_3) \\ \forall p_3 \exists p_1 \exists p_2 & \alpha(p_1, p_2, p_3) \\ \exists p_1 \exists p_2 \forall p_3 & \alpha(p_1, p_2, p_3) \\ \exists p_1 \exists p_2 \exists p_3 & \alpha(p_1, p_2, p_3) \end{array}$$

But the second to fifth formulas are implied by the first. Thus, Runner has a winning strategy in this game iff the first quantified Boolean formula is true. I suspect that the general situation with k propositional variables yields to a similar construction, which suggests that

Sabotaged graph tasks can be **PSPACE**-hard games.

Admittedly, the preceding example is contrived – and its trick does not work for the above sabotaged Reachability or Traveling Salesman. On the other hand, John Bell pointed at the related (though more involved) game 'Roadblock' in David Harel's beautiful book *Algorithmics*, which takes even essentially **EXPTIME**. The general complexity behaviour of sabotage still eludes me.

Modal Logics over Changing Models

Instead of solving our complexity problem, we will look at it from another angle. Solving, say, a standard reachability problem amounts to evaluating a modal formula with a number of disjunctions of the form $\Diamond \ldots \Diamond\ p$, where p holds in the goal states. Uniform model checking for modal formulas on a finite model is known to be in **P**-time – measured in the size of the model and the length of the formula. But the above sabotaged versions involve *changing the model* as we proceed. Here is a way of viewing this, thinking of models where arrows are treated as objects. Introduce a cross-model modality referring to submodels from which objects have been removed:

$\mathbf{M}, s \models \Diamond\!\!\!\!-\phi$ `iff` there is a world $w \neq s$ with $\mathbf{M} - \{w\}, s \models \phi$

Now the language has both an 'internal modality' $\Diamond$ and an 'external modality' $\Diamond\!\!\!\!-$, which can be combined. E.g., the fact that universal modal formulas are preserved under submodels shows in valid principles like $\Box \mathrm{p} \rightarrow -\!\!\!\!\Box\Box \mathrm{p}$. Actually, despite the modal notation, this language lacks some typical modal features.

For instance,

the formula $\Diamond\!\!\!\!-\Box \bot$ is not invariant for bisimulations.

This formula holds in an irreflexive 2-cycle, but it fails in the bisimilar model consisting of a single reflexive point. Nevertheless, the formulas of this language are still translateable into an obvious first-order language, using the same trick as above. E.g., $\Diamond\!\!\!\!-\Box \bot$ says that $\exists y \neq x\ \neg \exists z \neq y : Rxz$.

The blow-up in this translation is only polynomial, and hence model checking in the new language can be performed in **PSPACE**, the complexity of uniform model checking for first-order formulas. But is this upper bound also a lower one? After all, the model shrinks when we make a jump via the operator $\Diamond\!\!\!\!-$. The same open question of complexity now returns:

What is the complexity of uniform model checking for the $\Diamond$, $\Diamond\!\!\!\!-$ *language?*

Another obvious open questions concerns the complete logic of these operators. Logics like this axiomatize a bit of the meta-theory of modal evaluation plus some natural model operations. One could also add modalities for passing to arbitrary submodels (on finite models, this is the Kleene iteration of $\Diamond\!\!\!\!-$), and other model constructions have been considered as well. E.g., in modern update logics for communication, a public announcement A**!** of an assertion A restricts the current model to only those worlds where A holds. Evaluation of further modal statements about agents' knowledge and ignorance then moves to a *definable submodel*, driven by mixed assertions such as $\Box_{A!}\mathrm{K}_i\phi$. And there are still further examples in the literature. The above is just a start. New questions of model checking and completeness emerge in all such systems.

Disturbing a Game in General

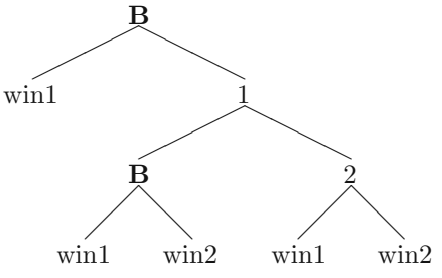
Time for some free association! We turned an algorithm into a game by introducing a disturbing player modeling the action of a hostile environment. But we can also disturb *any game* itself. One simple variant is as follows. Consider a game tree, with one player for convenience – and at the start of each round, let Blocker *prune one game move* from the tree. The effects of this local disturbance are not so dramatic. One can still compute winning positions for Runner in the original game tree, in the same inductive fashion as with Zermelo's algorithm for finding the winning player in finite game trees. The key observation is this:

Fact. At any game node, Runner has a winning strategy against Blocker `iff` he has winning strategies in the subtrees for *at least two* of his moves.

From right to left, this is obvious, as Blocker can only affect at most one of those subtrees in the first round, leaving Runner free to go to the other one. From left to right, if there was at most one such subtree, then Blocker can cut the move to that, and force Runner to choose a move to a losing position. The resulting inductive algorithm for computing winning positions is about as simple as that of Zermelo. But, our sabotaged graph tasks are not like this! Blocker removed an edge from the graph, which cuts travel options for Runner at many stages in the original search space. This is a global action, which amounts to *pruning a whole set of moves simultaneously* from a game tree. The effects of this are not as easy to compute inductively – again reflecting the essentially greater complexity of the new game.

Social Life and Coalition Logic

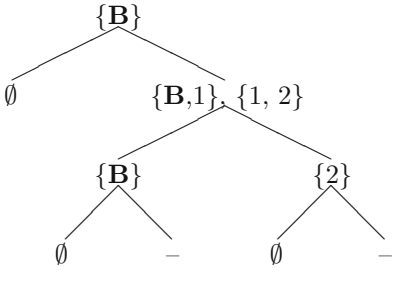
Sabotaging players also make sense in n-person games, whether removing single moves or whole sets of them. The corresponding game transformation produces an $(n+1)$-person game, and the above issue of winning or losing generalizes to *coalitional powers* of groups of players, including Blocker. Here is an example:



Working upward, inductively, one can compute *forcing coalitions* for any proposition p. At bottom nodes satisfying p, these are the empty set of players $\emptyset$ and its supersets. At end nodes not satisfying p, no forcing coalition exists for p. At higher nodes which are turns for player j , the rule is this:

Take all p-forcing coalitions that occur at daughters and add j to them.

To simplify the calculation, *supersets may be suppressed* here – as these represent weaker derived powers of groups. For the above game tree, with p the predicate win1, the minimal p-forcing coalitions would be:

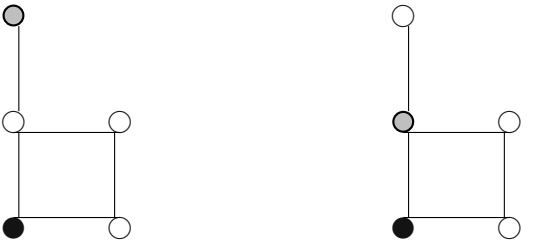


But the logic of sabotage supports more refined new notions, such as Blocker's being able to determine the outcome of the game – in the sense of being able to make either player 1 or player 2 win. Also, players can make a pact with the Devil, and oppose their old antagonists more effectively. Modern modal game logics describe these complex statements of powers for individuals and coalitions in a general way. With our earlier games on graphs **G** , such statements still translate into first-order properties of **G** – and the earlier observation about model checking still applies.

Thus, after the game transformation, many interesting aspects of sabotage become matters of coalitional game theory – and with graph games, even of standard first-order logic.

Receptions, Circulation, and Pacts with the Devil

This mathematical calculus is closer to real life than you might think. Directors of institutes like Jörg need the skill of *circulation at receptions*. Suppose there are 5 positions, with your starting point at the black dot, and mine at the grey one. At each round, we have to shift position: you move first, and I follow. One important skill among academics of high standing is "meet" versus "avoid". Here is a picture of an initial situation plus the situation after one possible round:



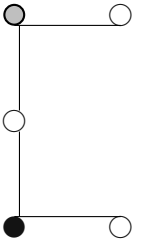
Analysing the strategic situation, one easily finds the following three 'powers':

(a) You can force us to meet,
(b) I can force us to meet,
and therefore, none of us can force 'avoidance' – but still:
(c) You and I *together* can force us to avoid each other,
most easily by cycling back and forth at two disjoint links.

More spectacular avoidance strategies for one player against another occur around cycles, much like sequences of running around obstacles in action movies. Now introduce a Blocker who can take away links, say a manipulative host. The new power structure will depend on the scheduling. Consider first a rule where Blocker starts each round, then you, then me. Then it is easy to see that

Blocker can force us to meet, but also force avoidance.

He forces a meet by cutting links to decrease our room of manoevre, but still in one connected component. He forces us to avoid each other by imprisoning us inside disjoint components. Pacts with the devil make no sense here, as Blocker controls all relevant outcomes anyway. With a rule: "First you, then me, then Blocker", you retain your earlier power to force a meet, Blocker can force the same, and I have no significant powers. A pact with the devil does make sense at the following reception:

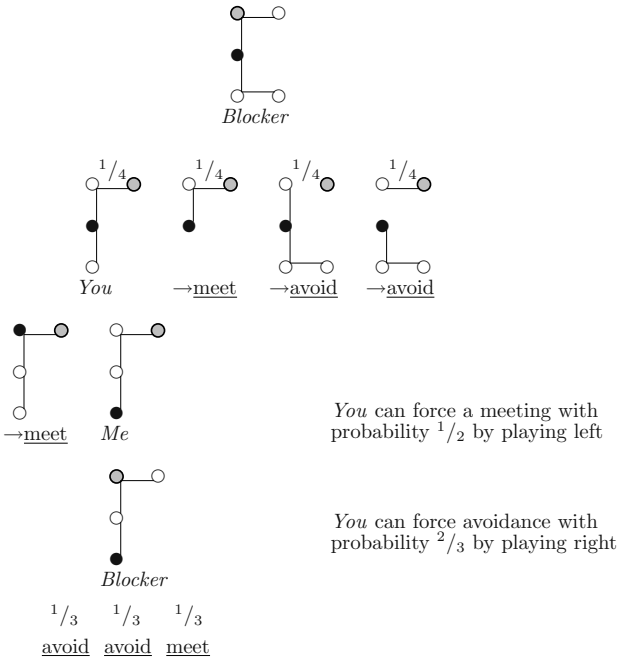


Neither you nor I alone have the power to force a meet here, or force avoidance. But the *coalition* {you, me} can force a meet, and it can also force avoidance – by both going to our nearest end-points: Blocker must then cut us off in the middle. Also, in a pact with Blocker, each of us can force a meet, or avoidance.

These outcomes show that the above general Meet and Avoid strategies for Blocker alone have their limitations. Still, it is easy to see that they will always work if he has *empty moves*, allowing him to wait until we have made our compulsory moves. Another aspect of social algorithmics is potential redesign. Once powers for meeting and avoiding have been computed for a reception scheme, a host might want to design simpler – perhaps cheaper – social events with the same effect. Moreover, these are not mere formal phantasies. When the German president von Weiszäcker visited Groningen University in the early 1980s, we professors at his gala reception were all given a mathematical circulation diagram with exact choreographic instructions on how to move from table to table. I guess the much greater professionalism at our modern Dutch universities also includes instructions on what to *say*...

The Meaning of Sabotage

Any algorithmic task can be 'sabotaged' to create real-life versions in a hostile environment. This can be done by turning it into a game, which can be studied by known techniques. Thus, in Clausewitz' immortal phrasing, game theory is "algorithmics pursued by other means". But there are also other perspectives on these phenomena. In particular, the logical model-checking angle suggests a study of evaluation of first-order logic on structures *which change under evaluation*. E.g., an object might become unavailable once drawn from the domain, or a fact might change when inspected (think of measurement in quantum mechanics). This would be like adding aspects of sabotage to logical evaluation games. Finally, in the realities of modern transport, sabotage comes in a more global and *statistical* fashion. Thus, we might also assume that Blocker is a blind opponent cutting links with equal probability. This will generally improve the situation for Runner, and one can use a Zermelo algorithm to compute his *expected game values*, indicating the degree of deterioration from Runner's prospects in the initial task. E.g., with a random Blocker, here is a piece of the game tree for the second reception, whose schedule was '*You, Me, Blocker*':



This is more like known tasks in Graph Theory, where *random graph problems* may involve removal of nodes and edges. By contrast, our opponents in this essay are maximally malevolent. More positively, canceled connections may come *alive* again in the course of our trip when we add a third player 'Deblocker'. Lots of further complications to investigate! But then, the field is still young.

Even so, one deeper question remains. *Why* does the Dutch railway system behave in its current erratic mode? Instead of thinking negatively about failure and doom here, we can also think positively about the intentions of our political leaders who govern it. For many centuries, the average member of our nation has muddled through life in **P**-like, or at best **NP**-like patterns of behaviour. But now, in the new millennium, the authorities have decided that we have reached a new plateau of intelligence. The Dutch are ripe for **PSPACE** tasks, and we are given a challenging chance to practice these. Similar encouraging experiences are reported from the British Isles. This represents a new stage in human evolution, and I am sure we will see many similar phenomena all across Europe soon!

Acknowledgment

Peter van Emde Boas and Hanno Hildmann provided indispensable support.

Postscript

In the few months between the first posting of this essay and publication, Christof Löding and Philipp Rohde (RWTH Aachen) have answered the main complexity question. Through an ingenious construction, sabotaged Graph Reachability and Traveling Salesman both turn out PSPACE-hard on finite graphs.

Bridging Theorem Proving and Mathematical Knowledge Retrieval

Christoph Benzmüller[1], Andreas Meier[1], and Volker Sorge[2]

[1] FR 6.2 Informatik, Universität des Saarlandes,
Saarbrücken, Germany
{chris,ameier}@ags.uni-sb.de
[2] School of Computer Science,
University of Birmingham, UK
V.Sorge@cs.bham.ac.uk

Abstract. Accessing knowledge of a single knowledge source with different client applications often requires the help of mediator systems as middleware components. In the domain of theorem proving large efforts have been made to formalize knowledge for mathematics and verification issues, and to structure it in databases. But these databases are either specialized for a single client, or if the knowledge is stored in a general database, the services this database can provide are usually limited and hard to adjust for a particular theorem prover. Only recently there have been initiatives to flexibly connect existing theorem proving systems into networked environments that contain large knowledge bases. An intermediate layer containing both, search and proving functionality can be used to mediate between the two.
In this paper we motivate the need and discuss the requirements for mediators between mathematical knowledge bases and theorem proving systems. We also present an attempt at a concurrent mediator between a knowledge base and a proof planning system.

1 Introduction

When sharing knowledge of one database amongst several clients or when accessing several databases by one client it is often necessary to use mediators as middleware components to tailor the provided knowledge to the particular needs of an application. By assigning sharable functionalities into mediator services the high costs of adapting both knowledge servers and requesting client applications to their mutual needs can be avoided. While this insight has become common ground in the development of large client-server systems, only recently a similar phenomenon can be observed in the area of theorem proving.

For being effective tools theorem provers need to be provided with a fair amount of knowledge. In particular interactive theorem provers require large libraries of formalized mathematics or knowledge for verification issues. Building these libraries is a time consuming and tedious activity. In large parts it is also duplicated effort, since many problems require similar theories of basic concepts

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 277–296, 2005.

and therefore most systems require the formalization of roughly equivalent basic knowledge. Recent initiatives try to minimize the knowledge engineering effort by sharing knowledge between different systems. This can be done directly or via distributed networks in which broad knowledge bases of mathematical theories are jointly developed and employed.

The integration of a shared database with one particular client theorem prover can naturally not be as close as in an exclusive connection, where the knowledge base is tailored explicitly for the needs of the particular theorem prover. While the database can provide mainly syntax-based retrieval procedures like regular expression search or simple matchings and unifications, it usually cannot deal with requests that need a semantic search possibly depending on the particular nature and proof context of the requesting client. For instance, it should be possible to retrieve from the database all facts – theorems, lemmas, definitions – containing a certain concept. However, it is unlikely that the database can be queried for a theorem inferring a particular goal in question, since this query also depends on questions such as: What is the logic and consequence relation of the requesting system? What kind of unification is suitable for the request? etc.

One solution to bridge the gap between theorem provers and knowledge base is to inject an intermediate layer of mediator systems whose task is (1) to transmit suitable queries to a knowledge base and (2) to adequately process the received data for the needs of a requesting prover. In this paper we shall examine the situation how knowledge is currently handled and processed in state-of-the-art theorem proving systems. We shall further motivate the need for mediators between mathematical knowledge bases and theorem proving systems and discuss the particular requirements for this kind of middleware components. Finally, we present a first prototype implementation of a concurrent mediator between a knowledge base and a proof planning system.

2 Mediating Mathematical Knowledge

The notion of a mediator was first introduced by Wiederhold [36] in the context of general information systems. Mediators are motivated by the emerging gap between the information requested by an application and the information available in distributed information sources: “Knowing that information exists, and is accessible creates expectations by end-users. Finding that it is not available in a useful form or that it cannot be combined with other data creates confusion and frustration.” Wiederhold distinguishes three layers: the layer of databases and information sources, the layer of independent applications, and between them the layer of mediators where a mediator is a “software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications”. Mediators thus make applications independent of the particular information sources. Generally they comprise heterogeneous functionalities such as transformation and subsetting from information sources, methods to access and merge data from multiple information sources, computations that support abstraction and generalization over underlying data, and

methods to deal with uncertainty and missing data because of incomplete or mismatched sources.

Two concrete mediator approaches are TSIMMIS [30] and KOMET [12]. In TSIMMIS so-called wrappers first convert data from each information source into a common model; they also provide a common query language for information extraction. A mediator now combines, integrates, or refines data from the wrappers, providing applications with a "cleaner view". The *Mediator Specification Language (MSL)* is used to specify mediators declaratively. Similar to TSIMMIS, KOMET is a shell for developing dedicated mediators by means of a declarative language. Employing an annotated logic for the latter it is capable of performing various types of reasoning.

The mentioned approaches address the general problem of information retrieval in distributed information sources. We will now shed some light on mathematical knowledge retrieval in current theorem proving or reasoning systems.

2.1 Current Systems

Traditional Automated Theorem Prover. The first automated theorem proving systems were mainly designed as stand-alone systems not connected to a database of mathematical knowledge such as theorems, lemmas, and definitions. Many modern systems such as OTTER [25], SPASS [35], PROTEIN [4], SETHEO [24], VAMPIRE [31], BLIKSEM [28], and WALDMEISTER [20] still follow this tradition. In order to prove hard mathematical theorems T with these systems assumptions $A_1, \ldots, A_n$ and probably some lemmas $L_1, \ldots, L_m$ have to be carefully chosen by the user in advance. Thus, the problem processed by the prover is: $A_1 \wedge \ldots \wedge A_n \wedge L_1 \wedge \ldots \wedge L_m \Rightarrow T$. Dynamic retrieval of further mathematical facts during proof search is not addressed in the traditional theorem proving context.

An automated theorem prover that supports dynamic knowledge retrieval is TPS [1], which is based on higher order mating search. Its dual instantiation mechanism [8] dynamically requests definitions from TPS's own library and expands them stepwise during proof search. Hence, the TPS user does not have to decide in advance which definitions to expand (and which occurrences of a definition to expand) since this is done by the system at runtime.

An independent library for traditional automated theorem provers is the TPTP [33] library for first order problems. TPTP provides a common basis of problems for the development and testing of automated theorem provers. Problems can be stored in a structured way with respect to their mathematical domain and standard mathematical axiomatizations (e.g., equality axioms for group theory etc.) are provided. New problems can inherit knowledge along the existing structures. With the tptp2x utility the TPTP provides also some mediator functionalities. The tptp2x utility converts TPTP problems from the TPTP format to formats used by existing automated theorem provers (e.g., the OTTER format). However, dynamic retrieval of knowledge from TPTP during a proof attempt has not been addressed yet in traditional automated theorem proving.

Interactive Theorem Prover. Interactive theorem proving environments for mathematics or program verification usually closely integrate proof development and proof manipulation facilities with a proof and knowledge maintenance system in the background. They often provide elaborate mechanisms to maintain, manipulate, and access highly structured knowledge. The knowledge is usually encapsulated in mathematical theories consisting of definitions, axioms, lemmas, and theorems and can be hierarchically arranged with the help of inheritance mechanisms. For instance, a theory of state machines may be based on a theory for integer arithmetic and lists. Additionally proofs and further domain dependent knowledge such as specialized tactics or proof methods may be maintained.

Existing interactive theorem proving environments for mathematics and verification differ concerning how close the knowledge base is integrated. For instance, in the latest NUPRL release, NUPRL LPE [32] (logical programming environment), the library is the central module. It contains definitions, theorems, inference rules, meta-level code (e.g., tactics), and structure objects that can be used to provide a modular structure for the library's contents. A collection of independent, cooperating processes are centered around this library. They include inference engines, user interfaces, rewrite engines, and translators.

While some other systems, like PVS [29], follow a similar approach, there are also systems in which the mathematical library has a less central function and which realize a more loose integration of proof development and knowledge maintenance. In ΩMEGA [5] the mathematical library originally also was interwoven with other parts of the system. In a recent reorganization of the system it became an independent module.

Cooperating Reasoning Systems. In recent years many experiments to integrate reasoning systems have been carried out. For such cooperations the sharing and exchanging of mathematical knowledge is crucial.

One approach to make two systems cooperate is to transform the theory libraries in the format of the one system into the format of the other system. Then knowledge of the former system can be used in the latter system. For instance, [21, 16] describe the cooperation of NUPRL and HOL [19]. Proofs are developed in NUPRL employing HOL libraries and a connection between NUPRL's and some of HOL's packages for adding constants, axioms, and theorems. Crucial for this cooperation is the import of HOL theories into NUPRL such that the NUPRL user gains full access to them. The main problem is the translation of concepts in the logic of the one system into the logic of the other system.

Other approaches of cooperating systems do not transform concepts at the theory level but do transform proofs. For instance, [9] describes the interface between HOL and the proof planner CLAM [10] which is a system specialized on induction. CLAM is treated as a black box to which HOL passes goals to be proved automatically. The approach avoids the modification of CLAM in order to suit the classical higher order logic used in HOL. Instead, correspondences between mathematical knowledge and structures in both systems are established and exploited. That is, both systems maintain their own database of definitions, lemmas, induction rules, wave rules, etc. and corresponding concepts are identi-

fied by their names. When CLAM returns a proof plan to HOL then the mapping of the names is used to guide the construction of a corresponding proof in HOL. A similar approach is also used in the interface between ΩMEGA and TPS [6].

Currently, there are no approaches of cooperating systems that rely on a jointly developed, shared mathematical database. However, the described cooperations of NUPRL and HOL as well as HOL and CLAM demonstrate that recent approaches strive in this direction. In neither approaches is the knowledge translation done by independent mediators in the sense of Wiederhold, but instead encoded in one or the other system. However, there are already independent mediators for translating proofs from more machine oriented calculi of automated theorem provers into the more human oriented formalisms of interactive reasoning systems. A system specialized on this type of transformation is TRAMP [26], which translates the output of several automated theorem provers (e.g., OTTER, SPASS, WALDMEISTER) for first order logic with equality into natural deduction proofs at the assertion level.

General Mathematical Databases. There have also been approaches for universal mathematical knowledge bases that are not connected to a particular system but that want to offer the infrastructure for a repository of formalized mathematics. Most notable is probably the MIZAR library [34], which is being assembled for more then two decades now. It contains more than 2 thousand definitions of mathematical concepts and about 20 thousand theorems. The retrieval of these facts is mainly text-based and thus of rather limited use for a concrete client theorem prover. Therefore, a suitable postprocessing of MIZAR's data is always necessary to actually apply the collected knowledge.

Around 1994, the "QED Manifesto" [2] was put forward, which advocates building up a mathematical knowledge base as a kind of "human genome project" for the deduction community. Unfortunately, the vision has failed to catch on in spite of a wave of initial interest.

Only recently the mathematical library MBASE [18] emerged as a spin-off of the ΩMEGA system. The outsourcing of ΩMEGA's database and its separation from the inference mechanisms of the system inspired the development of a mathematical library that wants to serve as a distributed repository of mathematical knowledge for other client systems as well. Thus, MBASE is independent of a particular deduction system or a particular logic. Although MBASE aims at providing elaborate, partially semantic-based retrieval facilities, it its current state of development – the first working prototype has been released just recently – it is difficult to assess whether this will be general enough to suit all the needs of a requesting client.

Networks of Mathematical Systems. As the number of differently specialized reasoning systems is growing, the idea of cooperation between those reasoners catches more and more on. This in turn has led to the development of several networks that provide the necessary infrastructure to easily connect different reasoning systems as distributed mathematical services. Examples of system networks are PROSPER [15], LOGICBROKER [3], and MATHWEB [17].

The latter currently provides 22 mathematical services such as theorem provers, Computer Algebra Systems, model generators, and also the prototype of the MBase database.

It is predictable that in the future more and more systems will cooperate and exchange mathematical knowledge via networks of mathematical services. Reasoning systems will request knowledge fractions from shared databases and probably add new or modify existing knowledge chunks. Hence mediating mathematical knowledge will become an increasingly important topic.

2.2 Mediating Requests for Applicable Assertions

In the rest of the paper we focus on the more concrete problem of mediating the retrieval of applicable assertions. Assertion is a collective name for definitions, theorems, and lemmas which we assume to be stored in a database. As part of the mathematical theory assertions can be crucial information for the success of a proof attempt of a theorem prover. For instance, the application of a suitable assertion can dramatically ease and shorten the proof construction.

However, the search for applicable assertions is a non-trivial task. A database may contain thousands of assertions, how can then the retrieval of applicable assertions be efficiently realized? Somehow we have to filter and structure the applicable assertions. Thereby, the potential filtering and structuring criteria range from simple syntactic information to complex semantical properties. As an example consider the proof goal $|(f(x)+a)-(g(x)+a)| \leq \epsilon$. The first information we can use to filter assertions is the syntactic information of the occuring defined symbols $|.|, +, -, <$. The symbols can be employed to identify assertions containing at least one of these symbols where the most promising assertions could be those who contain several of the defined symbols. Similarly, we can use the information on the mathematical domains the investigated subgoal belongs to. Thus in our case we would collect the assertions that belong to the domains *real* or *ordered fields*. A stronger (i.e., more restricting) but more costly filter criterion is to find all assertions which unify with the concrete proof goal. In a higher order context or in case we are interested in theory unification (e.g., associativity, commutativity, and distributivity of '+'), however, we then quickly face undecidable filter criteria. Even more complex would be to identify all assertions from that a goal is deducible with respect to further facts given in the current proof.

Since filter and structuring criteria can become very complex and even undecidable the question is where those criteria should be applied? There are two "extreme" scenarios:

In the Theorem Prover: The theorem prover requests once or even in each proof step a set of potentially applicable assertions from the knowledge bases using requests that are rely on mainly simple syntactical criteria. It then structures the received, probably very heterogeneous data and analyses itself within its main theorem proving loop whether the candidate assertions are indeed applicable. The applicable ones are then integrated as additional hypotheses to be considered for the subproblem in the proof search.

In the Knowledge Base: The knowledge bases completely handle the search for actually applicable assertions with respect to a proof context received from a requesting theorem prover. Then they pass only the applicable assertions to the theorem prover. A task that nevertheless remains for the theorem prover is to structure and merge (e.g., remove duplicates) the assertions received this way from different knowledge bases.

We argue that these two extreme scenarios are not suitable in a network of heterogeneous reasoning systems and mathematical knowledge sources. The interleaving of assertion filtering and structuring with the main theorem proving loop in the first scenario is a rather ineligible option for both automatic and interactive theorem proving. For non-trivial mathematical problems the sets of potentially applicable assertions easily become very large. Consequently the applicability checks can dramatically slow down the theorem proving process. In case of undecidable filter criterions the theorem prover would even have to decide when to interrupt the filtering process. The second scenario, in which the database does the main filtering and structuring, presupposes practically infeasible, complex, and logic and context sensitive search facilities in the knowledge bases. That is, a knowledge base would have to support the different logics and consequence relations of all requesting theorem provers. Another problem for the knowledge bases is that numerous, simultaneous requests from different theorem provers could greatly reduce the performance of the knowledge base if too complex or even undecidable filter criteria are employed.

Although the "extreme" scenarios would probably not exist in their pure form, they demonstrate that both theorem provers and knowledge bases should be kept free of the respective other's task. In particular, to avoid adjusting the theorem prover to the abilities of the database or, conversely, tuning the knowledge retrieval for the needs of a particular theorem prover, we suggest an intermediate layer of mediators to interface between theorem provers and knowledge bases.

2.3 Requirements of a Mediator

Figure 1 depicts a mediator between a theorem prover and a database. Foremost, the mediator acts as an interface and performs translation tasks. Therefore, the theorem prover needs no knowledge about how to access the database; it only has to pass queries to the mediator. The mediator creates then suitable requests for the database. Moreover, the theorem prover does not have to accept the data from the database in its actual formalism, rather the mediator can pass data to the theorem prover in a suitable formalism. The interface functionality on the one hand enables a database to serve several different client theorem provers. On the other hand a single theorem prover can also easily access several databases: The theorem prover has still to communicate via only one mediator, which passes the requests of the mediator to the different databases in their respective formalisms and returns the data of different databases to the theorem prover in a unique formalism.

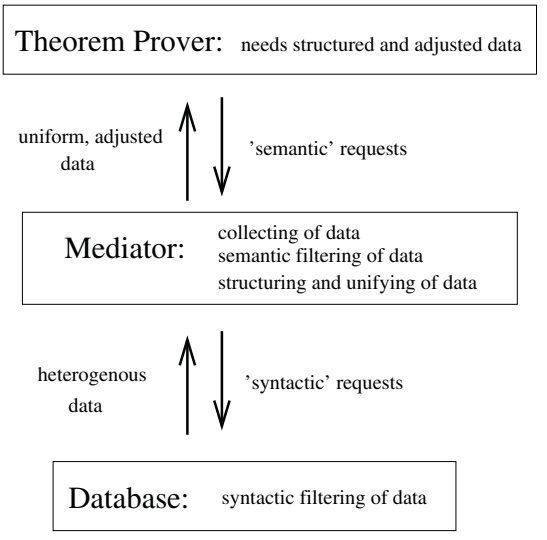


Fig. 1. A mediator between theorem prover client and mathematical knowledge base.

Apart from the simple translation functionality, the mediator should also combine data retrieval mechanisms with theorem proving functionality based on the requirements of a requesting client. The mediator processes the data retrieved from the databases and provides elaborate filtering functionalities that are adjusted to the particular needs of this theorem prover. Concretely it processes the received data to identify portions that are suitable with respect to the current proof context of the theorem prover. Therefore, information about the logical context of the theorem prover is part of the request. The mediator can then choose more appropriate semantic filters with respect to this information. Additionally, the mediator can also combine heterogeneous subprocesses such as structuring of the retrieved data, merging of data retrieved from multiple databases (i.e, removal of duplications), support of abstraction and generalization, dealing with inconsistent data, etc.

Note, that the picture in Fig. 1 gives a rather high-level view on the connections between mediator, theorem prover, and database. It is reasonable to have all three as separated processes, i.e., to enable the theorem prover to proceed with proof search without having to wait for the mediator to terminate its search for applicable assertions. However, the boundaries between theorem prover functionalities, mediator functionalities, and database functionalities in concrete applications may not be as clear as in this picture. In general it is the job of the mediator to apply elaborate filters as well as to structure and unify the data. However, concrete theorem provers or databases may already offer some if these functionalities (e.g., it is planned to implement a unification algorithm with associativity and commutativity in MBase). In such a case the mediator should know this and employ such facilities.

In our concrete scenario the mediator should request assertions from the database and pass only applicable ones to the theorem prover. To check for the

applicability of theorems various algorithms can be employed, for instance, first- and higher-order matching, first- and higher-order unification, restricted forms of unification or matching such as higher-order pattern matching, theory unification or matching where the considered theory depends on the incoming problem, other domain or theory specific algorithms and filters consisting of simple deductive processes adjusted to the requesting theorem provers. The mediator should have all these algorithms at its disposal; however, for concrete requests it should be *parameterizable*. That is, information of the concrete algorithms it should employ are part of a request of the theorem prover. Since some of the algorithms are very complex or even undecidable the mediator should be able to employ them *concurrently*. Then assertions whose applicability can be quickly determined with simple, deterministic algorithms are not blocked by assertions whose applicability test requires non-trivial computations or deductions. This enables also an *any-time character* of the mediator; that is, the more time the mediator has to compute a response the more and probably even better suited assertions it can suggest.

In order to meet these requirements we propose a distributed, concurrent architecture for the mediator. In the next section we shall present a first attempt at such a mediator in a proof planning scenario.

3 An Example Architecture and Application

In this section we present the concrete implementation of a mediator between a theorem prover and a mathematical database and its application in a proof planning environment. We shall firstly introduce the particularities of assertion applications in proof planning before we explain the adaption of the Ω-ANTS [7] suggestion mechanism to a distributed, concurrent mediator system and its concrete application to an example from finite algebra.

3.1 Motivation: Assertion Retrieval

Huang has identified the *assertion level* as a well defined abstraction level for natural deduction proofs [22, 23]. Proofs at assertion level are composed of the direct application of assertions, like theorems, axioms, and definitions.

To clarify the notion of assertion application we pick one of Huang's examples as given in [23]. An assertion application is for instance the application of the *SubsetProperty*

$$\forall S_1 . \forall S_2 . S_1 \subset S_2 \equiv \forall x . x \in S_1 \Rightarrow x \in S_2$$

in the following way:

$$\frac{a \in U \quad U \subset F}{a \in F} \; Assertion(SubsetProperty)$$

The direct application of the assertion is thus an abbreviation for a more detailed reasoning process on the calculus level; that is, the explicit derivation of the goal

$a \in F$ from the two premises by appropriately instantiating and splitting the *SubsetProperty* assertion.

In the ΩMEGA system assertions are applied using a specialized *Assertion* tactic. Its purpose is to derive a goal from a set of premises with respect to a theorem or axiom. It thus enables the more abstract reasoning at the assertion level with respect to given assumptions. We can depict the assertion tactic as a general inference rule in the following way

$$\frac{Prems}{Goal}\ Assertion(Ass)$$

where *Prems* is a list of premises, *Goal* is the conclusion and *Ass* is the assertion that is applied.

3.2 Assertion Application in Proof Planning

Proof planning [11] considers mathematical theorem proving as planning problem where an *initial partial plan* is composed of the proof *assumptions* and the theorem as *open goal*. A proof plan is then constructed with the help of abstract planning steps, called *methods*, that are essentially partial specifications of tactics known from tactical theorem proving. In order to ensure correctness, proof plans have to be executed to generate a sound calculus level proof. The proof planner generally follows a depth first or iterated deepening search strategy, which can be guided by certain heuristics implemented in *control rules*. Methods are tested sequentially and if possible they are immediately applied. In case the proof attempt gets stuck the planner backtracks.

Traditionally in proof planning assertions are applied using a generic method which essentially corresponds to the proof rule displayed in the preceding section. The number and types of assertions considered is usually heuristically limited by a control rule. The method is applicable if one of the considered assertions is applicable to the given goal. In particular, assertions usually are applied backwards in proof planning. That is, to close a goal the planner searches for an assertion whose application to a set of premises deduces the goal; then the premises are inserted as new subgoals. This requires that each assertion in question or at least some part of it is matched with the current goal, which is usually done sequentially, i.e., one by one. Assertions can be applied in different ways. For instance, the assertion $A \Rightarrow B$ can be applied backwards to reduce a goal that matches with B to the new subgoal A or it can be applied backwards to reduce a goal that matches with $\neg A$ to a new subgoal $\neg B$ when applied with respect to its contrapositum. Thus, in order to be as complete as possible the assertion method in ΩMEGA checks all possible directions in which assertions can be applied where each check of a direction comprises a matching of the goal with some parts of the assertion. Naturally, in order to keep method and thus assertion application feasible, matching has to be restricted. For instance, the generic method is equipped with a first order matching algorithm, only. However, there can exist other, additional methods to apply theorems that are better tailored to the needs of a specific set of assertions and hence can use more complicated,

albeit decisive algorithms for determining applicability. Apart from the decidability problems and the lack of support for more complicated matching schemes it is also quite infeasible to check applicability of a large number of assertions in each step of the proof planning process: As discussed already in Sec. 2.2 the applicability checks dramatically slow down the proof planning process.

A second drawback of the direct integration of assertion application into the main proof planning process is the lack of flexibility to adjust the assertion application to the state of the knowledge available. Usually, the proof planner has heuristical information on what assertions it should consider when proof planning in a certain domain. This information is generally directly linked with the knowledge base containing the assertions. Thus, the control unit of the proof planner itself requests certain assertions regardless of the current state of the knowledge base. While some of the requested assertions might not even be contained in the knowledge base, there might be other more suitable ones that are, however, not requested. Moreover, the extension of these heuristics is rather cumbersome and again knowledge base dependent.

Therefore, an ideal support for the overall proof planning process is to have a flexible mediator that adjusts itself both to the requirements of the proof planner and the current state of the knowledge base. Moreover, the mediator should free the proof planner from the encumbering task of constantly checking the applicability of assertions in each single step.

3.3 Using Ω-ANTS as a Mediator

As a mediator between the proof planner and the knowledge base we employ the hierarchical blackboard architecture Ω-ANTS [7] which supports both distribution and concurrency.

Ω-ANTS was originally conceived to support interactive theorem proving in ΩMEGA. It provides the user with information about which inference steps are applicable in the actual proof situation. In the Ω-ANTS context, all inference rules such as calculus rules, tactics, or planning methods are uniformly regarded as sets of premises, conclusions, and additional parameters

$$\frac{Prems}{Cons}\ I(Params).$$

The elements of these three sets generally have some dependencies amongst each other. To apply an inference rule at least some of its arguments have to be instantiated by elements of the given proof context, where the arguments that are actually instantiated determine the direction in which the inference rule is applied. The task of the Ω-ANTS architecture is now to determine the possible applications of inference rules by computing instantiations for their arguments.

The architecture consists of two layers of blackboards: The lower layer of the architecture consists of a set of *rule blackboards*, one for each inference rule. We view the knowledge sources of these blackboards as society of agents (i.e., we have one society for each inference rule) since they are realized in independent, concurrent processes. Their task is to search the current partial proof for partial

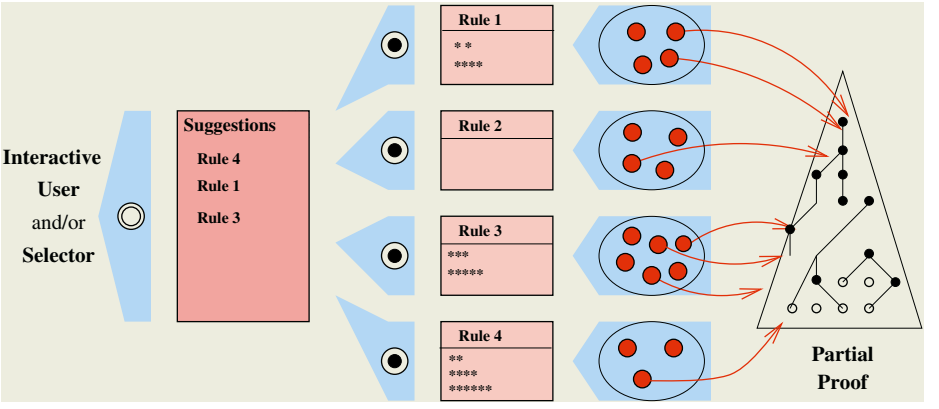


Fig. 2. The original Ω-ANTS architecture.

argument instantiations for the inference rule. They communicate via their rule blackboard and can cooperate by adding further specification to a partial argument instantiation other agents have already placed on the blackboard. Each rule blackboard is monitored by one agent that reports the heuristically prefered partial argument instantiation to the suggestion blackboard, which comprises the upper layer of the architecture. This blackboard accumulates a set of inference rules that are applicable in the current proof state and which are subsequently passed to the user.

A graphical representation of Ω-ANTS architecture is given in Fig. 2. Agents are displayed by circles, agent societies are grouped in elliptic frames, and blackboards are displayed by boxes. In the figure the architecture is rotated; that is, the lower layer with rule blackboards and their respective agent societies are on the right hand side whereas the upper layer with the suggestion blackboard is on the left hand side.

We adapt Ω-ANTS to concurrently retrieve applicable assertions during proof planning by distributing the applicability checks for sets of assertions to several agents. Like in the original Ω-ANTS architecture we want to compute, now in particular argument instantiations for applicable assertions. Instead of a layer of rule blackboards we provide for this a layer of assertion blackboards. Again with each blackboard of this layer an agent society is associated. Moreover, with each assertion blackboard a cluster of assertions is associated, which consists of related assertions applicable to subgoals that share a certain property. The agent society of an assertion blackboard is responsible to check the applicability of the assertions belonging to its cluster. Thus the agents search both the current partial proof and the associated assertion cluster. As in the original Ω-ANTS system they cooperate via the blackboards by exchanging partial argument instantiations for assertion applications. Also similar to the original Ω-ANTS system complete argument instantiations are passed to the upper layer and then to the proof planner.

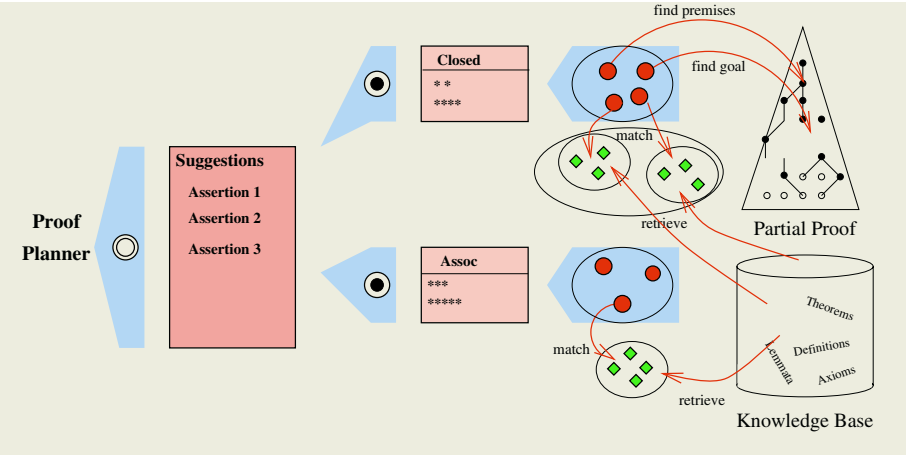


Fig. 3. The use of Ω-ANTS as a mediator.

Figure 3 shows the adapted architecture. It differs essentially from the original architecture in the point that each agent society on the lower level has one cluster of assertions associated. These clusters are depicted below the respective agent societies and the single assertions are represented as diamonds.

Each agent society consists of three types of agents: one filter agent, one or several retrieval agents, and one premise agent. During the proof planning process, first the filter agents look-up the current partial proof and search for open subgoals that could be suitable for their respective assertion cluster. If the filter agent succeeds for a subgoal it places a partial argument instantiation on its blackboard containing the subgoal as only element. For such entries on the blackboard the retrieval agents become active, look-up the associated assertion cluster, and attempt to find actually applicable assertions, which is usually done with some matching algorithm that is part of the specification of the retrieval agent. If retrieval agents are successful they suggest the matching assertions as applicable and add the theorems to the partial argument instantiations. This triggers the premise agent, which examines each suggested assertion if its application will lead to new open subgoals. In this case the premise agent tries to identify whether the proof context already contains presuppositions that can justify the premises of the assertion. Complete argument instantiations are then passed as suggestions to the proof planner. Each individual suggestion contains information on the investigated subgoal, an identified applicable assertion, and presuppositions justifying the premises of the assertion.

Filter agent and retrieval agents enable a separation of simple and difficult tests. The filter agent usually performs only simple checks, for instance, whether a goal contains a certain concept such that the assertions in the cluster deal with this concept. The retrieval agents employ more expensive applicability checks such as first order matching, higher order matching, or even full higher order theorem proving. This separation of pre-selection of goals by the filter agent and the main check by the retrieval agents prevents the application of complicated

matchings and unifications to check the applicability of assertions to goals which obviously will fail. Moreover, a society can have more then one retrieval agent, which can employ different algorithms and are possibly considering different subsets of assertions. This is sketched in Fig. 3 by the two subclusters that comprise the upper assertion cluster. Different retrieval agents allow for further separation of simpler and more complicated checks. Since all agents are separate processes simple checks are not blocked by complicated checks that get stocked.

The retrieval agents comprise a further functionality, they establish the interface to the database and form the associated clusters of assertions dynamically at runtime. Technically, this is realized as follows: Each retrieval agent is equipped with specifications about the type of assertions it can process. At runtime the retrieval agent sends a request to the knowledge base to receive a set of assertions that comply with the specification. This request can be adapted to the format and abilities of the respective contacted database. Furthermore, selecting the assertions via specifications enables a more refined selection of assertions and makes this selection independent of explicit references to assertions or a particular database. The database queries are periodically repeated so when new assertions become available they are automatically fitted into the existing clusters.

The adapted Ω-ANTS architecture as mediator combines both theorem proving and database functionality. On the one hand the filter and premise agents search in the given partial proof on the theorem prover side. On the other hand the retrieval agents request assertions from the knowledge base and model advanced, theorem prover dependent retrieval functionality. However, while the formation of theorem clusters is already a dynamic process the agents themselves have to be explicitly specified.

3.4 A Concrete Application

The example we present is taken from a case study on the proofs of properties of residue classes. In this case study we apply ΩMEGA's proof planner to classify residue class sets over the integers together with given binary operations in terms of their basic algebraic properties. The case study is described in detail in [27]. We concentrate here on how Ω-ANTS determines the applicability of assertions in this context. We consider the first step in the proof of the theorem

$$Conc. \vdash Closed(\mathbb{Z}_5, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, (x \bar{*} y) \bar{+} \bar{3}_5).$$

It states that the given residue class set $\mathbb{Z}_5$ is closed with respect to the operation $\lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, (x \bar{*} y) \bar{+} \bar{3}_5$. Here $\mathbb{Z}_5$ is the set of all congruence classes modulo 5, i.e., $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$. $\bar{*}$ and $\bar{+}$ are the multiplication and addition on residue classes.

Among the theorems we have for the domain of residue classes there are some that are concerned with statements on the closure property. In particular, we have the following six theorems:

$ClosedConst : \forall n{:}\mathbb{Z}_{\blacksquare} \forall c{:}\mathbb{Z}_{n\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, c)$
$ClosedFV : \quad \forall n{:}\mathbb{Z}_{\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, x)$
$ClosedSV : \quad \forall n{:}\mathbb{Z}_{\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, y)$

$$ClComp\bar{+}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_{1\blacksquare}\forall op_{2\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{+}(x\ op_2\ y))$$

$$ClComp\bar{-}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_{1\blacksquare}\forall op_{2\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{-}(x\ op_2\ y))$$

$$ClComp\bar{*}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_{1\blacksquare}\forall op_{2\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{*}(x\ op_2\ y))$$

The theorems *ClosedConst*, *ClosedFV*, and *ClosedSV* talk about residue class sets with simple operations whereas $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ are concerned with combinations of complex operations. The difference between the groups of theorems is that the applicability of the former can be checked with slightly adapted first order matching whereas for the latter we need higher order matching. For example, when applying the theorem $ClComp\bar{+}$ to our problem at hand the required instantiations are $op_1 \leftarrow \lambda x_{\blacksquare}\lambda y_{\blacksquare}x\bar{*}y$ and $op_2 \leftarrow \lambda x_{\blacksquare}\lambda y_{\blacksquare}\bar{3}_5$, which cannot be found by first order matching. However, since we are concerned only with a distinct set of binary operations and their combinations, we can keep things decidable by using a special, decidable algorithm, which matches the statements of the theorems $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ with nested operations on congruence classes.

In Ω-ANTS we have the agent society as depicted in Figure 4 for the cluster comprising the theorems given above. The filter agent $\mathfrak{F}$ searches for possible conclusions that contain an occurrence of the *Closed* predicate. $\mathfrak{F}$ writes respective suggestions of goals to the blackboard. We then have two retrieval agents, $\mathfrak{R}_1$ and $\mathfrak{R}_2$, that try to match the theorems. $\mathfrak{R}_1$ tries to match the theorems *ClosedConst*, *ClosedFV*, and *ClosedSV* to the formulas suggested by $\mathfrak{F}$ using first order matching. $\mathfrak{R}_2$ uses the special algorithm instead of matching the theorems $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ conventionally. $\mathfrak{R}_1$ and $\mathfrak{R}_2$ have additional acquisition predicates specifying that the agents can acquire theorems whose conclusions have *Closed* as the outermost predicate. $\mathfrak{R}_1$ furthermore requires that the theorem conclusion contains a simple, constant operation while $\mathfrak{R}_2$ expects a complex operation. The acquisition predicate serves to retrieve appropriate theorems from the knowledge base initially and dynamically at runtime if new theorems are added. $\mathfrak{R}_1$ and $\mathfrak{R}_2$ place new extended suggestions on the blackboard for each applicable theorem they detect. The last agent is the

$\mathfrak{F}$ = { *Goal*: *Goal* contains the *Closed* predicate}

$\mathfrak{R}_1$ = { *Thm*: Conclusion matches *Goal* with first order matching}
{ *Acquisition*: Conclusion contains *Closed* as outermost predicate and a constant operation }

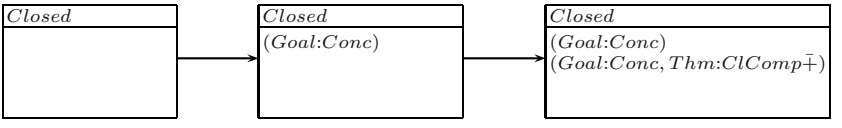
$\mathfrak{R}_2$ = { *Thm*: Conclusion matches *Goal* with special algorithm}
{ *Acquisition*: Conclusion contains *Closed* as outermost predicate and a binary operation }

$\mathfrak{P}$ = { *Prem*: The nodes matching the premises of *Thm*}

Fig. 4. Agent society for the *Closed* theorem cluster.

premise agent $\mathfrak{P}$, which has an algorithm to extract the necessary premises from a theorem suggested by $\mathfrak{R}_1$ or $\mathfrak{R}_2$, if there are any. For instance, if the $ClComp\bar{+}$ theorem has been successfully matched the agent would extract the succedent of the implication (i.e., $Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)$) as well as the single conjuncts comprising the succedent. The agent $\mathfrak{P}$ then tries to find appropriate lines in the current proof containing these premises.

For our concrete example theorem the information that accumulates on the command blackboard for the *Closed* theorem cluster is as follows:



First $\mathfrak{F}$ detects an occurrence of the *Closed* predicate in the given goal *Conc* and adds an entry suggesting it as instantiation for *Goal* to the blackboard. With this entry, $\mathfrak{R}_1$ and $\mathfrak{R}_2$ start matching their respective theorems to *Conc*. $\mathfrak{R}_2$ is successful with the $ClComp\bar{+}$ theorem and adds the matched theorem as suggestion. Then $\mathfrak{P}$ starts its search; for our example it is looking for premises of the form $Closed(\mathbb{Z}_5, \lambda x_{\blacksquare} \lambda y_{\blacksquare} \bar{3}_5)$ and $Closed(\mathbb{Z}_5, \lambda x_{\blacksquare} \lambda y_{\blacksquare} x \bar{*} y)$.

4 Outlook

This paper discussed the possible role of mediators between theorem provers and mathematical knowledge bases. Mediators should provide all kinds of functionalities that neither can be provided by general, shared databases nor should be integrated in the main proving process of client theorem provers. As a first attempt at a concrete mediator system we have presented an adaptation of the Ω-ANTS blackboard architecture to retrieve applicable assertions for ΩMEGA's proof planner. The architecture combines both theorem proving and database functionality. Moreover, it enables concurrent computations and supports anytime character in the way that applicable assertions are immediatly reported to the theorem prover before all computations are finished. However, the architecture is only partly parametrizable and flexible. In particular, agents have to be specified and arranged explictly. The developer of the Ω-ANTS mediator has to specify which matching and unification algorithms are applied via agents to which assertions (that are collected also via the agents). That is, the agents can not arrange flexibly to new societies. Thus the Ω-ANTS mediator can not process new assertions that do not fit into the existing societies. One way to overcome this, is to develop more general specifications how to identify applicable assertions and to parameterize these, for instance, with respect to a given mathematical theory.

Related to our assertion retrieval scenario is the work of Dahn *et al.* [14]. They apply the ILF system [13] to equip the Computer Algebra System MATHEMATICA [37] with the possibility to retrieve theorems from a part of the mathematical library of MIZAR. The approach is motivated by the observation that an ordinary

search for text strings is an unsatisfactoring retrieval approach since the theorem might be stated slightly different in the database (e.g., different variable names) or it might merely be inferable from other theorems and simple properties. Dahn *et al.* therefore employ ILF as a mediator that performs a semantical search for suitable theorems supported by the first order provers connected to it. For this, first a set of candidate theorems is selected based on the signature of the request using conventional database techniques. The candidates are then extended by some auxiliary axioms and several provers are started competetively to prove that the requested theorem follows from the extended set. If a proof is found it is inspected to determine the library theorems actually used in it.

While the work has a different direction with respect to the actual use of the retrieved mathematical knowledge it nevertheless complys with our desiderata for mediator systems. The motivation is to remedy the shortcomings of current mathematical knowledge bases and the standard database retrieval is indeed enhanced using more elaborate, in particular theorem proving, techniques.

Apart from the context of theorem proving, mediators can also be used to make mathematical knowledge available to a wide range of other applications such as Computer Algebra Systems, tutor systems, electronic publishing, web browsers, or even human mathematicians. Mathematical knowledge management and its application is a field that is just emerging and we believe that the design and implementation of mediator systems will play an important role in this field.

Acknowledgements

We would like to thank Thomas Hillenbrand and Andrew Adams who provided us with insights into the working of some of the cited systems.

References

1. Peter B. Andrews, Matthew Bishop, and Chad E. Brown. TPS: A theorem proving system for type theory. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in LNAI, pages 164–169, Pittsburgh, 2000. Springer.
2. Anonymous. The qed manifesto. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 238–251, Nancy, 1994. Springer.
3. Alessandro Armando and Daniele Zini. Towards interoperable mechanized reasoning systems: the logic broker architecture. In A. Poggi, editor, *Proceedings of the AI*IA-TABOO Joint Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, Parma, Italy, 2000.
4. P. Baumgartner and U. Furbach. PROTEIN: A prover with a theory extension interface. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 769–773, Nancy, 1994. Springer.
5. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 252–255. Springer, 1997.

6. Christoph Benzmüller, Matthew Bishop, and Volker Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5(3):188–207, March 1999. Special issue on Integration of Deduction System.
7. Christoph Benzmüller and Volker Sorge. Critical Agents Supporting Interactive Theorem Proving. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, Proc. of the 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*, pages 208–221, Évora, Portugal, 21–24, September 1999. Springer.
8. Matthew Bishop and Peter Andrews. Selectively instantiating definitions. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, volume 1421 of *LNAI*, pages 365–380. Springer, 1999.
9. Richard Boulton, Konrad Slind, Alan Bundy, and Mike Gordon. An interface between CLAM and HOL. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, number 1479 in LNCS, pages 87–104, Canberra, 1998. Springer.
10. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
11. Alan Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE–9)*, volume 310 of *LNCS*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany.
12. J. Calmet, S. Jekutsch, P. Kullmann, and J. Schü. KOMET: A system for the integration of heterogenous information sources. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 1997.
13. B. I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ilf. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 57–60. Springer, 1997.
14. Ingo Dahn, Andreas Haida, Thomas Honigmann, and Christoph Wernhard. Using mathematica and automated theorem provers to access a mathematical library. In *Proceedings of the CADE-15 Workshop on Integration of Deductive Systems*, 1998.
15. Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The prosper toolkit. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS-2000*, LNCS, Berlin, Germany, 2000. Springer Verlag.
16. Amy Felty and Douglas Howe. Hybrid interactive theorem proving using Nuprl and HOL. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, number 1249 in LNAI, pages 351–365, Townsville, 1997. Springer.
17. A. Franke and M. Kohlhase. MATHWEB, an agentbased communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, volume 1631 of *LNAI*, pages 217–221, Trento, 1999. Springer.
18. A. Franke and M. Kohlhase. MBase: representing mathematical knowledge in a relational data base. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 455–459, Pittsburgh, 2000. Springer.
19. Mike J. C. Gordon and Tom F. Melham. *Introduction to HOL*. Cambridge University Press, Cambridge, United Kingdom, 1993.

20. Thomas Hillenbrand, Andreas Jaeger, and Bernd Loechner. WALDMEISTER: Improvements in performance and ease of use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 232–236, Trento, 1999. Springer.
21. Douglas J. Howe. Importing mathematics from HOL in Nuprl. In J. von Wright, J.Grundy, and J. Harrison, editors, *Proceedings of Theorem Proving in Higher Order Logics*, number 1125 in LNCS, pages 267–282. Springer, 1996.
22. X. Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
23. X. Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 738–752, Nancy, 1994. Springer.
24. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
25. William McCune. OTTER 2.0. In M. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, number 449 in LNAI, pages 663–664, Kaiserslautern, 1990. Springer.
26. Andreas Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 460–464, Pittsburgh, USA, 2000. Springer, Germany.
27. Andreas Meier, Martin Pollet, and Volker Sorge. Classifying Isomorphic Residue Classes. In R. Moreno-Diaz, B. Buchberger, and J.-L. Freire, editors, *A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, volume 2178 of *LNCS*, pages 494 – 508, Las Palmas, Spain, 2001. Springer.
28. H. De Nivelle. *Bliksem 1.10 User Manual*. MPI Saarbruecken, 1999.
29. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in LNAI, pages 748–752, Saratoga Springs, 1992. Springer.
30. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, 1996.
31. Alexandre Riazanov and Andrei Voronkov. VAMPIRE. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 292–296, Trento, 1999. Springer.
32. S.Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The nuprl open logical environment. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 170–176, Pittsburgh, 2000. Springer.
33. Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 252–266, Nancy, 1994. Springer.
34. Andrzej Trybulec and Howard Blair. Computer Assisted Reasoning with MIZAR. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 26–28, Los Angeles, CA, USA, August 18–23 1985. Morgan Kaufmann, San Mateo, CA, USA.

35. Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs, Engel Thorsten, Enno Keen, Christian Theobalt, and Dalibor Topic. SPASS version 1.0.0. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 378–382, Trento, 1999. Springer.
36. Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.
37. Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, fourth edition edition, 1999.

Formal Description of Natural Languages: An HPSG Grammar of Polish

Leonard Bolc

Institute of Computer Science,
Polish Academy of Sciences, Warsaw

1 Introduction

In this paper we present a Head-driven Phrase Structure Grammar (HPSG) grammar of Polish – the result of one of the few attempts (e.g., [Szp86], [Świ92]) to build formal and computationally tractable grammars of Polish. The choice of the formalism used was motivated by several promising features of the formalism which we will present shortly.

The research concerning HPSG description of Polish started in 1994, when members of the Linguistic Engineering Group of Institute of Computer Science, Warsaw, have undertaken research aimed at the description of the large subset of Polish syntax in the terms of this formalism. At the beginning of the work separate syntactic issues were worked up and some theories based on the fundamental HPSG theory described in [PS94] were formulated. The need of the coherent theory of Polish syntax which can become a foundation of the implementation of a relatively large Polish grammar led to further work aimed at integration of all subtheories. The effect of these efforts is the book "Formalny opis języka polskiego. Teoria i implementacja" (Formal description of Polish. Theory and implementation.) by Adam Przepiórkowski, Anna Kupść, Małgorzata Marciniak and Agnieszka Mykowiecka. This paper is a short presentation of the results of the efforts to describe Polish within HPSG formalism included in that book.

HPSG was developed as a comprehensive linguistic formalism for work on syntax, morphology and semantics, as well as phonology and pragmatics. It is a monostratal theory of language: there are no derivations transforming one grammatical structure into another. Any grammatical structure is well-formed if it simultaneously satisfies all constraints that the grammar imposes. Further, all constraints are local, limited to one structure at a time. HPSG puts emphasis on explicitness and precision, its linguistic analyses are couched in a mathematical formalism with well-defined syntax and model-theoretic semantics. Because of this explicitness and formality, HPSG has become one of the most popular linguistic formalisms in computational linguistic applications and this is one of the most important reasons why we have chosen it in our work.

HPSG is a linguistic formalism, i.e., a set of formal tools for formalising linguistic analyses of various phenomena, but it is also a linguistic theory, i.e., a collection of analyses of various phenomena described using this formalism. In this work we accept the main ideas of the formalism but at the same time we

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 297–320, 2005.

introduce some changes in the theory itself and in ways of representing particular aspects of the linguistic constructs.

HPSG grammars consist of a signature and a theory proper. The theory is a set of constraints that all objects in the model must simultaneously satisfy. The signature defines what types of objects there are (e.g., verbs, nouns, cases, genders) and what features they may have (e.g., verbs have person but not case, nouns have case, genders are atomic objects, i.e., do not have any features). In particular all linguistic expressions are represented by objects of the type *sign* having two subtypes: *phrase* and *word*.

The next part of any HPSG theory is a set of constraints. The most famous HPSG constraint is the Head Feature Principle, a version of which is given in (1).

(1) $$\textit{phrase} \rightarrow \begin{bmatrix} \textsc{synsem|local|cat|head} & \boxed{1} \\ \textsc{head-dtr|synsem|local|cat|head} & \boxed{1} \end{bmatrix}$$

Head Feature Principle is an implicational constraint: every object that is characterised by the left-hand side of '→' must also be characterised by the right-hand side. In this particular case, every object of type *phrase* must be such that the value of its SYNSEM|LOCAL|CAT|HEAD attribute is also the value of the SYNSEM|LOCAL|CAT|HEAD attribute of its head daughter. The tag '1' is just a variable used for indicating equality between paths.

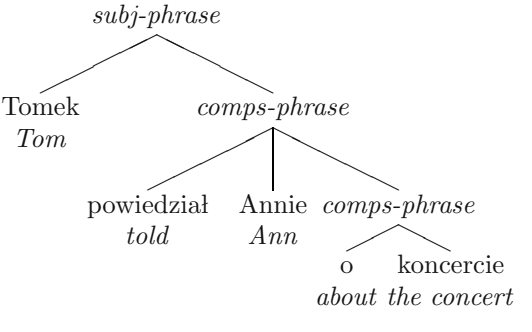
We will not introduce here the HPSG theory itself, the reader is referred to [PS94]. We will focus on presenting new elements of the theory and their interpretation.

2 Modifications of the Standard Theory

2.1 "Flat" Phrase Structure

According to the generally accepted assumption, in HPSG (and in other generative formalisms) the head element (e.g., verb *powiedział 'told'*) takes first its complements (e.g., noun *Annie 'Ann'* and prepositional phrase *o koncercie 'about the concert'*) forming almost saturated phrase, e.g., verb phrase *powiedział Annie o koncercie 'told Ann about the concert'*. Then this phrase takes the subject and forms a saturated phrase, i.e., a structure with empty valence lists SUBJ and COMPS, e.g., a clause *Tomek powiedział Annie o koncercie* (see (2) below).

(2)

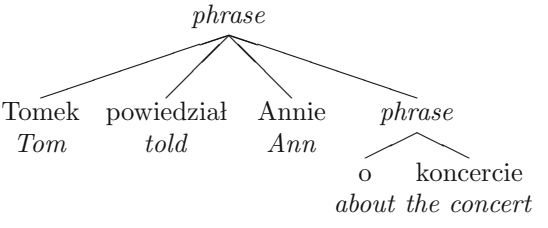


Arguments for two stages' phrase construction, that is separate realisation of subject and complements, are not sufficiently convincing for Polish. There is no place here for a detailed discussion of the subject (such a discussion can be found in [PKMM01]), so we present only examples (3) showing that in Polish there are no order rules supporting the distinction between the subject and complements. Sentences in which the subject is realised 'closer' to a verb than its complement are quite frequent in Polish.

(3) a. Pwiedział Tomek Annie o koncercie.
b. O koncercie powiedział Annie Tomek.
c. Annie Tomek powiedział o koncercie.

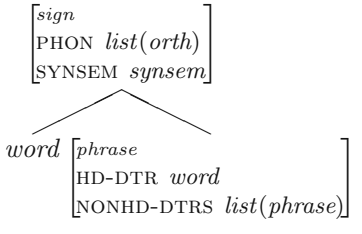
Consequently, we assume that all arguments of a head are syntactically realized at the same level of phrase structure. As a result, phrase structures are flat, as in (4) below, and there is no need for the distinction between types *subj-phrase* and *comps-phrase*.

(4)



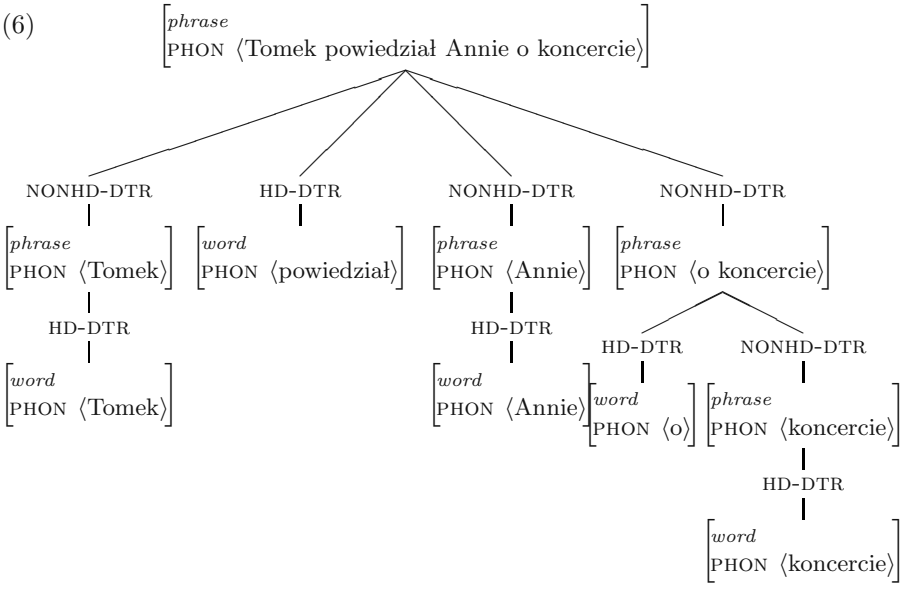
The next assumption made by us about the construction of phrases is the constraint imposed on the types of phrase elements. It states that the head element of the phrase should be a *word* (not a *phrase*) while the elements of the NONHD-DTRS list should be *phrase*s (not *word*s). This assumption is formalised in the hierarchy of the *sign* type below, (5).

(5)



The above assumption allows for eliminating redundant parses which differ only in treating a particular input element as a word or as a (one word) phrase. At the same time, it imposes the simultaneous realisation of all the arguments. For example, the only parse tree for the sentence *Tomek powiedział Annie o koncercie* will be the that given in (6).

(6)



The changes just introduced allow for the simpler formulation of the Valence Principle. As all head's arguments have to be satisfied simultaneously, the value of the phrases' VALENCE attribute has to be an empty list. Thus, it turns out that for phrases this attribute is not necessary at all. So, we introduce it only for objects of the type *word* (to do so, we must change the place of introducing this attribute to the highest level within the *word* structure). Since we proposed not to distinguish subjects and complements at the phrase structure level, the division of a VALENCE list into SUBJ and COMPS attributes is no longer necessary – the VALENCE attribute simply has a list of *synsem*s as its value:

(7) $\begin{bmatrix} word \\ \text{VALENCE } list(synsem) \end{bmatrix}$

The above changes make it possible to express the Valence Principle in the following way:

(8) **Valence Principle**

$$phrase \rightarrow \begin{bmatrix} \text{HD-DTR}|\text{VAL } \boxed{1} \\ \text{NONHD-DTRS } \boxed{2} \end{bmatrix} \wedge \texttt{synsems-signs(}\boxed{1}\texttt{,}\boxed{2}\texttt{).}$$

(9) `synsems-signs(<>,<>).`

$$\texttt{synsems-signs(<}\boxed{1}\texttt{|}\boxed{2}\texttt{>,<}\boxed{1'}\texttt{|}\boxed{2'}\texttt{>)} \quad \overset{\forall}{\Longleftarrow}$$

$$\boxed{1'} = \begin{bmatrix} sign \\ \text{SYNSEM } \boxed{1} \end{bmatrix}$$

$$\wedge \texttt{ synsems-signs(}\boxed{2}\texttt{,}\boxed{2'}\texttt{).}$$

In particular, the Valence Principle requires that the head element has the VALENCE attribute, so it automatically excludes phrases as head elements.

2.2 The Correspondence Between ARG-ST and VALENCE

In this subsection we will describe the relation between VALENCE and ARG-ST attributes. First, although we think that the distinction between subjects and complements plays no role in describing the structure of sentences, it is important for describing some language phenomena, such as agreement, case assignment or binding theory. Since these phenomena are accounted for with the help of the ARG-ST attribute, we re-introduce the subject/complements distinction at the level of ARG-ST list and posit that values of this argument be structures of the following *arg-st* type:

(10)
$$\begin{bmatrix} \textit{arg-st} \\ \text{SUBJ } \textit{list}(\textit{synsem}) \\ \text{ARGS } \textit{list}(\textit{synsem}) \end{bmatrix}$$

The next change concerning argument structure description concerns defining the ARG-ST attribute within the *head* structure. Consequently, the ARG-ST attribute is now defined not only for *words* but also for *phrases* (a detailed discussion concerning this problem may be found in [Prz01]).

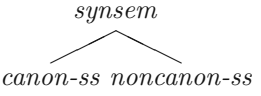
(11)
$$\begin{bmatrix} \textit{category} \\ \text{HEAD} \begin{bmatrix} \textit{head} \\ \text{ARG-ST} \begin{bmatrix} \textit{arg-st} \\ \text{SUBJ } \textit{list}(\textit{synsem}) \\ \text{ARGS } \textit{list}(\textit{synsem}) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Attributes ARG-ST (SUBJ and ARGS) and VALENCE include similar but not necessarily the same elements. On the VALENCE list we put those elements from the ARG-ST|SUBJ and ARG-ST|COMPS lists which are realised in the local syntactic tree, while on the ARG-ST lists all predicate arguments, even those not syntactically realised, are present. These non realised arguments can be of three following kinds:

- dummy subject of personal verb forms (***pro***),
- subject of non personal verb forms (e.g., inifinitive *ogolić się* in (12) or participle *myśląc* w (13)).
  (12) Kazał Tomkowi się ogolić. He told Tom to shave himself.
  (13) Jadła myśląc o swojej przyszłości. She ate thinking about her future.
- arguments of verbs located 'lower' in the syntactic tree realised 'higher' in the syntactic structure, e.g., a complement of the verb *zaprosić* in (14) realised 'higher' as a interrogative pronoun or a subject realised in (15) as a relative pronoun *który*. Such non-locally realised arguments are called *gaps*.
  (14) Kogo chciałeś, żebym zaprosił __?
  Who you wanted that me invited __
  'Who did you want me to invite?'
  (15) ... facet, który chciałam, żeby __ przyszedł.
  ... guy who I wanted that __ came
  '... a guy whom I wanted to come'

To allow for nonlocal arguments we introduce (after [MS97], [Sag97] and [BMS01]) two subtypes of the *synsem* type: *canonical-synsem* (*canon-ss*), representing arguments which are locally realised, and *noncanonical-synsem* (*noncanon-ss*), representing non-realised arguments.

(16)



We require that values of the SYNSEM attribute be of the *canon-synsem* type, see (17). This will cause all VALENCE elements to have their SYNSEM values of the type *canon-synsem*, so objects of the *noncanon-synsem* can appear only on the ARG-ST lists.

(17)
$$\begin{bmatrix} sign \\ \text{PHON } list(orth) \\ \text{SYNSEM } canon\text{-}ss \end{bmatrix}$$

A revised version of Argument Structure Principle is given below[1]:

(19) **Argument Structure Principle**

$$word \rightarrow \begin{bmatrix} \text{SYNSEM|LOCAL|CAT|HEAD|ARG-ST} & \begin{bmatrix} arg\text{-}st \\ \text{SUBJ } \boxed{1} \\ \text{ARGS } \boxed{2} \end{bmatrix} \\ \text{VALENCE } \boxed{3} \end{bmatrix}$$
$$\wedge\ \boxed{1} \oplus \boxed{2} = \boxed{3} \bigcirc list(noncanon\text{-}ss).$$

In the following section we will introduce the notion of raising elements. The distinction between raised and non-raised elements make the actual *synsem* hierarchy a little more complicated (see (20)), but the formulation of the Argument Structure Principle remains unchanged.

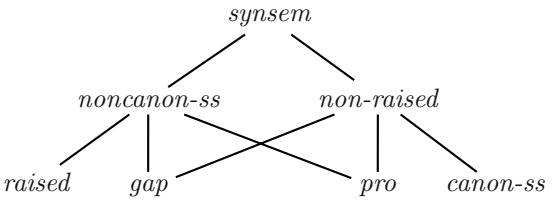
2.3 Noncanonical Arguments

In this section, we will present non-canonical arguments in more detail and we will introduce further refinements of the *synsem* hierarchy, given in (20). We will not present here the detailed discussion on their distribution, which can be found in [PKMM01].

[1] "$\oplus$" is an abbreviation for the `append` relation; "$\bigcirc$" is an infix notation of the shuffle relation [Rea92], i.e., `shuffle(`$\boxed{1}$`,`$\boxed{2}$`,`$\boxed{3}$`)` $\equiv \boxed{1} \bigcirc \boxed{2} = \boxed{3}$. The `shuffle` relation is defined as follows:

(18)
`shuffle(<>,<>,<>).`
`shuffle(<`$\boxed{1}$`|`$\boxed{2}$`>,`$\boxed{3}$`,<`$\boxed{1}$`|`$\boxed{4}$`>)` $\overset{\forall}{\Longleftarrow}$ `shuffle(`$\boxed{2}$`,`$\boxed{3}$`,`$\boxed{4}$`).`
`shuffle(`$\boxed{2}$`,<`$\boxed{1}$`|`$\boxed{3}$`>,<`$\boxed{1}$`|`$\boxed{4}$`>)` $\overset{\forall}{\Longleftarrow}$ `shuffle(`$\boxed{2}$`,`$\boxed{3}$`,`$\boxed{4}$`).`

(20)



Raised Arguments. We assume that some arguments may be 'raised' higher in the syntactic hierarchy instead of being realised locally; i.e. they are realised as syntactic arguments of the higher verb. An example of such a construction is an infinitival phrase. In the sentence below, arguments of the infinitive verb *dać 'give'* can be realised locally or they can be raised and realised as arguments of the verb *chciał 'wanted'*.

(21) Janek chciał dać Marysi kwiaty.
'John wanted to give Mary flowers.'

Partial motivation for introducing argument raising comes form the phenomenon of Genitive of Negation. If the higher verb (i.e., *chciał*) is negated, the argument *kwiaty* should occur in genitive, not in accusative; compare (22) with (21).

(22) Janek nie chciał dać Marysi kwiatów.

In order to maintain local case assignment principles, we are forced to assume that, in (21), *kwiaty* is in some sense the argument of the verb *chiał*. On the other hand, Genitive of Negation is to some extent optional. In some cases, such arguments may stay in the accusative case (cf. [Prz99,Prz00]). Consequently, we assume that argument raising is optional, i.e., examples like (21) have several parses differing in the placement of infinitival's arguments.

Lexical entries do not specify which arguments are of the *canon-ss* type, i.e., which arguments are realised locally. However, they have to represent the fact that those arguments which are not realised locally have to be raised to a higher level. To represent objects which can be raised, we introduce the *raised – non-raised* distinction within the *synsem* type. All *raised* objects are of *noncanon-ss* subtype while *non-raised* arguments are divided into *canon-ss*, *pro* and *gap* subtypes, (20).

The adequate lexical entry for the verb *chciał* is given below. The `raise-local` function is responsible for raising LOCAL structures only. Subjects of infinitives are never locally realised, so their SUBJ value is of the *noncanon-ss* type.

(23)

$$\left[\begin{array}{l} word \\ \text{PHON } \langle \text{chciał} \rangle \\ \text{SS|L|C|H|ARG-ST } \left\langle \left[\begin{array}{l} arg\text{-}st \\ \text{SUBJ } \left\langle \left[\begin{array}{l} synsem \\ \text{L|C|H } \left[\begin{array}{l} noun \\ \text{CASE } nom \end{array}\right] \end{array}\right] \right\rangle \end{array}\right], \left[\begin{array}{l} synsem \\ \text{L|C|H } \left[\begin{array}{l} inf \\ \text{ARG-ST } \left[\begin{array}{l} arg\text{-}st \\ \text{SUBJ } \langle noncanon\text{-}ss \rangle \\ \text{ARGS } list(canon\text{-}ss) \bigcirc \boxed{2} list(raised) \end{array}\right] \end{array}\right] \end{array}\right] \right\rangle \oplus \boxed{2'} \end{array}\right]$$

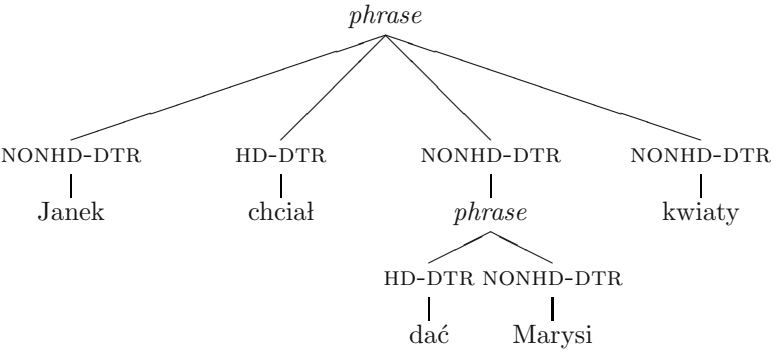
$\wedge$ `raise-local(`$\boxed{2}$`,`$\boxed{2'}$`).`

(24) `raise-local(<>,<>).`

`raise-local(<`$\left[\begin{array}{l} synsem \\ \text{LOCAL } \boxed{1} \end{array}\right]$`|`$\boxed{2}$`>,<`$\left[\begin{array}{l} synsem \\ \text{LOCAL } \boxed{1} \end{array}\right]$`|`$\boxed{3}$`>)` $\overset{\forall}{\Longleftarrow}$ `raise-local(`$\boxed{2}$`,`$\boxed{3}$`).`

If we assume that the dative complement of *dać* is realised locally, while the accusative complement is not, we will achieve the structure given in (25).

(25)



Pro. The other kind of non-canonical argument is a dummy subject, *pro*. In (26), the subject is not realised on the surface, so it can be present only in the ARG-ST|SUBJ list, not in the VALENCE list. Dummy subjects are represented by a special subtype of the *noncanon-ss* type – *pro*[2].

(26) a. Dał Marysi kwiaty.

b. $\left[\begin{array}{l} word \\ \text{PHON } \langle \text{dał} \rangle \\ \text{SYNSEM|LOCAL|CAT|HEAD|ARG-ST } \left[\begin{array}{l} arg\text{-}st \\ \text{SUBJ } \langle \boxed{1} pro \rangle \\ \text{ARGS } \langle \boxed{2}\text{NP}[dat], \boxed{3}\text{NP}[acc] \rangle \end{array}\right] \\ \text{VALENCE } \langle \boxed{2}, \boxed{3} \rangle \end{array}\right]$

[2] We give up here the traditional (in the generative literature) distinction between PRO and *pro* and represent both kinds of empty elements as *pro*.

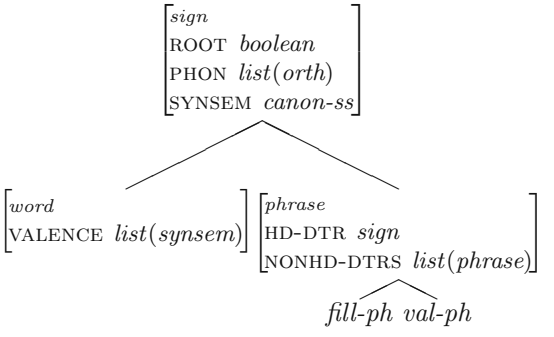
It is not always true, that a non realised subject is represented by the *pro* element. There exist arguments that infinitival subjects in sentences like (27) should not be represented as *pro* but should be treated as being raised. Such an analysis reflects number and gender agreement between *Janek* – the subject of the higher verb and the adjective *miły*.

(27) Janek chciał __ być miły.
John wanted to be nice.

2.4 Types of Phrases

We assume two phrase types: *valence-phrase* and *fill-phrase* (named *val-ph* nad *fill-ph* respectively). The complete hierarchy of *sign* is presented below:

(28)



Valence-Phrase. The basic phrase schema described in 2.1 represents phrases which consist of a head element (a word) and its dependents. This schema can be used to build not only clauses but also phrases with non-verbal head elements, e.g., nominal or prepositional phrases. It is also used for representing phrases with markers (complementizers), which are analysed as heads. To allow for this, we introduce a *marker* subtype of type *head*.

We call all phrases constructed according to this basic schema *valence-phrase* (*val-ph*) and limit the scope of the Valence Principle to this type of phrases only:

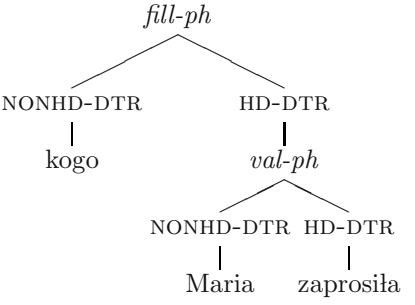
(29) **Valence Principle**

$$\textit{valence-phrase} \rightarrow \begin{bmatrix} \text{HD-DTR}|\text{VAL } \boxed{1} \\ \text{NONHD-DTRS } \boxed{2} \end{bmatrix} \wedge \texttt{synsems-signs}(\boxed{1},\boxed{2}).$$

A phrase of type *val-ph* is always most deeply nested, because its head element should have the VALENCE attribute, so it should be a word. For example, in the phrase in (30), a *val-ph* phrase has to occur inside a *fill-ph* phrase.

(30) a. Mówiłeś, że [kogo Maria zaprosiła]?

b.



Filler-Phrase. Traditionally, *filler-phrases* are used to represent structures within which some locally non-realised elements are finally realised. For example in the phrases below, the elements *kto*, *co*, *komu*, *którego*, *komu* are realised nonlocally.

(31) [Kto co komu] [dał]?
(32) ... facet, [któremu] [chciałaś, żebym dał tę książkę].

The analysis of non-local dependencies follows [BMS01] and it is based on the idea of passing information about non-realised arguments via the NONLOCAL structure. Non-realised elements are locally represented by the special object of the type *gap*, which is the last subtype of *noncanon-ss* that we define here. This type introduces a nonempty value of the SLASH attribute within the NONLOCAL structure, (33).

(33) $$gap \rightarrow \begin{bmatrix} \text{LOCAL } \boxed{1} \\ \text{NONLOC|SLASH } \langle \boxed{1} \rangle \end{bmatrix}$$

For example in the case of *dał*, (31), the NONLOCAL|SLASH value is as follows:

(34) $$dał\text{:} \begin{bmatrix} nonlocal \\ \text{SLASH } \langle \begin{bmatrix} local \\ \text{C|H } \begin{bmatrix} noun \\ \text{CASE } nom \end{bmatrix} \end{bmatrix}, \begin{bmatrix} local \\ \text{C|H } \begin{bmatrix} noun \\ \text{CASE } acc \end{bmatrix} \end{bmatrix}, \begin{bmatrix} local \\ \text{C|H } \begin{bmatrix} noun \\ \text{CASE } dat \end{bmatrix} \end{bmatrix} \rangle \end{bmatrix}$$

We impose the following constraint on the *fill-ph* type[3]:

(35) $$fill\text{-}ph \rightarrow \begin{bmatrix} \text{SYNSEM|NONLOC|SLASH } \langle\rangle \\ \text{HD-DTR } \begin{bmatrix} val\text{-}ph \\ \text{SS|NONLOC|SLASH } \boxed{1} nelist \end{bmatrix} \\ \text{NONHD-DTRS } \boxed{1'} \end{bmatrix} \wedge \texttt{locals-signs(}\boxed{1}\texttt{,}\boxed{1'}\texttt{).}$$

$$\texttt{locals-signs(<>,<>).}$$

$$\texttt{locals-signs(<}\boxed{1}\texttt{|}\boxed{2}\texttt{>,<}\boxed{1'}\texttt{|}\boxed{2'}\texttt{>)} \overset{\forall}{\Longleftarrow}$$

$$\boxed{1'} = \begin{bmatrix} sign \\ \text{SYNSEM|LOCAL } \boxed{1} \end{bmatrix}$$

$$\wedge\ \texttt{locals-signs(}\boxed{2}\texttt{,}\boxed{2'}\texttt{).}$$

[3] We require the SLASH value to be nonempty in order to exclude trivial *fill-ph* phrases with no non-locally realised elements.

2.5 Lexicon

Although it is possible to construct a more sophisticated lexicon structure taking advantage of HPSG type hierarchies, we adopt the simplest solution and define the lexicon as a set of lexical entries. A slight modification of the standard approach consists in introducing a difference between lexical entries (of type *entry*) and syntactic words (of type *word*). The Lexicon Principle is thus formulated as follows:

(36) **Lexicon Principle**

$$entry \rightarrow HS_1 \vee HS_2 \vee \ldots \vee HS_n$$

Objects of type *entry* introduce attributes PHON (with values of type *list*(*orth*)), CONT (with values of type *content*) and HEAD (with values of type *head*).

(37)
$$\begin{bmatrix} entry \\ \text{PHON } list(orth) \\ \text{HEAD } head \\ \text{CONT } content \end{bmatrix}$$

In the simplest cases, objects of type *word* may take their attribute values directly from the ENTRY structure which will be now a part of the *word* structure.

2.6 Modifiers

In HPSG, modifiers (adjuncts) are normally represented via the attribute MOD of type *head*, whose value is a (at most one element) list of objects of type *synsem*. We divide this *synsem* information into syntactic and semantic parts. Thus, the attribute MOD has values of type *mod*, which has two attributes: SYN of type *head* and SEM of type *content*:

(38)
$$\begin{bmatrix} mod \\ \text{SYN } head \\ \text{SEM } content \end{bmatrix}$$

(39)
$$\begin{bmatrix} head \\ \text{MOD } list(mod) \\ \text{ARG-ST } \begin{bmatrix} arg\text{-}st \\ \text{SUBJ } list(synsem) \\ \text{ARGS } list(synsem) \end{bmatrix} \end{bmatrix}$$

Since, in Polish, we do not observe any clear syntactic differences between complements and modifiers, we adopt here the solution known in HPSG as "adjuncts-as-complements" (see [BMS01], [Prz99]). The idea consists in placing modifiers together with arguments on the ARG-ST|ARGS list.

Taking the description of a word from a lexicon, beside taking the appropriate word's arguments form the ENTRY structure, we add to the ARG-ST|ARGS list a list of (4 in (41)). Moreover, according to the modifiers type the value of the attribute CONT can also be changed., (see "f(3,4)" in (41)).

Since the only part of the HEAD value of the ENTRY structure which can be changed is inside the ARG-ST attribute, we divide HEAD structure into ARG-ST attribute and MORSYN attribute containing all remaining head attributes:

(40)
$$\begin{bmatrix} head \\ \text{ARG-ST}\ \ arg\text{-}st \\ \text{MORSYN}\ \begin{bmatrix} morsyn \\ \text{MOD}\ \ list(mod) \end{bmatrix} \end{bmatrix}$$

Applying all the modifications just introduced, we can formulate the constraint describing the relation between ENTRY and SYNSEM|LOCAL structures in the following way:

(41)
$$word \rightarrow \begin{bmatrix} \text{PHON}\ \boxed{1} \\ \text{SS|LOC}\ \begin{bmatrix} \text{CAT|HEAD}\ \begin{bmatrix} head \\ \text{MORSYN}\ \boxed{6} \\ \text{ARG-ST}\ \begin{bmatrix} arg\text{-}st \\ \text{SUBJ}\ \boxed{5} \\ \text{ARGS}\ \boxed{2} \oplus \boxed{4} list\left(\left[\text{MOD}\ \left\langle \left[\text{SYN}\ \boxed{6}\right]\right\rangle\right]\right) \end{bmatrix} \end{bmatrix} \\ \text{CONT}\ f(\boxed{3},\boxed{4}) \end{bmatrix} \\ \text{ENTRY}\ \begin{bmatrix} entry \\ \text{PHON}\ \boxed{1} \\ \text{HEAD}\ \begin{bmatrix} head \\ \text{MORSYN}\ \boxed{6} \\ \text{ARG-ST}\ \begin{bmatrix} \text{SUBJ}\ \boxed{5} \\ \text{ARGS}\ \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{CONT}\ \boxed{3} \end{bmatrix} \end{bmatrix}$$

3 Selected Phenomena of Polish

3.1 Agreement

One of the main grammatical issue in Polish is agreement[4]. We concentrate on two main types of agreement: adjective–noun agreement and subject–verb agreement.

The adjective must agree with the noun in number, gender, and case, see (42). The same type of agreement takes place between the possessive pronoun and the noun (43), as well as between the numeral and the noun (44).

(42) a. pięknej dziewczynie
pretty$_{sg,fem,gen}$ girl$_{sg,fem,gen}$
b. *pięknemu dziewczynie
pretty$_{sg,masc,gen}$ girl$_{sg,fem,gen}$

(43) moją matką
my$_{sg,fem,inst}$ mother$_{sg,fem,inst}$

[4] The problem of agreement for Polish is discussed within the HPSG setup also in [Czu95] and [CP95].

(44) dwóch dziewczynach
two$_{fem,loc}$ girls$_{pl,fem,loc}$

The nominative subject agrees with the verb in person, number and gender, (45)

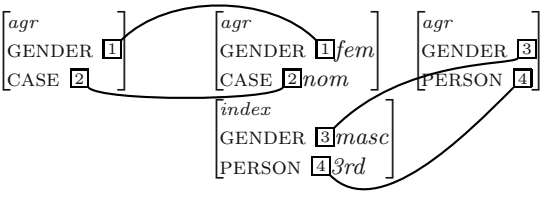
(45) a. Matka przyszła.
mother$_{sg,fem,nom}$ came$_{3rd,sg,fem}$
b. *Matka przyszedł.
mother$_{sg,fem,nom}$ came$_{3rd,sg,masc}$

The above examples show the common type of agreement in Polish. There are also untypical agreement where, e.g., semantic gender is not the same as the syntactic gender, and the verb agrees with the semantic gender of the noun, see (46).

(46) Jego wspaniałomyślna wysokość przyszedł.
His$_{fem,nom}$ magnanimous$_{sg,fem,nom}$ highness$_{sg,fem,nom}$ came$_{3rd,sg,masc}$
'His magnanimous highness came.'

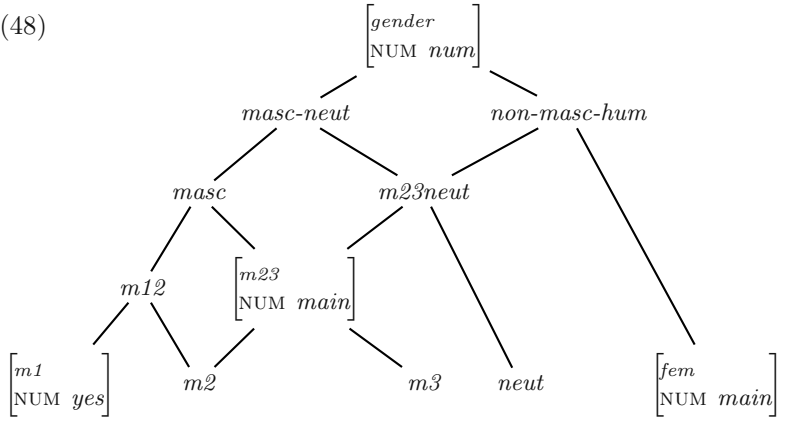
To cope with the problem of different syntactic and semantic gender of such nouns, the *index* structure (semantic gender, number and person) of the subject agrees with the syntactic gender, number and person (structure *agr*) of the verb, while the NP-internal agreement uses only syntactic attributes:

(47) Jego wspaniałomyślna wysokość przyszedł.



(48) presents the hierarchy of Polish gender elaborated to account for different types of agreement adjective–noun and numeral–noun agreement.

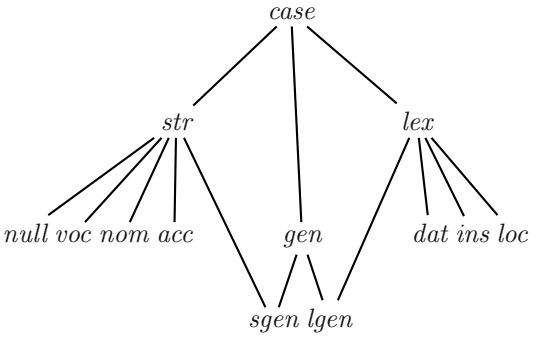
(48)



3.2 Case Assignment

The hierarchy of Polish cases[5] is given in (49). Cases are divided into structural and lexical types. The value of structural case is established not only by the subcategorisation rules but also by the environment, lexical cases are determined independently.

(49)



The most famous case phenomena in Polish is the Genitive of Negation (GoN), i.e., the shift of a direct object's case from accusative in a non-negated clause to genitive in the negated clause, see (50).

(50) a. Piszę listy.
write$_{1st,sg}$ letters$_{acc}$
'I am writing letters'
b. Nie piszę listów / *listy
NM write$_{1st,sg}$ letters$_{gen}$ / letters$_{acc}$
'I am not writing letters'

This phenomenon is nonlocal: in the case of the long distance Genitive of Negation, an argument of a *lower* verb may occur in the genitive when a *higher* verb is negated, see (51).

(51) Nie chciałem pisać listów / *listy.
NM want$_{1st,sg}$ write$_{inf}$ letters$_{gen}$ / letters$_{acc}$
'I didn't want write letters'

Interesting case assignment phenomena also include complex case patterns in numeral phrases, see (52)

(52) a. Pięć kobiet przyszło.
five$_{nom?/acc?}$ women$_{gen,pl}$ came$_{3rd,sg,neut}$
'Five women came.'
b. Rozmawiam z pięcioma kobietami / *kobiet.
talk$_{1st,sg}$ with five$_{ins}$ women$_{ins/*gen}$
'I am talking with five women.'

[5] The problem of case assignment is widely discussed in [Prz99].

Case patterns in predicative constructions in Polish are also interesting. In simple cases, predicative adjectives agree with predicated elements, see (53). But the predicative adjective can sometimes occur in the instrumental case, see (54).

(53) On jest miły.
he$_{nom}$ is nice$_{nom}$
'He is nice.'

(54) Pamiętam go miłego / miłym.
remember$_{1st,sg}$ him$_{acc}$ nice$_{acc}$ / nice$_{ins}$
'I remember him as nice.'

3.3 Binding Theory

The next problem addressed in the grammar is the binding theory for Polish[6]. It is not the whole theory but only Principles A and B formulated for pronominals and reflexive anaphors (possessive and non-possessive).

Anaphor binding in Polish can be roughly characterised as subject oriented and clause-bound. The distribution of personal pronouns in these sentences is complementary to that of anaphors, i.e., pronouns have to be disjoint with the subject, while coindexation with another non-subject argument of a verb or a clause external NP is correct, see (55)

(55) a. Jan$_i$ opowiadał Piotrowi$_j$ o sobie$_{i/*j}$/nim$_{*i/j}$.
John told Peter about self/him
'John told Peter about himself/him.'

b. Jan$_i$ powiedział, żeby Piotr$_j$ opowiedział o sobie$_{*i/j}$/nim$_{*i/j}$.
John told COMP Peter told about self/him
'John told Peter to tell about himself/him.'

The theory accounts for such important phenomena as medium distance binding in the case of control verbs, see (56). The possessive anaphor *swoje* has two possible antecedents: the sentential subject, *Jan*, or clause-internal one, *Piotrowi*. On the other hand, the possessive pronoun *jego* may not be bound by any of these elements.

(56) Jan$_i$ kazał Piotrowi$_j$ przynieść swoje$_{i/j}$/ jego$_{*i/*j}$ dokumenty.
John ordered Peter bring$_{inf}$ self's his documents
'John ordered Peter to bring his documents .'

We also analyse binding within noun phrases that can have subject (57) and attributive adjective phrases (58).

(57) wiara Marii$_i$ w siebie$_i$/ nią$_{*i}$
faith Mary's in self her
'Mary's faith in her (ability)'

[6] An HPSG theory of binding in Polish is presented in [Mar99,Mar01].

(58) Jan$_i$ zatelefonował do Piotra$_j$ napadniętego w swoim$_{*i/j}$/ jego$_{i/*j}$ domu.
John phoned to Peter robbed in self's his house
'John phoned Peter robbed in his house.'

A virtual subject is necessary to interpret the differences of binding in impersonal constructions, see (59).

(59) a. Kupiono sobie/ im lekarstwa.
bought self/ them medicines
'They bought medicines for themselves/ them.'

b.
$$\begin{bmatrix} word \\ \text{PHON } \langle \text{kupiono} \rangle \\ \text{SS|LOC} \begin{bmatrix} local \\ \text{CAT|HEAD|ARG-ST} \begin{bmatrix} arg\text{-}st \\ \text{SUBJ } \langle pro_j \rangle \\ \text{ARGS } \langle \text{NP}_k, \text{ NP: } pron_i \rangle \end{bmatrix} \\ \text{CONT} \begin{bmatrix} buying \\ \text{BUYER } j \\ \text{BOUGHT } k \\ \text{BENEFICIARY } i \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Theory of binding is defined on the ARG-ST structure. The most important relation, corresponding to local o-command relation (for English) [PS94, ch.6], is the relation of *local subject-command*, henceforth *local s-command* (see definition (61)). To formulate this relation, it is convenient to introduce a class of transparent phrases whose boundaries can be crossed in the process of binding.

(60) A *synsem* object X is *transparent* if X is a PP, VP[inf] or an NP without subject.

The definition of local s-command is given in (61):

(61) Let Y and Z be *synsem* objects. Then Y *locally s-commands* Z in case either:
i. exists a *arg-st* structure for which Y belongs to its SUBJ, and Z belongs to the list of its ARGS; or
ii. Y locally s-commands a transparent X and Z belongs to the ARG-ST structure of X

The local o-binding and local o-freeness relations (for English) are substituted by local s-binding and local s-freeness, respectively, see definition (62).

(62) Y *locally s-binds* Z just in case Y and Z are coindexed and Y locally s-commands Z. If Z is not locally s-bound, then it is said to be *locally s-free*.

The Principles A and B for Polish are formulated as in (63)

(63) Principle A. A reflexive anaphor must be locally s-bound.
Principle B. A pronoun must be locally s-free with the exception of possessive pronouns in first and second person and when possessive pronoun is bound by explicit subject of NP.

3.4 Relative Clauses

Polish relative clauses can be illustrated by the examples given in (64).

(64)
a. ten$_{nom,sg,masc}$, komu$_{dat,sg,masc}$ zazdrościcie$_{pl}$ _$_{dat}$
one whom you$_{pl}$ envy
'someone you envy'

b. chłopak$_{nom,sg,masc}$, [którego$_{acc,sg,masc}$ siostrze]$_{dat,sg,fem}$ zazdrościcie _$_{dat}$
a boy whose sister you$_{pl}$ envy _
'a boy whose sister I envy'

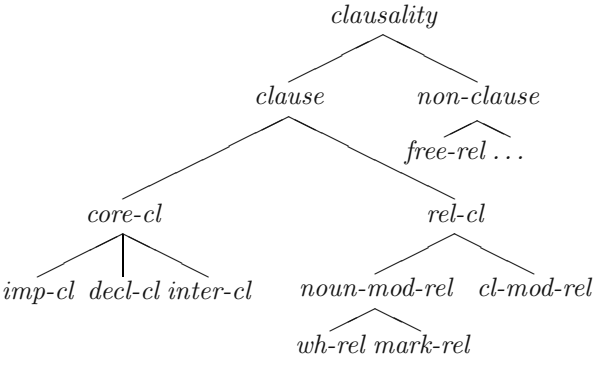
c. pióro$_{sg,neut}$, co nim$_{instr,sg,neut}$ /*() pisałam _$_{instr}$
a pen that with what /() I wrote
'a pen I wrote with'

d. Anna tańczyła, czemu Paweł przyglądał się uważnie.
Ann danced what$_{dat}$ Paul looked at carefully
'Paul looked carefully at Ann dancing'

e. ten/() kto sieje wiatr, zbiera burzę
this/() who$_{nom}$ sows a wind he picks a storm
'he who sows the wind shall reap the whirlwind'

The analysis of relative clauses presented here is based on the approach of [Sag97], which relies on multiple inheritance of constraints imposed on elements of phrase types hierarchy. However, we assume here only the phrase type hierarchy given in (28) while the clausality hierarchy is replaced by the CLAUSALITY attribute introduced for *phrases*. The possible values of this attribute are the subtypes of type *clausality* and are given in (65).

(65)



Clauses and non-clauses are distinguished on the basis of the type of the head element – for type *clause*, the value of the MORSYN attribute should be *personal, -no/-to, się, infinitival* or *marker*. Relative clauses are divided into *free-relatives*, which are a subtype of non-clauses, and relative clauses (proper), which are a subtype of clauses. The *clausality* hierarchy distinguishes relative clauses from core clauses on the basis of the *mod* attribute which is not empty for *rel-cl*. Relative clauses are then divided into those modifying noun phrases and those modifying clauses. Finally, noun modifying relatives are divided into

those starting with relative pronouns, those starting with the relative marker and the reduced relatives.

To account for non-local dependencies, we accept and extend the idea presented in [BMS01]. Information which is to be used non-locally is, as usual, grouped within the SYNSEM|NONLOCAL structure, (66). Here, apart from the SLASH and the REL attributes we define the RES attribute whose value is introduced by resumptive pronouns[7].

$$(66)\quad \begin{bmatrix} \textit{synsem} & \\ \text{LOCAL } \textit{local} & \\ \text{NONLOCAL} & \begin{bmatrix} \text{SLASH } \textit{list}(\textit{local}) \\ \text{REL } \boxed{1}\ \textit{list}(\textit{index}) \\ \text{RES } \boxed{2}\ \textit{list}(\textit{index}) \end{bmatrix} \end{bmatrix} \wedge \texttt{max-one}(\boxed{1}, \boxed{2}).^{8}$$

Phrases of the type *val-ph* inherit their NONLOCAL value from their head-daughters, while *fill-ph* phrases are places for binding nonlocal dependencies.

For a word, its SLASH value is obtained by gathering the SLASH values from all its dependents (if it has any), while the RES and REL values, apart from being gathered from all word's dependents, can also be specified within a lexicon description (inside the ENTRY structure). The way of computing the appropriate values are encoded in the Nonlocal Lexical Amalgamation Principle.

Polish relative pronouns can be divided into nominal relative pronouns: *który 'who/what', jaki 'which', kto 'who'* and *co 'what'* and adverbial relative pronouns, e.g., *gdzie 'when', kiedy 'where', skąd 'where from'*. In general, the use of the Polish relative pronouns is quite similar to other languages (e.g., Bulgarian, English). However, there are some specific features which have to be noted. One such idiosyncrasy is the use of different nominal relative pronouns (*który* vs. *kto*) in relation to different types of nominal phrases. Clauses beginning with *który* can modify noun phrases headed by common nouns, proper nouns, personal and demonstrative pronouns, (67a), while the relative pronouns *kto 'who'* and *co 'what'* can modify indefinite and negative pronouns, (67b).

(67) a. pies / Jan / on / tamten który / *kto biegnie
a dog / John / he / that KTóRY / *who runs

b. coś/nic czemu/*któremu się przyglądasz
something/nothing what/*KTóRY self you look at
something/nothing you look at

To give a complete analysis of Polish relative clauses beginning with pronouns, one has to deal with the following problems: ensuring the gender and number agreement between the modified noun and the pronoun, assigning the

[7] To make the formalization easier, we use lists instead of sets of values. In case of REL and RES this change is purely theoretical, as for Polish these attributes can have at most one element set (or list) as their value.

[8] Relation `max-one` represents the fact that, in Polish clauses, only one relative word or one resumptive pronoun can occur, so lists REL and RES can have in sum only one element.

correct case value to the nominal pronoun, ensuring that relative pronouns occur in he appropriate context. All these relations are represented by the appropratte constrains on the *wh-rel* type and lexical entries of relative pronouns.

The second type of noun modifying relative clauses are those begining with the relative marker *co*, (64c). In this clauses the modified object is repeated by a resumptive pronoun (unless it fulfils the role of a subject). Resumptive pronouns are all personal pronouns except their nominative and strong forms (if they exist). They have have two alternative lexicon entries – one with the empty RES value and second with one element on the RES list identified with the INDEX value. In subject *co*-relatives there is no resumptive pronoun in a subject position. To account for this fact, we assume that *pro* objects, which represent dummy subjects can be also interpreted as resumptive pronouns.

Since relative clauses beginning with the marker *co* and the relative pronoun *który* modify different noun phrases that these beginning with the relative pronouns *kto* and *co* we divide *index* into two subtypes: *inst-index*, which will be assigned to all nouns which can be modified by *który*, and *noninst-index* which is appropriate for indefinite and negative pronouns and relative pronouns *kto* and *co*.

As we analyse relative caluses as modifiers, we have to accept non-empty MOD value of verbal phrases. However, we would like to exclude sitations in which "an ordinary " clause (e.g., *Jan śpi 'John sleeps'*) is a modifier. Our solutions consists in changing the scope of the Head Feature Principle for the *fill-ph* phrases to all HEAD attributes besides MOD.

All constraints imposed on the types representing Polish relative clauses are given in [PKMM01], some previous work on the subject can be found in [Myk00].

3.5 Negation

There are several problems which are connected with the issue of negation in Polish[9]. Several words called *n-words* can appear only in the sentence where the environment is negated, it means that the verb is negated (68), or adjective has negative meaning (69) or that there is a negative preposition in the sentence (70). The examples of n-words are *nikt* nobody, *nigdy* never, *żaden* (none).

(68) Nikt *(nie) dał Marysi książki.
nobody NM gave Mary book
'Nobody gave Mary a book.'

(69) Westchnął *(nie)zauważalnie dla nikogo.
sight NM-noticably for nobody
'He sight without being noticed by anybody.'

(70) Zaczął bez żadnych wstępów.
started without none introductions
'He started straight away.'

[9] There are following papers connected with this subject: [PK97b,PK97a,PK99].

We can say that n-words in Polish are sensitive for negation. This feature is represented by attribute NEG-SENS which indicates that a word needs negative environment or does not need. In (71) is given the description of the n-word *nikt* (nobody).

(71)
$$\begin{bmatrix} \textit{entry} \\ \text{PHON } \langle \textit{nikt} \rangle \\ \text{HEAD} \begin{bmatrix} \text{MORSYN} \begin{bmatrix} \textit{n-noun} \\ \text{CASE } \textit{nom} \end{bmatrix} \\ \text{ARG-ST} \begin{bmatrix} \text{SUBJ } \langle\rangle \\ \text{COMPS } \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \textit{ppro} \\ \text{INDEX} \begin{bmatrix} \text{PER } \textit{3rd} \\ \text{NUM } \textit{sg} \\ \text{GEND } \textit{m1} \end{bmatrix} \\ \text{RESTR } \{\} \end{bmatrix} \\ \text{NEG-SENS } + \end{bmatrix}$$

To represent negative environment we use the attribute POLARITY. For example in (72) there is represented semantics of the following events: *lubi* (like) and *nie lubi* (does not like).

(72) a. lubi: $\begin{bmatrix} \textit{psoa} \\ \text{NUCL } \textit{like} \\ \text{POLARITY } + \end{bmatrix}$

b. nie lubi: $\begin{bmatrix} \textit{psoa} \\ \text{NUCL } \textit{like} \\ \text{POLARITY } - \end{bmatrix}$

The negative concord principle is defined in the book in order to attain the correspondance between an n-words and its environment.

3.6 Coordination

Coordinated structures are widespread in natural languages but their formal analysis presents many problems. It is possible to coordinate not only phrases of the same categories, but also different categories can be conjoined as well. Although it is often assumed that coordination may apply only to constituents, coordination of non-constituents or partial constituents (phrases which share arguments) is not uncommon in natural languages. In the book[10], we restrict ourselves only to constituent coordination. For this reason, we base our analysis on the HPSG account of constituent coordination presented in [Par92]. The analysis captures coordination of partial (unsaturated) constituents as well, see (73).

(73) Jan przeczytał i zrecenzował artykuł.
John read and reviewed paper

[10] This problem is also discussed in [KMMP00,KMM00].

We deal with coordination of unlike categories but only in the case of (verbal) modifiers, see (74), where the adverb *szybko* is coordinated with the preposition phrase *bez zastanowienia*.

(74) Odpowiadał szybko i bez zastanowienia.
answered-he quickly and without thinking

Following [Par92], we treat conjunction as a functional head of the coordinated phrase and coordinated elements as complements. In the book there are discussed several types of conjunctions: 'monosegmental' conjunctions, e.g., *i* 'and', *lub* 'or', etc, we consider also discontinuous conjunctions, such as *zarówno . . . jak też* 'both . . . and', *nie tylko. . . lecz również* 'not only. . . but also'.

Let us see the lexical entry for *zarówno. . . jak też* 'both. . . and', (75).

(75)
$$\begin{bmatrix} \textit{entry} \\ \text{PHON } \langle \textit{zarówno, jak i} \rangle \\ \text{HEAD}|\text{CONJ } \textit{conj} \\ \text{NEG-SENS } - \end{bmatrix}$$

In the (76) there is the example of anaysies of the phrase *zarówno Ania jak też Adam* 'both Ania and Adam'

(76)
$$\begin{bmatrix} \textit{phrase} \\ \text{PHON } \langle \textit{zarówno, Ania, jak i, Adam} \rangle \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON } \langle \textit{Ania} \rangle \\ \text{SS } \boxed{1}\textit{canon-ss} \end{bmatrix} \quad \begin{bmatrix} \textit{word} \\ \text{PHON } \langle \textit{zarówno, jak i} \rangle \\ \text{ARG-ST}|\text{ARGS } \langle \boxed{1}, \boxed{2} \rangle \end{bmatrix} \quad \begin{bmatrix} \text{PHON } \langle \textit{Adam} \rangle \\ \text{SS } \boxed{2}\textit{canon-ss} \end{bmatrix}$$

The correct order of conjuncts and the conjunction is obtained via a general constrained which adds the first phonological 'segment' of the conjunction to the phonology of the first conjunct while the second phonological 'segment' of the conjunction precedes the second conjunct.

In languages with a rich morphological system agreement patterns in coordinated structures is quite complex. In the book, we present some aspects of agreement in Polish coordinated NPs and what kind of restriction are undertaken in the grammar. We assume that coordinated phrase must agree with verbs in plural number, see (77), so we are not able to analyse correct sentence (78).

(77) Przyszli Jan i Maria.
came$_{pl,3rd}$ John and Mary
'John and Mary came.'

(78) Przyszedł Jan i Maria.
came$_{sg,masc,3rd}$ John and Mary
'John and Mary came.'

There are also defined relations *gender* (79) and *min* which assign respectively the proper gender and person of the verb connected with coordinated phrase.

(79) `gender(`*masc-hum*`,` *gender*`,` *masc-hum*`).`
`gender(`*gender*`,` *masc-hum*`,` *masc-hum*`).`
`gender(`*non-masc-hum*`,` *non-masc-hum*`,` *non-masc-hum*`).`

This relation are necessary to analyse following sentences:

(80) Chłopiec i dziewczynka biegali po parku.
boy_{masc} and girl_{fem} $\text{run}_{masc-hum}$ in park
'A boy and a girl were running in the park.'

(81) Ja i ty przyszliśmy / *przyszliście.
I_{1st} and you_{2nd} came_{1st}-we came_{2nd}-you
'I and you came.'

The approach presented in the book captures only several types of coordinated phrases but it allows us to apply general HPSG grammatical principles both to coordination and to other types of phrases.

4 Conclusion

In this paper, we have presented a range of phenomena typical for Polish and indicated ways of analysing those phenomena within HPSG. We have also presented modifications of the standard [PS87,PS94] HPSG theory useful or necessary to formulate a straightforward account of Polish syntax. The diversity of the phenomena accounted for, involving both textually frequent phenomena (negation, relative clauses, simple case assignment) and textualy untypical phenomena (idiosyncratic patterns of case assignment, special cases of binding and agreement), lead to the conclusion that Head-driven Phrase Structure Grammar is a formalism well-suited for analysing morphologically-rich "free word-order" languages such as Polish.

The account alluded to above, fully described in [PKMM01] and references therein, has been partially implemented in ALE [CP01]. The implementation varies from the theoretical analysis in many respects due to the underlying differences between ALE and HPSG. In particular, two versions of the grammar have been implemented: a version close to the linguistic theory which, however, led to a much less efficient implementation, and a more efficient version taking into account various non-HPSG mechanisms offered by ALE. We intend to extend the work reported here by constructing an HPSG-based parser of Polish going well beyond the empirical boundaries of the current HPSG grammar presented above.

References

[BMS01] Gosse Bouma, Robert Malouf, and Ivan A. Sag. Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory*, 19(1):1–65, 2001.

[BP99] Robert D. Borsley and Adam Przepiórkowski, editors. *Slavic in Head-Driven Phrase Structure Grammar*. CSLI Publications, Stanford, CA, 1999.

[BP00] Piotr Bański and Adam Przepiórkowski, editors. *Proceedings of the First Generative Linguistics in Poland Conference*, Warsaw, 2000. Institute of Computer Science, Polish Academy of Sciences.

[CP95] Krzysztof Czuba and Adam Przepiórkowski. Agreement and case assignment in Polish: An attempt at a unified account. Technical Report 783, Institute of Computer Science, Polish Academy of Sciences, 1995.

[CP01] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine (Version 3.2.1). User's Guide*. Carnegie Mellon University, Pittsburgh, December 2001.

[Czu95] Krzysztof Czuba. Zastosowanie dziedziczenia do analizy wybranych aspektów języka polskiego. Master's thesis, Uniwersytet Warszawski, Warsaw, 1995.

[KMM00] Anna Kupść, Małgorzata Marciniak, and Agnieszka Mykowiecka. Constituent coordination in Polish: An attempt at an HPSG account. In Bański and Przepiórkowski [BP00], pages 104–115.

[KMMP00] Anna Kupść, Małgorzata Marciniak, Agnieszka Mykowiecka, and Adam Przepiórkowski. Składniowe konstrukcje współrzędne w języku polskim: Próba opisu w HPSG. Technical Report 914, Institute of Computer Science, Polish Academy of Sciences, 2000.

[Mar99] Małgorzata Marciniak. Toward a binding theory for Polish. In Borsley and Przepiórkowski [BP99], pages 125–147.

[Mar01] Małgorzata Marciniak. *Algorytmy implementacyjne syntaktycznych reguł koreferencji zaimków dla języka polskiego w terminach HPSG*. Ph. D. dissertation, Polish Academy of Sciences, 2001.

[MS97] Philip H. Miller and Ivan A. Sag. French clitic movement without clitics or movement. *Natural Language and Linguistic Theory*, 15:573–639, 1997.

[Myk00] Agnieszka Mykowiecka. Polish relative pronouns: An HPSG approach. In Bański and Przepiórkowski [BP00], pages 124–134.

[Par92] Maike Paritong. Constituent coordination in HPSG. Technical Report CLAUS 24, Universität des Saarlandes, Saarbrücken, 1992.

[PK97a] Adam Przepiórkowski and Anna Kupść. Negative concord in Polish. Technical Report 828, Institute of Computer Science, Polish Academy of Sciences, 1997.

[PK97b] Adam Przepiórkowski and Anna Kupść. Verbal negation and complex predicate formation in Polish. In Ralph C. Blight and Michelle J. Moosally, editors, *Proceedings of the 1997 Texas Linguistics Society Conference on the Syntax and Semantics of Predication*, volume 38 of *Texas Linguistic Forum*, pages 247–261, Austin, TX, 1997.

[PK99] Adam Przepiórkowski and Anna Kupść. Eventuality negation and negative concord in Polish and Italian. In Borsley and Przepiórkowski [BP99], pages 211–246.

[PKMM01] Adam Przepiórkowski, Anna Kupść, Małgorzata Marciniak, and Agnieszka Mykowiecka. *Formalny opis języka polskiego: Teoria i implementacja*. Akademicka Oficyna Wydawnicza, 2001. In progress.

[Prz99] Adam Przepiórkowski. *Case Assignment and the Complement-Adjunct Dichotomy: A Non-Configurational Constraint-Based Approach*. Ph. D. dissertation, Universität Tübingen, Germany, 1999.

[Prz00] Adam Przepiórkowski. Long distance genitive of negation in Polish. To appear, *Journal of Slavic linguistics*, 2000.

[Prz01] Adam Przepiórkowski. ARG-ST on phrases headed by semantically vacuous words: Evidence from Polish. In Dan Flickinger and Andreas Kathol, editors, *Proceedings of the 7th International Conference on Head-Driven Phrase Structure Grammar*, pages 267–284. CSLI Publications, Stanford, CA, 2001.

[PS87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. Number 13 in CSLI Lecture Notes. CSLI Publications, Stanford, CA, 1987.

[PS94] Carl Pollard and Ivan A. Sag. *Head-driven Phrase Structure Grammar*. Chicago University Press / CSLI Publications, Chicago, IL, 1994.

[Rea92] Mike Reape. *A Formal Theory of Word Order: A Case Study in West Germanic*. Ph. D. dissertation, University of Edinburgh, 1992.

[Sag97] Ivan A. Sag. English relative clause constructions. *Journal of Linguistics*, 33(2):431–483, 1997.

[Świ92] Marek Świdziński. *Gramatyka Formalna Języka Polskiego*, volume 349 of *Rozprawy Uniwersytetu Warszawskiego*. Wydawnictwa Uniwersytetu Warszawskiego, Warsaw, 1992.

[Szp86] Stanisław Szpakowicz. *Formalny opis składniowy zdań polskich*. Wydawnictwa Uniwersytetu Warszawskiego, Warsaw, 1986.

Psychological Validity of Schematic Proofs

Mateja Jamnik[1] and Alan Bundy[2]

[1] University of Cambridge Computer Laboratory,
J.J. Thomson Avenue, Cambridge, CB3 0FD, England, UK
Mateja.Jamnik@cl.cam.ac.uk
http://www.cl.cam.ac.uk/~mj201
[2] Centre for Intelligent Systems and their Applications, Division of Informatics,
University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN, Scotland, UK
A.Bundy@ed.ac.uk
http://www.dai.ed.ac.uk/homes/bundy/

Abstract. Schematic proofs are functions which can produce a proof of a proposition for each value of their parameters. A schematic proof can be constructed by abstracting a general pattern of proof from several examples of a family of proofs. In this paper we examine several interesting aspects of the use of schematic proofs in mathematics. Furthermore, we pose several conjectures about the psychological validity of the use of schematic proofs in mathematics. These conjectures need testing, hence we propose an empirical study which would either support or refute our conjectures. Ultimately, we suggest that schematic proofs are worthy of a closer and more detailed study and investigation.

1 Introduction

In this paper we study and address several questions about the nature of mathematical proofs. How can a well chosen example often convey the idea of a proof better than the proof itself? How is it possible for proofs to be erroneous, and for such faulty "proofs" to persist for decades? Why are the proofs of some intermediate results less intuitive than the original theorem? We suggest that studying schematic proofs might provide some answers to such questions.

Schematic proofs have been used and studied in various branches of mathematics. Their use has been successfully mechanised in automated mathematical reasoning [1, 2]. We hypothesise that humans often use procedures similar to the construction of schematic proofs. The aim of this paper is to motivate cognitive scientists and cognitive psychologists that schematic proofs are an interesting concept in mathematics and that they are worthy of a closer investigation from a psychological point of view. Such an investigation would shed some light on the nature of human mathematical thought. We examine some interesting aspects of schematic proofs and postulate a number of conjectures about the psychological validity of schematic proofs. We have anecdotal evidence to support our intuitions, however, we have not conducted any systematic experiments. Hence, in §7, we propose an experimental investigation and we suggest some of the questions that such an investigation could attempt to answer.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 321–341, 2005.

Schematic proofs are functions, i.e., programs, which output a proof for each value of their parameters, i.e., inputs. That is, they are a way of capturing a family of proofs[1]. For example, consider a trivial theorem, let us call it *multiple addition*, which says that to get a value of an x in $(\ldots((x+a_1)+a_2)+\cdots+a_n) = y$ one has to subtract all the a_i from y. So, more formally, the theorem can be expressed as $((\ldots((x+a_1)+a_2)+\cdots+a_n) = y) \Rightarrow (x = (\ldots((y-a_n)-a_{n-1})-\cdots-a_2)-a_1)$. The schematic proof for this theorem is the following informal program (where we assume that we have definitions of *proof, apply*, etc.):

$$\mathit{proof}(n) = \mathit{apply}\ \ (U+V=W \Rightarrow U=W-V)\ \ n\ \mathit{times}$$

which rewrites n times, terms in the theorem of the form $U+V=W$ to terms of the form $U=W-V$. A schematic output of this program gives a proof of the *multiple addition* theorem (bold blocks represent program execution steps, i.e., applications of rewrite rules on the theorem):

$$\begin{array}{r}
(((\ldots((x+a_1)+a_2)+\cdots)+a_{n-2})+a_{n-1})+a_n = y \\
\Big\Downarrow \textbf{\textit{apply}}\quad (U+V=W \Rightarrow U=W-V) \\
((\ldots((x+a_1)+a_2)+\cdots)+a_{n-2})+a_{n-1} = y-a_n \\
\Big\Downarrow \textbf{\textit{apply}}\quad (U+V=W \Rightarrow U=W-V) \\
(\ldots((x+a_1)+a_2)+\cdots)+a_{n-2} = (y-a_n)-a_{n-1} \\
\vdots \\
x = (\ldots((y-a_n)-a_{n-1})-\cdots-a_2)-a_1
\end{array}$$

A procedure that can be used to construct schematic proofs is to prove some special cases of a proposition, extract a pattern from these proofs, and abstract this pattern into a general schematic proof. We give examples of proofs for special cases for the above theorem where $n=2$ and $n=3$. When the schematic proof is given an input 2, then the program is instantiated to $\mathit{proof}(2) = \mathit{apply}\ \ (U+V=W \Rightarrow U=W-V)\ \ 2\ \mathit{times}$. The output of this program is:

$$\begin{array}{r}
(x+a_1)+a_2 = b \\
\Big\Downarrow \textbf{\textit{apply}}\quad (U+V=W \Rightarrow U=W-V) \\
x+a_1 = b-a_2 \\
\Big\Downarrow \textbf{\textit{apply}}\quad (U+V=W \Rightarrow U=W-V) \\
x = (b-a_2)-a_1
\end{array}$$

Similarly, when the schematic proof is given an input 3, then the program is instantiated to $\mathit{proof}(3) = \mathit{apply}\ \ (U+V=W \Rightarrow U=W-V)\ \ 3\ \mathit{times}$. The output of this program is:

[1] Schematic proofs are often used as an alternative to mathematical induction (see §2).

$$\begin{aligned}((x + a_1) + a_2) + a_3 &= b \\ &\Downarrow \textbf{\textit{apply}} \quad (U + V = W \Rightarrow U = W - V) \\ (x + a_1) + a_2 &= b - a_3 \\ &\Downarrow \textbf{\textit{apply}} \quad (U + V = W \Rightarrow U = W - V) \\ x + a_1 &= (b - a_3) - a_2 \\ &\Downarrow \textbf{\textit{apply}} \quad (U + V = W \Rightarrow U = W - V) \\ x &= ((b - a_3) - a_2) - a_1\end{aligned}$$

Finally, the schematic proof needs to be shown to be correct, i.e., that *proof*(n) outputs a proof of the theorem for case n. This is discussed in §2. In §3 we give more examples of the use of schematic proofs in mathematics.

There are three particular aspects of schematic proofs that we investigate in some detail. First, we examine how schematic proofs can be constructed from examples of proofs. The mathematical foundation for the construction of schematic proofs provides a justification for the step from examples to general proofs to theorem-hood. So, in §4, our first conjecture is that:

Schematic proofs explain how examples can be used for constructing general proofs.

Second, we examine how schematic proofs have been used in the past to represent claimed proofs of theorems. However, upon closer examination, it turned out in some cases that what was thought to be a proof, was actually faulty and not a proof at all. We argue that this may be due to the omission of the verification of the schematic proof. Hence, in §5, our second conjecture is that:

Schematic proofs account for some erroneous proofs in mathematics.

We give some historical examples which support our conjecture.

Finally, schematic proofs of some theorems can be very different from their standard non-schematic inductive counterparts. They often seem to be more easily understood than inductive proofs. A number of examples are given to support our claim. Therefore, in §6, our third and final conjecture is that:

Schematic proofs are more intuitive than inductive proofs.

1.1 Technical Terminology

Here we give some definitions of technical terms used in this paper that might prove useful. Notice that in the literature, the terms induction, abstraction and generalisation are often used interchangeably for the same concept. We have three different notions for these terms, and hence define them here precisely.

A Recursive Function is a function whose definition appeals to itself without an infinite regression. For example, Hex is a recursive function which for each input natural number n gives the n^{th} hexagonal number:

$$Hex(0) = 0$$
$$Hex(1) = 1$$
$$Hex(n+1) = Hex(n) + 6 \times n$$

The Successor Function is a function that adds one to its argument. For example, $s(s(0)) = s(1) = 2$.

Instantiation is a process of replacing a variable with some value. Instantiation of a function is a process of assigning values to the arguments of the function and evaluating the function for these values. For example, instantiating the above function Hex for 3 gives $Hex(3) = Hex(2+1) = Hex(2) + (6 \times 2) = (Hex(1) + (6 \times 1)) + 12 = 1 + 6 + 12 = 19$.

Abstraction is a process of extracting a general argument from its examples. In this paper it refers to constructing a schematic proof from example proofs. For example, the process of constructing *proof*(n) for the *multiple addition* theorem given above from the examples of its proof for $n = 2$ and $n = 3$ is referred to as abstraction.

Another meaning of abstraction in this paper is to refer to an *abstraction device*, such as ellipsis (i.e., the "..." notation), to represent general diagrams. Abstraction is sometimes referred to as inductive inference, or "philosophical induction", or generalisation.

Generalisation replaces a formula by a more general one. For example, constants, functions or predicates can be replaced by variables (e.g., $x + 3 = y$ is generalised to $x + a = y$ where a constant 3 is replaced by a variable a), or universally quantified variables are decoupled (e.g., $\forall x.(x+x)+x = x+(x+x)$ is generalised to $\forall x \forall y \forall z.(x + y) + z = x + (y + z)$).

Object-Level Statement is a well-formed term, proof or inference step of the logic in use (cf. meta-level statement). For example, the proof of *multiple addition* theorem given above in §1 is an object-level statement.

Meta-level Statement is a statement *about* an object-level statement, in some logical theory (cf. object-level statement). For example, a claim that the proof of *multiple addition* theorem given above in §1, is a correct proof of this theorem, is a meta-level statement about the proof of the *multiple addition* theorem.

Mathematical Induction or standard induction is a rule of inference in some logical theory which is used to prove the statement that some proposition $P(n)$ is true for all values of $n > n_0$, where n_0 is some base value. This rule of inference makes an assertion about object-level statements (cf. meta-induction). For example, in Peano arthmetic, the rule of induction is:

$$\frac{P(0) \quad P(n) \rightarrow P(s(n))}{\forall n.P(n)}$$

Meta-induction is a rule of inference in some logical theory which is used to prove the meta-statement that some proposition $MP(n)$ about the object-level statement $P(m)$ is true for all values of $n > n_0$, where n_0 is some base value. This rule of inference makes an assertion about proofs rather than

object-level statements (cf. mathematical induction). For example, in Peano arithmetic, the rule of meta-induction is (where *proof* is a recursive function, and ":" stands for "is a proof of"):

$$\frac{\mathit{proof}(0) : P(0) \quad \mathit{proof}(n) : P(n) \rightarrow \mathit{proof}(s(n)) : P(s(n))}{\forall n.\mathit{proof}(n) : P(n)}$$

Schematic is an adjective that refers to some general way of describing a class of objects. We use this adjective when describing a program that generates a proof for all instances of some corresponding theorem. We refer to these programs as *schematic proofs*. A formal definition of a schematic proof is given in §2 in Definition 3.

2 Schematic Proofs

Our interest in schematic proofs comes from the perspective of automated reasoning, where the aim is to implement a system which constructs schematic proofs. The automation of proof extraction requires some suitable mechanism to capture a general proof. Schematic proofs provide such a mechanism. General schematic proofs can be constructed from a sequence of instances. A mathematical basis which justifies the step from specific examples to a general schematic proof is the constructive ω-rule [1]. ω is the name given to the infinite set $\{0, 1, 2, 3, \ldots\}$, or equivalently, using the successor function s (see §1.1), the set $\{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$. Typically, a schematic proof is formalised as a recursive program. This recursive program allows us to conclude a general schematic proof for the universally quantified theorem. In this section, we formally define what a schematic proof is, and what is the mathematical basis for its formalisation.

The mathematical basis for extraction of schematic proofs is the constructive ω-rule. This rule is a version of the ω-rule [3]:

Definition 1 (ω-Rule).
The ω-rule allows inference of the sentence $\forall x.\ P(x)$ from an infinite sequence $P(n)$ for $n \in \omega$ of sentences

$$\frac{P(0), P(1), P(2), \ldots}{\forall n.P(n)}$$

Using the ω-rule, an infinite number of premisses needs to be proved in order to conclude a universal statement. This makes the ω-rule unusable for automation. Hence, we consider the constructive version of this rule [1]:

Definition 2 (Constructive ω-Rule).
The constructive ω-rule allows inference of the sentence $\forall x.\ P(x)$ from an infinite sequence $P(n)$ for $n \in \omega$ of sentences

$$\frac{P(0), P(1), P(2), \ldots}{\forall n.P(n)}$$

such that each premiss $P(n)$ is proved **uniformly** *(from parameter n).*

Note that the ω-rule and the constructive ω-rule are stronger alternatives for mathematical induction.

The uniformity criterion is taken to be the provision of a computable procedure describing the proof of $P(n)$, e.g., $proof(n)$. The requirement for a computable procedure is equivalent to the notion that the proofs for all premisses are captured in a recursive function. We refer to such a recursive function as a *schematic proof*.

Definition 3 (Schematic Proof).
A schematic proof is a recursive function[2]*, e.g.,* $proof_P(n)$[3]*, which outputs a proof of some proposition* $P(n)$ *given some* n *as input.*

Suppose the recursive function, *proof*, is a schematic proof. The function *proof* takes one argument, namely a parameter n. In general, this function can be defined to take any number of arguments. By instantiation, i.e., by assigning a particular value to n and passing it as an argument to the function *proof*, and by application of this instantiated function to the theorem, $proof_P(n)$ generates a proof for a particular premiss $P(n)$. More precisely, $proof_P(n)$ describes the inference steps (i.e., rules) made in proofs for each $P(n)$. Now, $proof(n)$ is schematic in n, because we may apply some rule R a function of n (or a constant) number of times. That is, the number of times that a rule R is applied in the proof might depend on the parameter n. This recursive definition of a proof is used as a basis for implementation of the schematic proofs [2, 1].

From a practical point of view, the constructive ω-rule and schematic proofs eliminate the need for an infinite number of proofs, or in other words, they enable us to specify an infinite number of proofs in a finite way. Moreover, they provide a technique which enables an automation of search for proofs of universally quantified theorems from instances of proofs.

We now show how schematic proofs of universally quantified theorems can be found using several heuristics.

2.1 Finding a Schematic Proof

A schematic proof can be constructed by considering individual examples of proofs for instances of a theorem, and then extracting a general pattern from these instances. The idea is that in order to extract a general structure common to all instances of a proof, the particular examples of proofs of a theorem which are considered, need to be general representatives of all instances, and not special cases. These are normally taken to be some intermediate values, e.g., 5 and 6, or 7 and 9, rather than the initial values, e.g., 0 and 1, since the proofs for initial values of a parameter n are almost always special cases. Therefore, we use such intermediate values, e.g., $P(7)$ and $P(9)$ and correspondingly $proof(7)$ and $proof(9)$, to extract the pattern, which we hope is general. A structure

[2] Technical terminology is explained in §1.1

[3] Note that we omit the use of subscript P in $proof_P(n)$ where it is clear which theorem *proof* proves.

which is common to the considered examples is extracted by an abstraction. The result is the construction of a general schematic proof. If the instances for the intermediate values that were considered are not representative of all instances, so that the abstraction was carried out on incomplete information, then the constructed recursive function *proof* could be wrong. Therefore, the function *proof* needs to be verified as correct. This involves reasoning about the proof (using meta-level reasoning), and showing that *proof* indeed generates a correct proof of each $P(n)$.

The following procedure summarises the essence of using the constructive ω-rule in schematic proofs:

1. Prove a few particular cases (e.g., $P(7)$, $P(9)$, ... and thereby discover $\mathit{proof}(7)$, $\mathit{proof}(9)$, ...).
2. Abstract $\mathit{proof}(n)$ from these proofs (e.g., from $\mathit{proof}(7)$, $\mathit{proof}(9)$, ...).
3. Verify that $\mathit{proof}(n)$ proves $P(n)$ by meta-induction[4] on n.

The general pattern is abstracted from the individual proof instances by learning induction or abstraction. By meta-induction we mean that we introduce a theory META such that for all n the base case of the meta-induction is:

$$\text{META} \vdash \mathit{proof}(0) : P(0)$$

and the step case is:

$$\text{META} \quad \vdash \quad \mathit{proof}(n) : P(n) \longrightarrow \mathit{proof}(n+1) : P(n+1)$$

By meta-induction we need to show in the meta-theory that given a proposition $P(n)$, $\mathit{proof}(n)$ indeed proves it, i.e., it gives a correct proof with $P(n)$ as its conclusion, and axioms of some object logic as its premisses. This ensures that the constructed general schematic proof is indeed a correct proof for all instances of a proposition.

3 Application of Schematic Proofs

To illustrate the use of the constructive ω-rule in schematic proofs, we give here five examples of schematic proofs for the following theorems: an arithmetic schematic proof of *associativity of addition* implemented by Baker [1], a schematic proof of *rotate-length theorem*, two diagrammatic schematic proofs, the first of the theorem regarding the *sum of odd naturals* implemented by Jamnik *et al* [2], and the second regarding the *sum of hexagonal numbers* presented by Penrose [4], and a faulty schematic proof of *Euler's* theorem presented by Lakatos in [5].

[4] The meta-induction is often much simpler than the mathematical induction that is alternative to the schematic proof. For example, whereas generalisation is required in some object-level inductive proofs, no generalisation is required in the meta-induction at the verification stage of the corresponding schematic proof. See §4 and §6 for more discussion and some examples.

3.1 Associativity of Addition

Consider a theorem about the *associativity of addition*, stated as

$$(x + y) + z = x + (y + z)$$

Baker studied schematic proofs of such theorems in [1]. The recursive definition of "+" is given as follows:

$$0 + Y = Y \tag{1}$$
$$s(X) + Y = s(X + Y) \tag{2}$$

We also need a reflexive law $\forall n.\ n = n$.

The constructive ω-rule is used on x in the statement of the *associativity of addition*. We write any instance of x as $s^n(0)$. By $s^n(0)$ is meant the n-th numeral, i.e., the term formed by applying the successor function to 0 n times. Next, the axioms are used as rewrite rules from left to right, and substitution is carried out in the ω-proof, under the appropriate instantiation of variables. Hence, the following encoding:

$$\frac{\forall n.(s^n(0) + y) + z = s^n(0) + (y + z)}{\forall x. \quad (x + y) + z = x + (y + z)}$$

where n is the parameter, represents any instance of the constructive ω-rule in our example (note the use of ellipsis):

$$\frac{\begin{array}{l}(0 + y) + z = 0 + (y + z),\ \ (s(0) + y) + z = s(0) + (y + z),\\ (s(s(0)) + y) + z = s(s(0)) + (y + z), \qquad \ldots\end{array}}{\forall x. \quad (x + y) + z = x + (y + z)}$$

We construct a schematic proof in terms of this parameter, where n in the antecedent captures the infinity of premisses actually present, one for each value of n. This removes the need to present an infinite number of proofs. The aim is to reduce both sides of the equation to the same term. The schematic proof of this theorem is the following program:

$$\begin{array}{rl} \textit{proof}(n) = & \text{Apply rule (2) } n \text{ times on each side of equality,}\\ & \text{Apply rule (1) once on each side of equality,}\\ & \text{Apply rule (2) } n \text{ times on left side of equality,}\\ & \text{Apply Reflexive Law}\end{array}$$

Running this program on the associativity theorem proves it. For example:

$$
\begin{array}{rcl}
(s^n(0)+y)+z & = & s^n(0)+(y+z) \\
\Downarrow & & \textbf{Apply rule (2) } n \textbf{ times on each side} \\
\vdots & & \\
s^n(0+y)+z & = & s^n(0+(y+z)) \\
\Downarrow & & \textbf{Apply rule (1) on each side} \\
s^n(y)+z & = & s^n(y+z) \\
\Downarrow & & \textbf{Apply rule (2) } n \textbf{ times on left} \\
\vdots & & \\
s^n(y+z) & = & s^n(y+z) \\
\Downarrow & & \textbf{Apply Reflexive Law} \\
true & &
\end{array}
$$

Note that the number of proof steps depends on n, which is the instance of x we are considering. We see that the proof is schematic in n – certain steps are carried out a number of times depending on n.

3.2 Rotate-Length Theorem

The *rotate-length* theorem is about rotating a list its length number of times, and can be stated as:

$$rotate(length(l), l) = l$$

where $length(l)$ gives the length of a list l, and $rotate(x, l)$ takes the first x elements of a list l and puts them at its end (e.g., $rotate(3, [a, b, c, d, e]) = [d, e, a, b, c]$), and can be defined as:

$$
\begin{aligned}
rotate(0, l) &= l \\
rotate(x, [\,]) &= [\,] \\
rotate(n+1, l :: ls) &= rotate(n, ls@[l])
\end{aligned}
$$

Note that :: is infix cons (it takes an element and a list and puts the element at the front of the list, e.g., $1 :: [2, 3, 4] = [1, 2, 3, 4]$) and @ is infix append (it takes two lists and puts them together, e.g., $[1, 2, 3]@[4, 5] = [1, 2, 3, 4, 5]$). Consider a schematic proof of this theorem. First we give an example proof for some instance of a theorem. An example proof for the instance of a list of any five elements $l = [a, b, c, d, e]$, i.e., $length(l) = 5$ goes as follows. Let the list l consist of five elements. We take the first element of the list and put it to the back of the list. Now, we do the same for the remaining four elements.

$$\begin{aligned} rotate(length([a,b,c,d,e]),[a,b,c,d,e]) &= \\ rotate(5,[a,b,c,d,e]) &= \\ rotate(4,[b,c,d,e,a]) &= \\ rotate(3,[c,d,e,a,b]) &= \\ rotate(2,[d,e,a,b,c]) &= \\ rotate(1,[e,a,b,c,d]) &= [a,b,c,d,e] \end{aligned}$$

It is very easy to see that this process gives us back the original list. Moreover, it is clear that if we follow the same procedure, i.e., schematic proof, for a list of any length, we always get back the original list. Hence, the number of inference steps in the proof depends on n, so a proof is schematic in n:

$$\begin{aligned} rotate(length([a_1,a_2,a_3,\ldots,a_n]),[a_1,a_2,a_3,\ldots,a_n]) &= \\ rotate(n,[a_1,a_2,a_3,\ldots,a_n]) &= \\ rotate(n-1,[a_2,a_3,\ldots,a_n,a_1]) &= \\ rotate(n-2,[a_3,\ldots,a_n,a_1,a_2]) &= \\ \vdots & \\ rotate(1,[a_n,a_1,a_2,a_3,\ldots]) &= [a_1,a_2,a_3,\ldots,a_n] \end{aligned}$$

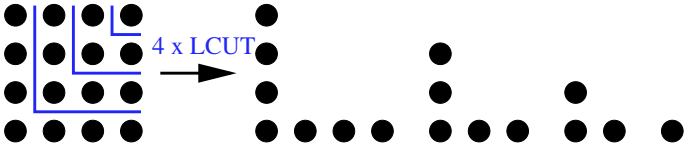
3.3 Sum of Odd Natural Numbers

We now consider a theorem about the *sum of odd naturals* and its schematic proof as studied by Jamnik *et al* in [2] and [6]. Jamnik *et al* studied the notion of diagrammatic proofs and formalisation of diagrammatic reasoning. A diagrammatic proof is captured by a schematic proof that is constructed from examples of graphical manipulations of instances of a theorem. This diagrammatic schematic proof has to be checked for correctness. A diagrammatic proof consists of diagrammatic inference steps, rather than logical inference rules. Diagrammatic inference steps are the geometric operations applied to a diagram. The operations on diagrams produce new diagrams. Chains of diagrammatic inference rules, specified by the schematic proof, form the diagrammatic proof of a theorem. In Jamnik *et al*'s formalisation of diagrammatic reasoning, diagrams are used as an abstract representation of natural numbers, and are represented as collections of dots. Some examples of diagrams are a square, a triangle, an ell (two adjacent sides of a square). Some examples of geometric operations are *lcut* (split an ell from a square), *remove_row*, *remove_column*.

We demonstrate here a diagrammatic proof of the theorem about the *sum of odd natural numbers*. The theorem can be stated as

$$n^2 = 1 + 3 + 5 + \cdots + (2n-1)$$

We consider an instance of the theorem $4^2 = 1+3+5+7$ and its diagrammatic proof where $n = 4$. Let us choose that n^2 is represented by a square of magnitude n, $(2n-1)$ is represented as an ell whose two sides are both n long, i.e., odd natural numbers are represented by ells, and a natural number 1 is represented as a dot. The proof of this instance of the theorem consists of cutting a square 4 times into ells.



Notice, that a similar procedure holds for a square of any size, i.e., for any instance of the theorem. Therefore, these steps are sufficient to transform a square of magnitude n representing the LHS of the theorem to n ells of increasing magnitudes representing the RHS of the theorem.

Note that the number of proof steps (i.e., diagrammatic inference steps) depends on n – for a square of size n the proof consists of n lcuts. Hence the proof is schematic in n. Here is a definition of this schematic proof:

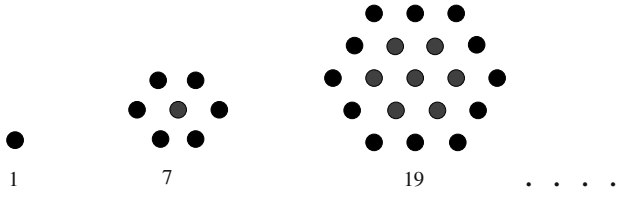
$$\begin{aligned} \mathit{proof}(n+1) &= \mathit{apply\ lcut,\ then\ proof}(n) \\ \mathit{proof}(0) &= \mathit{empty} \end{aligned}$$

3.4 Sum of Hexagonal Numbers

Let us now examine a theorem about the *sum of hexagonal numbers* and its (diagrammatic) schematic proof as presented by Penrose in [4]. We repeat here the formal recursive definition of hexagonal numbers from §1.1:

$$\begin{aligned} Hex(0) &= 0 \\ Hex(1) &= 1 \\ Hex(n+1) &= Hex(n) + 6 \times n \end{aligned}$$

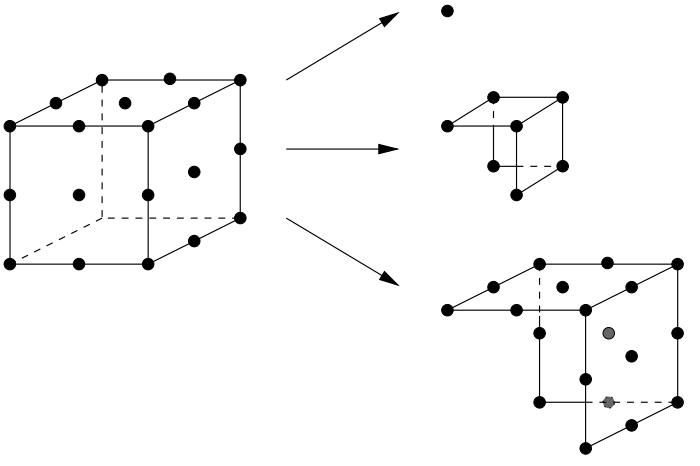
Informally, hexagonal numbers could be presented as hexagons where the hexagonal number is the number of dots in a hexagon:



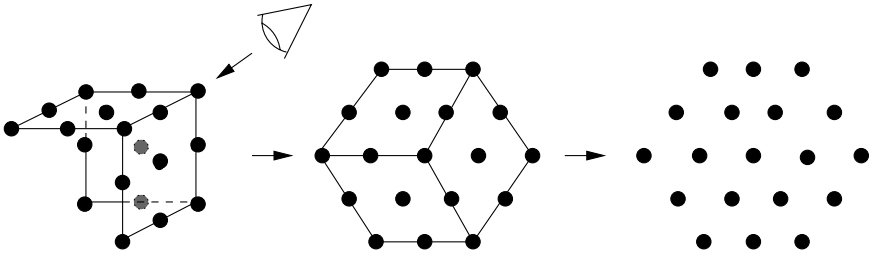
The theorem is stated as follows:

$$n^3 = Hex(1) + Hex(2) + \cdots + Hex(n)$$

Let n^3 be represented by a cube of magnitude n and $Hex(n)$ by an n^{th} hexagon. The instance of the proof that we consider here is for $n = 3$. The diagrammatic proof of the *sum of hexagonal numbers* consists of breaking a cube into a series of half-shells. A half-shell consists of three adjacent faces of a cube.



If each half-shell is projected onto a plane, that is, if we look at the top-right-back corner of each half-shell down the main diagonal of the cube from far enough, then a hexagon can be seen. So the cube is then presented as the sum of all half-shells, i.e., hexagonal numbers.



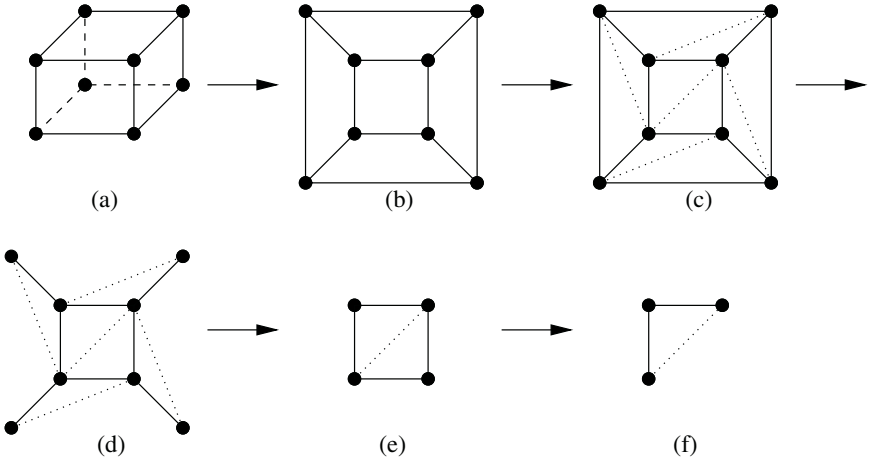
Again, notice that the general proof holds for any instance n. That is, these steps are sufficient to transform a cube of magnitude n representing the LHS of the theorem to n increasing hexagons representing the RHS of the theorem. Note that the number of diagrammatic inference steps depends on the value of n, so the proof is schematic in n.

3.5 Euler's Theorem

Let us consider a famous example of an erroneous schematic "proof", namely, the history of *Euler's theorem* [5]. *Euler's theorem* states that for any polyhedron $V - E + F = 2$ holds, where V is the number of vertices, E is the number of edges, and F is the number of faces. Lakatos[5] initially gives a proof, historically due to Cauchy, of the theorem, which is a uniform method for proving instances of *Euler's theorem*. Thus, the method is a schematic proof. However parts of the method are not explicitly stated, but seem very convincing when applied

[5] The proof of *Euler's theorem* is also discussed in [7, pages 47-48].

to simple polyhedra. Here is a summary of the proof method taken from [5, pages 7-8][6].



Take any polyhedron (note that in our case, we take a cube, but the result is the same for any polyhedron). Imagine that it is hollow, and that its faces are made out of rubber (see (a) of the diagram above). Now, remove one face from the polyhedron, and stretch the rest of the polyhedron onto the plane (see (b) of the diagram). Note that since we have taken one face off, our formula should be $V - E + F = 1$. Note also that the relations between the vertices, edges and faces are preserved in this way. Triangulate all of the faces of this plane network (i.e., we are adding the same number of edges and faces to the network, so the formula remains the same – see (c) of the diagram). Now, start removing the boundary edges (see (d) of the diagram). This will have the effect of removing an edge and a face from the network at the same time, or two edges, one vertex and one face, so our formula is still preserved. We continue removing edges in appropriate order (see (e)), thus preserving the formula, until we are left with one triangle only. Clearly, for this triangle $V - E + F = 1$ holds, since there are three vertices, three edges and one face. Here is an informal diagrammatic schematic proof:

1. remove one face from any given polyhedron,
2. stretch the rest of the polyhedron onto the plane,
3. triangulate all of the faces that are not triangles already,
4. remove the boundary edges one after another, until you are left with a single triangle.

However, this schematic "proof" is faulty, and we will discuss the reasons for this in §5.

[6] The diagram demonstrating the proof of *Euler's theorem* is also taken from [5, page 8].

4 Learning from Examples

Schematic proofs and the constructive ω-rule explain why one or more examples can represent a general proof. Therefore, our first conjecture is that *schematic proofs explain the use of examples for construction of proofs.* Furthermore, we propose that reasoning with concrete cases, i.e., instances or examples, is often more easily understood than reasoning with abstract notions.

As described in §2, the constructive ω-rule enables us to capture infinitary concepts in a finite way. It enables us to use schematic proofs in order to prove universal statements. The constructive ω-rule gives us a mathematical basis which justifies how and why the examples or instances of problems can be used in order to conclude a general statement, in our case a general proof of a universally quantified theorem. We describe two systems which use schematic proofs, and hence reason with instances of theorems in order to prove universally quantified theorems, namely Baker's system CORE which reasons about theorems of arithmetic [1], and Jamnik's system DIAMOND which formalises diagrammatic reasoning [2].

Baker used schematic proofs in order to prove theorems of arithmetic, especially the ones which could not be proved by automated systems without the use of generalisation (for definition, see §1.1). One of Baker's example theorems is a special version of the theorem about *associativity of addition.* In §3.1 we gave a general version of this theorem. Baker's special version of the theorem can be stated as:

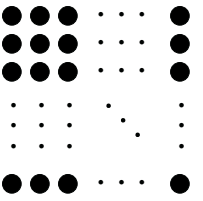
$$(x + x) + x = x + (x + x)$$

The CORE system automatically proves this theorem by enumerating instances of a proof, then constructing a general schematic proof, and finally, verifying that the schematic proof is correct. Instances of the theorem can be encoded as:

$$(s^n(0) + s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0))$$

for each parameter n. The schematic proof of this theorem is identical to the one in §3.1. In a theorem prover that cannot construct schematic proofs, this theorem would normally be proved by mathematical induction. But induction in this case is blocked, as $P(s(n))$ cannot be given in terms of $P(n)$ (for more details see [1]). Hence, generalisation to full associativity $(x+y)+z = x+(y+z)$ is necessary. Rather than using generalisation, as in other automated reasoning systems, CORE was able to prove this theorem using concrete instances of a theorem and its proof.

Jamnik uses schematic proofs for diagrammatic proofs of theorems of natural number arithmetic, like the theorem about the *sum of odd natural numbers* given in §3.3. To devise a general diagrammatic proof of this theorem, one would need to use abstract diagrams, i.e., diagrams of a general size. Therefore, diagrams would have to be represented using abstraction devices, such as ellipsis. Abstraction devices in diagrams are problematic as they are inherently ambiguous. The pattern on either end of the ellipsis needs to be induced by the system. For instance, it is ambiguous whether an abstract collection of rows or columns of dots with ellipsis, like this:



is a square or a rectangle, or if it is of odd or even magnitude. The problem becomes more acute when dealing with more complex structures. To recognise the pattern that the ellipsis represents, the system needs to carry out some sort of pattern recognition technique which deduces the most likely pattern and stores it in an exact internal representation. This guessed pattern might still be wrong. Because of the ambiguity of ellipsis it is difficult to keep track of it during manipulations of diagrams. Schematic proofs are a good way of avoiding this problem, as they allow us to use concrete instances of a theorem and its proof, and yet prove a general theorem. A procedure to construct a schematic proof in DIAMOND and CORE is to first prove instances of a theorem, e.g., a diagram, then construct a schematic proof, and finally prove that this schematic proof is correct. Using instances of a theorem enables us to use concrete diagrams in order to extract formal general proofs.

Besides the ability to extract general proofs from examples, it also appears that reasoning with examples seems easier for humans to understand than reasoning with abstract notions. The usual way in logic to prove Baker's theorem by a mechanised provers is to use mathematical induction and a generalisation, which is difficult to find for both, a human and an artificial mathematician – a mechanised mathematical reasoning system. Furthermore, another way of diagrammatically proving Jamnik's theorem is to reason with abstract diagrams which contain problematic ellipses. Using schematic proofs and instances of theorems seems an easier way to prove these theorems, and seems to convey better why the theorems hold.

5 Erroneous Proofs

A generally accepted definition of a proof of a theorem in mathematical logic is the one given by Hilbert. Here is a translation of a quote from Hilbert's article [8].

> *"Let me still explain briefly just how a* **mathematical proof** *is formalized. As I said, certain formulas, which serve as building blocks for the formal edifice of mathematics, are called axioms. A mathematical proof is an array that must be given as such to our perceptual intuition; it consists of inferences according to the schema*
>
> $$\frac{\mathcal{S} \quad \overset{\mathcal{S}}{\rightarrow} \quad \mathcal{T}}{\mathcal{T}}$$
>
> *where each of the premisses, that is, the formulas $\mathcal{S}$ and $\mathcal{S} \rightarrow \mathcal{T}$ in the array, either is an axiom or results from an axiom by substitution, or else*

> *coincides with the end formula of a previous inference or results from it by substitution. A formula is said to be provable if it is the end formula of a proof."* [9, pages 381-382]

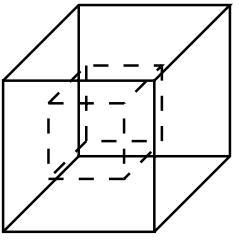
What Hilbert is talking about is sometimes referred to as Hilbert's Programme and is about the axiomatisation of mathematical systems. The definition of a proof in such a system can be summarised as follows. A proof of a theorem is a sequence of inference steps which are valid in some logical theory that has a complete axiomatisation, and which reduces a theorem that also belongs to this logical theory to a set of axioms, i.e., known true facts of the same logical theory.

However, this definition is questionable as it implies that the only explanation for errors in proofs is that they must be syntactic ones. Namely, Hilbert's argument suggests that all proofs boil down to a mechanical exercise of decomposing a theorem into a set of axioms of the theory to which they all belong. We suggest that syntactic errors could be automatically detected during this decomposition, and so erroneous proofs would not survive for years. In mathematics, people do not always formalise all axioms and inferences, yet their justifications for the truthfulness of theorems are generally accepted as correct proofs of theorems. For instance, consider Euclid's proofs of theorems of geometry long before a complete axiomatisation of geometry was given by Hilbert [10].

Mathematical proofs of theorems sometimes turn out to be faulty. The history of mathematics has taught us that there are plenty of faulty proofs of theorems which were for a long time considered to be correct, but later it turned out that the "proofs" were not proofs at all, that is, they were incorrect. Amongst famous examples is Cauchy's proof of the conjecture which says that the limit of any convergent series of continuous functions is itself continuous. Cauchy's "proof" persisted for almost forty years until the faulty conjecture was modified [5]. Another example is the *4-colour conjecture* which had faulty proofs [11]. An interesting discussion of this conjecture and its "proofs" is given in [12], and a correct proof of this theorem can be found in [13]. If Hilbert's definition of a proof was an accurate description of mathematical practice, then these erroneous "proofs" would not arise – any fault in the "proof" would be detected quickly as syntactic error. So what is going on, why do erroneous "proofs" persist?

Clearly, in mathematics in general Hilbert's definition of a proof holds only for a small part of mathematics, namely conjectures in logical theories which have complete axiomatisations. However, not all mathematical conjectures are part of known axiomatised logical theories.

Let us consider the famous example of an erroneous proof of *Euler's theorem*, given in §3.5. Analysing this proof, Lakatos [5] presents a number of counter examples in which the method of proof, i.e., the schematic proof, fails. It turns out that the initial theorem does not hold for *all* polyhedra. For example, it does not hold for hollow polyhedra, e.g., a solid cube with a cubical hole inside it, since $V - E + F = 4$. Note that the schematic proof fails at step 2.



The reader is referred to [5] for a number of counter examples of this theorem. One of the problems with Cauchy's schematic proof is that the definition of a polyhedron is not clearly stated. Therefore, a refinement of a theorem is needed. Lakatos's suggestion for this is to define a polyhedron as a surface and not as a solid. Lakatos proceeds to discuss other counter examples to Cauchy's schematic proof, and finally refines the definition of a polyhedron in a way that *Euler's theorem* does hold. It turns out that the theorems holds for all *simple*[7] [5, page 34] polyhedra whose faces are *simply connected*[8] [5, page 85].

Cauchy used a procedure for construction of schematic proofs in order to convince us of his "proof" of *Euler's theorem*. However, he did not carry out the last step of the procedure for extraction of schematic proofs, namely, he did not verify that the schematic proof is indeed correct[9]. We argue that if he did use the complete procedure, then the fallacy of the procedure would be detected at the verification stage. Note that this would require a constructive definition of a polyhedron.

It seems plausible that humans use some sort of schematic procedure to find general proofs of theorems. In particular, humans often use examples of proofs for certain instances and then abstract them into a general schematic proof. If not all the cases are covered by the examples, then the schematic proof might be incorrect, as in the case of the proof of *Euler's theorem* mentioned above. If a counter example is encountered, then the method needs to be revised to exclude such cases. It seems that humans sometimes omit this step all together. Human machinery for extracting a general schematic argument is usually convincing

[7] Simple polyhedra are ones which can be stretched onto the plane, i.e., those that are topologically equivalent to a sphere.

[8] A surface S is defined to be *connected* if any pair of its points can be joined by a continuous curve lying entirely within the surface. Further, a surface is said to be *simply connected* if any *closed* curve C on the surface divides the surface into *two* distinct regions, each of which is internally connected in the sense just described, and such that any continuous curve which joins a point in one of those regions to a point in the other must cross the closed curve C.

[9] A modern formal proof of *Euler's theorem* was devised only much later and is according to Lakatos [5, page 118] due to Poincaré [14]. It works by representing polyhedra as sets of vertices, edges and faces together with incidence matrices to say which vertices are in each edge and which edges are in each face. A restricted class of polyhedra is then turned into a formulae of vector algebra and a calculation in this algebra gives the value 2 for $V - E + F$. The proof is not intuitively clear, and it is not easy to see why the theorem holds and why this formal proof is correct.

enough to reassure them that the schematic argument is correct, e.g., consider the "proof" of *Euler's theorem*. Humans are happy with intuitive understandings of definitions and steps in the proof – as long as they do not encounter a counter example, their general pattern of reasoning in the proof is acceptable. Lakatos refers to such mathematical proofs as "thought experiments". It is only recently, in the 20th century, that thought experiments were replaced by logical proofs.

In an automated reasoning system, formality is of crucial importance. The correctness of the induced schematic argument has to be formally shown. This confirms that a schematic proof is indeed a correct formal proof of a theorem. If all proofs of theorems that people find followed rules of some formal logic, then there would be no explanation for how erroneous proofs could arise. The errors would always be detected as syntactical errors, provided that the rules used to prove the theorem are correct.

So, our second conjecture is that *human mathematicians often use a procedure similar to the construction of schematic proofs in order to find proofs of theorems, but they often omit the verification step which ensures that the proof is correct.* We propose further, that omitting the verification step of such procedure accounts for numerous examples of faulty "proofs". For instance, if one has not considered all the representative examples, then the schematic proof may not prove all cases of the theorem. A counter example may be found.

6 Intuitiveness of Schematic Proofs

Here, we extend the point in §4 that reasoning with examples or instances of a problem is easier than reasoning with abstract notions. We propose that *schematic proofs seem to correspond better to human intuitive proofs.* It appears easier to see why the theorem holds by looking at the instances of a theorem and its proof and then constructing a schematic proof, than considering a logical proof. As evidence, we give four examples of theorems from §3, where their schematic proofs are easier to understand than formal logical proofs: Baker's proof of *associativity of addition* from §3.1, Jamnik's diagrammatic proof of the *sum of odd naturals* from §3.3, Penrose's *sum of hexagonal numbers* from §3.4, and *rotate-length* theorem from §3.2.

We now consider further the *rotate-length* theorem. The informal schematic proof of this theorem is very easy to understand and to generalise to all cases of any list.

In contrast to a schematic proof of the *rotate-length* theorem, this theorem is not easy to prove by a conventional (non-diagrammatic) theorem prover. The inductive proof of the *rotate-length* theorem usually requires generalisation: e.g., $rotate(length(l), l@k) = k@l$, where @ is the list append function as defined in §3.2. It is harder to see that this theorem is correct. Schematic proofs avoid such generalisations. Baker used schematic proofs to exploit this fact for theorems of arithmetic [1].

We propose that the schematic proof given in §3.2 is a common way that people think about the proof of this theorem. Anecdotal evidence from humans suggests that schematic proofs are psychologically plausible. This supports our conjecture that schematic proofs correspond better to human intuitive proofs.

7 A Proposed Study

In this paper we proposed a number of conjectures about schematic proofs.

1. *Schematic proofs explain the use of examples for inducing formal proofs.*
2. *Schematic proofs account for erroneous proofs.*
3. *Schematic proofs are more intuitive than standard inductive proofs.*

These conjectures are not yet supported by an empirical study, but by our intuition and some suggestive examples. Hence, we propose an experimental study which could support or refute our intuitions. The study would look at some or all of the aspects of schematic proofs addressed in the previous sections. In particular, it would attempt to answer the following questions:

1. Do humans prefer to reason with concrete rather than general cases of a problem? Do humans use a procedure similar to the construction of schematic proofs when solving problems? If so, in what way do they use it and when?
2. Are there other examples which support the conjecture that incomplete schematic proofs account for some erroneous proofs?
3. Is reasoning with examples easier than reasoning with abstract notions? Are schematic proofs more easily understood than formal inductive proofs? If so, why do they appeal to humans more than formal inductive proofs?

The study proposed here would explore human intuitive reasoning in a novel way. We think that humans find schematic proofs easier to understand and more compelling than their logical counterparts. This is also part of the reason why humans might find diagrammatic proofs more intuitive than standard inductive proofs. We have only anecdotal evidence to support our belief. However, a comparative psychological validity experimental study could be carried out to answer some of the questions posed above and to provide some empirical evidence for or against our claims.

The proposed study could take the following form. An experiment could be carried out on a class of students with a certain level of mathematical knowledge (probably final year of secondary school level – the students should be equipped with the notion of mathematical induction). The class should be sufficiently large that the results are statistically significant. The students would be given examples of inductive theorems and non-theorems, and asked if they think the theorem is true or not. If they think it is true, the students would be asked to give an argument why they think it is true. Some of the non-theorems could be those which hold for the majority of cases, but are not true for some special and non-obvious cases. The students would also be asked to provide details of their problem solving process, i.e., the arguments that helped them reach a proof of a theorem or a conclusion that the theorems does not hold.

The data collected from the students would be analysed. Here are a few aspects that could be addressed in the analysis:

- classification of problem solving strategies using some existing techniques,
- analysis of whether the arguments used in the proof are inductive, schematic (using something like the constructive ω-rule), or some other type,
- analysis of the responses for non-theorems which are true for most cases, but not true for some more obscure special cases:
  - If the students realise that the conjecture is a non-theorem, how did they discover this (especially in the case of a schematic argument)?
  - If the students do not realise that the conjecture is a non-theorem, what are the arguments that falsely reassure them that the conjecture is a theorem and that it is true?

Another test that the students could be given consists of theorems and non-theorems, and their proofs and faulty "proofs" respectively. Each (non-) theorem could be accompanied with, say, three different (faulty) proofs each following a different strategy, e.g., inductive, schematic or other. In the case of non-theorems, the inductive argument would contain some syntactic errors and the schematic argument would not be verified for correctness. The students would be asked to choose the proof that is most convincing and that they think they understand best, and to elaborate on the reasons for their choice.

The questions which should be studied in more detail before the experiment is conducted include how much mathematical knowledge and knowledge of logic should the students have. Should they be trained in mathematical induction, constructive ω-rule, and other problem solving techniques? The danger is that people who have some training in mathematics, but not in logic would solve problems differently from those trained in logic, or those with little knowledge of mathematics and logic. Hence, the results would say less about the nature of proofs than about the abilities of individual students. A possibility is to separate subjects into two or more groups according to their level of training, and study the data according to these groups.

Here, we gave some preliminary suggestions for the design of the proposed experimental study. However, these ideas should be investigated in much greater detail before an experiment is conducted.

8 Conclusion

In this paper we posed several conjectures about the use of schematic proofs in mathematics. These conjectures make claims about the psychological validity of schematic proofs. First, we suggested that humans often use examples in order to conclude a general mathematical statement. Second, we conjectured that incomplete schematic proofs account for some erroneous proofs. Our suggestion is that looking at faulty proofs that have survived for years might give us useful insights into human reasoning. Finally, we conjectured that often schematic proofs are more intuitive than their inductive counterparts. These three conjectures are only supported by anecdotal evidence, so there is a clear need for a scientific experimental study which would test them. The motivation for this work is to investigate the nature of human mathematical thought and the notion

of mathematical proof. Schematic proofs provide a good case study for such an investigation. Hence, our aim was to demonstrate that schematic proofs are worthy of a further study by cognitive scientists, and to propose the sort of questions that such an experiment could aim to answer. We hope that we provided enough evidence and motivation that the study of psychological validity of schematic proofs will be seen as a profitable scientific investigation, and will ultimately lead to further research and useful results.

References

1. Baker, S., Ireland, A., Smaill, A.: On the use of the constructive omega rule within automated deduction. In Voronkov, A., ed.: International Conference on Logic Programming and Automated Reasoning – LPAR-92. Number 624 in Lecture Notes in Artificial Intelligence, Berlin, Germany, Springer Verlag (1992) 214–225
2. Jamnik, M., Bundy, A., Green, I.: On automating diagrammatic proofs of arithmetic arguments. Journal of Logic, Language and Information **8** (1999) 297–321
3. Sundholm, B.: A Survey of the Omega Rule. PhD thesis, University of Oxford, Oxford, UK (1983)
4. Penrose, R.: Mathematical intelligence. In Khalfa, J., ed.: What is Intelligence?, Cambridge, UK, The Darwin College Lectures, Cambridge University Press (1994) 107–136
5. Lakatos, I.: Proofs and Refutations: The Logic of Mathematical Discovery. Cambridge University Press, Cambridge, UK (1976)
6. Jamnik, M.: Mathematical Reasoning with Diagrams: From Intuition to Automation. CSLI Press, Stanford, CA (2001)
7. Gamow, G.: One two three ... infinity: Facts & Speculations of Science. Dover Publications, New York (1988)
8. Hilbert, D.: Über das Unendliche. Mathematische Annalen **95** (1926) 161–190
9. Van Heijenoort, J., ed.: From Frege to Gödel – A Source Book in Mathematical Logic, 1879-1931. Havard University Press, Harvard, MA (1967)
10. Hilbert, D.: Grundlagen der Geometrie. Teubner, Stuttgart, Germany (1899) English translation 'Foundations of Geometry' published in 1902 by Open Court, Chicago.
11. Kempe, A.: On the geographical problem of the four colours. American Journal of Mathematics **2** (1879) 193–200
12. Mackenzie, D.: Mechanizing Proof. MIT Press, Boston, MA (2001)
13. Appel, K., Haken, W.: Every planar map is four colorable. Bulletin of the American Mathematical Society **82** (1976) 711–712
14. Poincaré, H.: Complément à l'analysis situs. Rendiconti del Circolo Matematico di Palermo **13** (1899) 285–343

Natural Language Proof Explanation

Armin Fiedler

FR Informatik, Universität des Saarlandes,
Postfach 15 11 50, D-66041 Saarbrücken, Germany
afiedler@cs.uni-sb.de

Abstract. State-of-the-art proof presentation systems suffer from several deficiencies. First, they simply present the proofs without motivating why the proof is done as it is done. Second, they neglect the issue of user modeling and thus forgo the ability to adapt the presentation to the specific user. Finally, they do not allow the user to interact with the system to ask questions about the proof.
As a first step to overcome these deficiencies, we developed a computational model of user-adaptive proof explanation, which is implemented in a generic, user-adaptive proof explanation system, called ***P.rex*** (for ***pr****oof* ***ex****plainer*). To do so, we used techniques from three different fields, namely from computational logic to represent proofs ensuring the correctness; from cognitive science to model the users mathematical knowledge and skills; and from natural language processing to plan the explanation of the proofs and to react to the user's interactions.

1 Introduction

Today, automated theorem provers are becoming increasingly important in practical industrial applications and increasingly useful in mathematical education. For many applications, it is important that a deduction system communicates its proofs reasonably well to the human user. To this end, proof presentation systems have been developed.

However, state-of-the-art proof presentation systems suffer from several deficiencies. First, they simply present the proofs, at best in a textbook-like format, without motivating why the proof is done as it is done. Second, they neglect the issue of user modeling and thus forgo the ability to adapt the presentation to the specific user, both with respect to the level of abstraction chosen for the presentation and with respect to steps that are trivial or easily inferable by the particular user and, therefore, should be omitted. Finally, they do not allow the user to interact with the system. He can neither inform the system that he has not understood some part of the proof, nor ask for a different explanation. Similarly, he cannot ask follow-up questions or questions about the background of the proof.

As a first step to overcome these deficiencies, we developed a computational model of user-adaptive proof explanation, which is implemented in a generic, user-adaptive proof explanation system, called ***P.rex*** (for ***pr****oof* ***ex****plainer*). The

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 342–363, 2005.

demands we make on the system are the following: The system should adapt to the user with respect to the level of abstraction at which the proof is presented. Moreover, the system should allow the user to intervene if he is not satisfied with an explanation and to ask follow-up or background questions. The metaphor we have in mind is a human mathematician who teaches a proof to a student or else explains a proof to a colleague.

In our approach, we combine techniques from three different fields, namely cognitive science, computational logic and natural language processing, as we shall discuss in the following.

Modern natural language generation systems take into account the intended audience's knowledge in the generation of explanations (see e.g. [7, 39, 42]). Most of them adapt to the addressee by choosing between different discourse strategies. Since proofs are inherently rich in inferences, the explanation of proofs must also consider which inferences the audience can make [44, 19, 20]. However, because of the constraints of the human memory, inferences are not chainable without costs. Explicit representation of the addressee's cognitive states proves to be useful in choosing the information to convey [43].

In cognitive science various theories of human cognition have been described by means of a *cognitive architecture*, that is, the fixed structure that realizes the cognitive apparatus. One of these theories is ACT-R [1], a cognitive architecture that separates declarative and procedural knowledge into a declarative memory and a production rule base, respectively. A goal stack allows ACT-R to fulfill a task by dividing it into subtasks, which can be fulfilled independently.

ACT-R combines the abilities for *user modeling* and *planning* in a uniform framework and is therefore particularly well suited as a basis for a user-adaptive dialog planner. Using ACT-R, we model a teacher who explains mathematical proofs to his student. In particular, we model the teacher's assumptions about the students cognitive states during the explanation (which establish the user model) and the teacher's knowledge of the mathematical theories and the way to explain the proof of a theorem in these theories (which is used to plan the explanation). The assumptions about the user's cognitive states are employed to choose the level of abstraction at which a proof is presented.

An important prerequisite that enables *P.rex* to choose among different levels of abstraction is the simultaneous representation of a proof at several levels of abstraction. This representation, which also serves as the interface between theorem provers and *P.rex*, is realized in TWEGA, an implementation of an extension of the *logical framework* LF [17] that corresponds to the *calculus of constructions* [9]. Both are very powerful typed lambda calculi that allow us to represent other logical calculi, and thus give us the possibility to represent the calculus of the theorem provers that are to be connected to *P.rex*. The input proof to be explained as well as relevant information from the mathematical theories that relate to the proof are encoded in TWEGA and then used by *P.rex*. In particular, using an expansion mechanism that defines for any derivation step a subproof at the lower level of abstraction, TWEGA allows us to represent a proof at several levels of abstraction simultaneously.

Successful communication via a natural language generation system presupposes that the content to be conveyed is appropriately structured to ensure a coherent semantic organization. For an explanation system, it is also important to accept user feedback and follow-up questions. To be able to clarify misunderstood explanations, the system needs to represent the different parts of the explanation as well as the relations between them.

An appropriate discourse theory that allows for these representations was formulated by Mann and Thompson [30]. This theory, called *Rhetorical Structure Theory (RST)*, states that the relations that hold between segments of normal English text can be represented by a finite set of relations. Based on RST, Hovy [21] described discourse as the nesting of discourse segments. According to his discourse theory, each segment essentially contains the communicative goal the speaker wants to fulfill with this segment and either one to several discourse segments with intersegment discourse relations or the semantic material to be communicated. Adapting Hovy's discourse segments, we define *discourse structure trees* as a representation of discourses for *P.rex*. The discourse structure trees are built by plan operators, which are defined in terms of ACT-R production rules.

This paper is organized as follows: First, in Section 2 we shall review relevant research in proof presentation and formulate requirements for proof explanation. Next, in Section 3, we shall give an overview of the architecture of *P.rex*. Section 4 is devoted to ACT-R, the theory of human cognition that serves as a basis for the dialog planner of *P.rex*. The dialog planner itself is the subject of Section 5. We shall define discourse structure trees to represent the discourse and plan operators to construct them. In particular, we shall show how the system uses special plan operators to adapt its explanations to the user. Moreover, we shall discuss the system's reaction to a user's interactions. This will be illustrated by some examples in Section 6.

2 Proof Presentation

While the field of automated theorem proving matured in the last four decades, it became more and more apparent that the systems had to output proofs that can be more easily understood by mathematicians. To provide the proofs in the internal, machine-oriented formalisms of the theorem provers is by far not sufficient. Thus, proof presentation systems had to be designed that presented proofs in natural language at an appropriate level of abstraction.

The problem of obtaining a natural language proof from a machine-found proof can be divided into two subproblems: First, the machine-found proof is transformed into a human-oriented calculus, which is much better suited for presentation. Second, the transformed proof is verbalized in natural language. In the following, we shall examine both stages in some more detail.

2.1 Proof Transformation

Already in the thirties, Gentzen devised a human-oriented calculus that aimed to reflect the way mathematicians reason, the *natural deduction (ND) calculus*

[16]. However, most automated theorem provers are based on machine-oriented formalisms, such as resolution or matings. As a consequence, proofs encoded in such formalisms are not suited for a direct verbalization, because their lines of reasoning are often unnatural and obscure. To remedy this problem, researchers developed algorithms to transform proofs in machine-oriented calculi into ND proofs [2, 27, 35, 40, 29].

Initially, the idea was that the transformation into an ND proof is sufficient for the subsequent verbalization. But the results of the transformations into the ND calculus turned out to be far from satisfactory. The reason is that the obtained ND proofs are very large and too involved in comparison to the original proof. Moreover, in the ND calculus, an inference step merely consists of the syntactic manipulation of a quantifier or a connective. Huang [23] realized that in human-written proofs, in contrast, an inference step is often described in terms of the application of a definition, axiom, lemma or theorem, which he collectively calls *assertions*. Based on this observation, he defined an abstraction of ND proofs, called the *assertion level*, where a proof step may be justified either by an ND inference rule or by the application of an assertion, and gave an algorithm to abstract an ND proof to an assertion level proof [23]. Based on Huang's ideas, Meier described an algorithm to transform refutation graphs (a data structure that represents resolution proofs) directly into assertion level proofs [33]. The assertion level proves to be much better suited for a subsequent verbalization of the proofs than a traditional calculus. However, the assertion level sometimes includes steps that include implicit applications of modus tollens, which prove to be difficult to comprehend for human beings. Therefore, Horacek introduced the *partial* assertion level, where these applications are made explicit [20].

Since traditional search-based automated theorem provers find only proofs for theorems that are usually considered easy by mathematicians, theorem provers based on human-oriented approaches such as planning have been developed [5, 3]. The key idea here is that proof techniques as used by mathematicians are encoded into plan operators, which are used by the proof planner to find a proof plan. Because a proof plan is an abstract representation of a proof, it provides a level that is better suited for presentation, such that proof transformation becomes obsolete for proof planners.

2.2 Proof Verbalization

Now, let us turn our attention to approaches to verbalize proofs. One of the earliest proof presentation systems was EXPOUND [8]. It translated the formal proofs directly into English. Even though sophisticated techniques were developed to plan the paragraphs and sentences that made up the written proof, the system verbalized every single step of the formal proof in a template driven way, such that even small proofs are too detailed and still not easy to follow.

Proofs were also used as test input for early versions of MUMBLE [32], a natural language generation (NLG) system that adopted more advanced generation techniques. However, its main concern was not proof presentation, but to show the feasibility of its two-staged architecture for the generation of natural language.

Whereas the previously mentioned systems focused on problems in natural language generation and used formal proofs only as well-defined input for the generation process, research in the field of automated theorem proving addressed the readability of proofs as well. The following systems were developed in the field of automated theorem proving with the aim to obtain human-readable proofs.

The χ-proof system [14] was one of the first theorem provers that was designed with a natural language output component. The system showed every step of the derivation by using predefined templates with English words, but left the formulae in their logical form. The same is true for the pseudo-natural language presentation components of Coq [10] and the proof system *Theorema* [4]. The latter additionally allows the user to hide or unhide proof parts that he considers too detailed or not detailed enough, respectively.

Natural Language Explainer [12] was devised as a back end for the natural deduction theorem prover THINKER. Employing several isolated strategies, it was the first system to acknowledge the need for higher levels of abstraction when explaining proofs. Another presentation system that uses templates with canned sentence chunks to verbalize proofs is ILF [11]. It has been connected to several automated theorem provers, whose output proofs are presented in natural language at the logic level of the machine-oriented formalism of the respective prover.

PROVERB [22, 24] can be seen as the first serious attempt to build a generic, comprehensive NLG system that produces adequate argumentative texts from machine-found proofs. It takes as input ND proofs, which are first abstracted to the assertion level before any subsequent processing starts. *PROVERB* consists of a text planner, which chooses the information to be conveyed, a sentence planner, which plans the internal structure of the sentences, and the linguistic realizer TAG-GEN [26], which produces the output text. The system uses presentation knowledge and linguistic knowledge to plan the proof texts, which are output in a textbook-like format.

Another recently developed NLG system that is used as a back end for a theorem prover is the presentation component of Nuprl [18]. The system consists of a pipeline of two components. It employs a content planner that selects the information to be included in the output text and decides how to refer to the information in the given context. The text plan is then passed on to the surface realizer FUF [13], which chooses the words and outputs the actual sentences.

To sum up, to remedy the problem that many theorem provers return proofs in their internal, machine-oriented formalisms, which are very hard to understand, more and more human-oriented interfaces for theorem provers have been developed. But these interfaces are mostly used in the theorem proving community, that is, by people who usually have an insight in the provers' formalisms. The systems present proofs mostly at a very low level of abstraction and none of the systems adapts its output to the user or can handle follow-up questions. Whereas this might be tolerable for the developers of the theorem provers, it is certainly not acceptable for mathematicians or mathematics students who want to work with theorem provers.

2.3 Requirements for the Explanation of Proofs

When designing a proof explanation system, we may study a mathematics teacher and examine how he explains proofs to his students.

In education, human teachers use *natural language* for the presentation and explanation. Mathematics teachers often experience that students are demotivated by an overload of formulae. Hence, it can be expected that a proof explanation system, in particular if used by novices, is much more accepted if it also communicates derivations and, to some extend, formulae in natural language.

Many concepts and ideas are much easier to understand when they are depicted graphically; the inclusion of graphs and diagrams in addition to natural language is standard routine in mathematics communication. The computer allows us to go beyond these traditional ways of presentation and to include parameterized animations as well, which, for example, display how a diagram changes when parameters are varied. That is, a *multi-modal* user interface would be desirable.

The major advantage of a teacher in comparison to a textbook is that the students can interact with the teacher during the lesson. For example, they can ask the teacher when they do not understand a derivation. These forms of *interaction* should be supported by an intelligent explanation system as well.

To communicate a proof, the teacher has to present individual proof steps choosing a degree of *explicitness*. Usually, he does not mention all proof steps explicitly by giving the premises, the conclusion and the inference method. Often, he only hints implicitly at some proof steps (e.g., by giving only the inference method) when the hint is assumed to suffice for the student to reconstruct the proof step. Other proof steps are completely omitted when they are obvious or easy to infer.

However, a presentation that consists of a mere enumeration of proof steps is often unnatural and tedious to follow. Therefore, teachers add many *explanatory comments*, which motivate a step or explain the structure of a subproof, for example, by stressing that a case analysis follows.

Both decisions whether a proof step is only hinted at or omitted, and whether an explanatory comment is given are usually made *relative to the context*. For example, if a premise A of a proof step with conclusion B is used immediately after it was derived, the teacher only hints at it by saying: "Therefore, B holds." But if the premise A was derived a while ago he explicitly mentions it by saying: "Since A, we obtain B." A similar argument holds for explanatory comments. For example, if it is obvious that there is a case analysis it is not explicitly introduced, but the cases are presented right away.

The level of *abstraction* at which the proof is presented plays a major role. For example, the author of a textbook has an idea of his intended audience and adapts his presentation to that audience. Likewise, a teacher takes into account the abilities of his students when he decides at which level of abstraction he presents a derivation. To choose between different levels of abstraction, an intelligent proof explanation system needs a *user model*, which records the facts and inference methods the user knows.

Finally, a proof explanation system should also account for different *presentation strategies* with respect to the purpose of the session. For example, different strategies are needed when mere mathematical facts are to be conveyed in contrast to when mathematical skills are to be taught. In the former case, a proof can be presented in textbook style where the essential proof steps are shown. In the latter case, the presentation should be structured differently to convey also control knowledge, which explains *why* the various proof steps are taken as opposed to just show *that* they are taken [28]. These two strategies reflect the difference in the difficulty of checking the correctness of a proof versus finding a proof.

P.rex is the first attempt to build an interactive, user-adaptive proof explanation system. Except for multi-modality, we shall address all the requirements formulated in this paper. Although graphs and diagrams play an important role when included in a presentation, they occur only in certain mathematical theories and even in these theories, they do not occur very often. Therefore, we decided for the moment to neglect the generation of graphs and diagrams in our system.

3 The Architecture of *P.rex*

P.rex is a generic proof explanation system that can be connected to different theorem provers. Except for multi-modality, it fulfills all requirements given in Section 2.3. In this section, we shall give an overview of the architecture of *P.rex*, which is displayed in Figure 1.

As the interface between theorem provers and *P.rex*, we defined the formal language Twega, the implementation of an extension of the logical framework LF [17] that corresponds to the calculus of constructions [9]. Both are very powerful calculi from type theory that allow us to represent other logical calculi and, thus, to represent the proofs of most currently implemented theorem provers. Hence, Twega ensures that *P.rex* can be connected to these theorem provers. The type-theoretic foundations of Twega guarantee that only those proofs can be represented that are correct with respect to the calculus of the corresponding prover. The calculus of the prover, the input proof to be explained, as well as relevant information from the mathematical theories that relate to the proof are represented in Twega for further use by *P.rex*. An expansion mechanism that defines for any derivation step a subproof at the lower level of abstraction allows Twega to represent a proof at several levels of abstraction simultaneously.

The central component of *P.rex* is the *dialog planner*. It chooses the content and determines the order of the information to be conveyed, and organizes the pieces of information in a rhetorical structure, called *discourse structure*. The discourse structure also specifies the large-scale segmentation of the discourse into paragraphs.

The dialog planner is implemented in ACT-R, a production system that aims to model the human cognitive apparatus [1]. ACT-R separates declarative and procedural knowledge into a declarative memory and a production rule base,

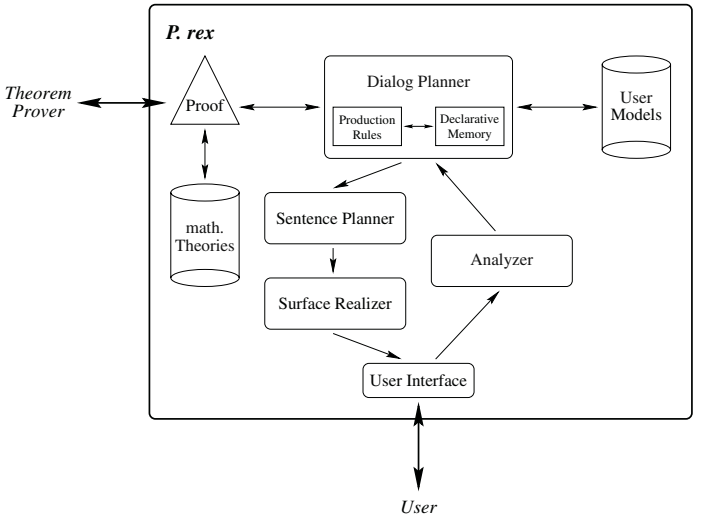


Fig. 1. The Architecture of *P.rex*.

respectively. A goal stack allows ACT-R to fulfill a task by dividing it into subtasks, which can be fulfilled independently. We shall review ACT-R briefly in Section 4.

The plan operators of the dialog planner organize the discourse structure. They are defined in terms of ACT-R productions. A discourse history is represented in the declarative memory by storing conveyed information. Moreover, presumed declarative and procedural knowledge of the user is encoded in the declarative memory and the production rule base, respectively. This establishes that the dialog planner is modeling the user.

In order to explain a particular proof, the dialog planner first assumes the user's cognitive state by updating its declarative and procedural memories. This is done by looking up the user's presumed knowledge in the user model, which was recorded during a previous session. An individual model for each user persists between the sessions. It is stored in the database of *user models*. Each user model contains assumptions about the knowledge of the user that is relevant to the proof explanation. In particular, it makes assumptions about which mathematical theories the user knows and which definitions, proofs, inference methods and mathematical facts he knows.

After updating the declarative and procedural memories, the dialog planner sets the global goal to show the proof. ACT-R tries to fulfill this goal by successively applying productions that decompose or fulfill goals. Thereby, the dialog planner not only produces a dialog plan, but also traces the user's cognitive states in the course of the explanation. This allows the system both to always choose an explanation adapted to the user, and to react to the user's interactions in a flexible way: The dialog planner interprets an interaction in terms of applications of productions. Then it plans an appropriate response. Section 5 is devoted to the dialog planner.

The dialog plan produced by the dialog planner is passed on to the *sentence planner*. We adapted and extended *PROVERB*'s micro-planner [15, 24] to use it in *P.rex* to plan the internal structure of the sentences. The sentence planner aggregates domain concepts when possible and maps them into a linguistic structure. The linguistic structure specifies the lexical items and referring expressions that realize the domain concepts as well as the small-scale segmentation, that is, the scope of the phrases and sentences.

The linguistic structure is then realized by the syntactic generator TAG-GEN [26], which ensures the correct morphology of the surface words. Note that dialog planner, sentence planner and surface realizer are organized in a pipeline.

The utterances that are produced by *P.rex* are presented to the user via a *user interface* that allows the user to enter remarks, requests and questions. An *analyzer* receives the user's interactions, analyzes them and passes them on to the dialog planner. Since natural language understanding is beyond the scope of this work, we use a simplistic analyzer that understands a small set of predefined interactions.

4 ACT-R: A Cognitive Architecture

In cognitive science several approaches are used to describe the functionality of the cognitive apparatus, for example, production systems, mental models or distributed neural representations. Production systems that model human cognition are called *cognitive architectures*. In this section we describe the cognitive architecture ACT-R [1], which is well suited for user adaptive explanation generation because of its conflict resolution mechanism. Further examples for cognitive architectures are SOAR [38] and EPIC [34].

ACT-R has two types of knowledge bases, or *memories*, to store permanent knowledge in: declarative and procedural representations of knowledge are explicitly separated into the declarative memory and the procedural production rule base, but are intimately connected.

Procedural knowledge is represented in production rules (or simply: *productions*) whose conditions and actions are defined in terms of declarative structures. A production can only apply if its conditions are satisfied by the knowledge currently available in the declarative memory. An item in the declarative memory is annotated with an activation that influences its retrieval. The application of a production modifies the declarative memory, or it results in an observable event. The set of applicable productions is called the *conflict set*. A *conflict resolution* heuristic derived from a rational analysis of human cognition determines which production in the conflict set will eventually be applied.

In order to allow for a goal-oriented behavior of the system, ACT-R manages goals in a goal stack. The current goal is that on the top of the stack. Only productions that match the current goal are applicable.

4.1 Declarative Knowledge

Declarative knowledge is represented in terms of *chunks* in the declarative memory. Below is an example for a chunk encoding the fact that $F \subseteq G$, where

`subset-fact` is a concept and `F` and `G` are contextual chunks associated to `factFsubsetG`.

```
factFsubsetG
  isa  subset-fact
  set1 F
  set2 G
```

Chunks are annotated with continuous activations that influence their retrieval. The activation A_i of a chunk C_i consists of its base-level activation B_i and the weighted activations of contextual chunks. In B_i, which is defined such that it decreases logarithmically when C_i is not used, ACT-R models the forgetting of declarative knowledge. Note that the definition of the activation establishes a spreading activation to adjacent chunks, but not further; multi-link-spread is not supported.

The constraint on the capacity of the human working memory is approached by defining a retrieval threshold τ, where only those chunks C_i can be matched whose activation A_i is higher than τ. Chunks with an activation less than τ are considered as forgotten.

New declarative knowledge is acquired when a new chunk is stored in the declarative memory, as is always the case when a goal is popped from the goal stack. The application of a production may also cause a new chunk to be stored if required by the production's action part.

4.2 Procedural Knowledge

The operational knowledge of ACT-R is formalized in terms of *productions*. Productions generally consist of a condition part and an action part, and can be applied if the condition part is fulfilled. In ACT-R both parts are defined in terms of chunk patterns. The condition is fulfilled if its first chunk pattern matches the current goal and the remaining chunk patterns match chunks in the declarative memory. An example for a production is

IF the current goal is to show that $x \in S_2$ and it is known that $x \in S_1$ and $S_1 \subseteq S_2$
THEN conclude that $x \in S_2$ by the definition of $\subseteq$

Similar to the base-level activation of chunks, the strength of a production is defined such that it decreases logarithmically when the production is not used. The time spent to match a production with a chunk depends on the activation of the chunk[1]. It is defined such that it is negative exponential to the sum of the activation of the chunk and the strength of the production. Hence, the higher the activation of the chunk and the strength of the production, the faster the production matches the chunk. Since the activation must be greater than the retrieval threshold τ, τ constrains the time maximally available to match a production with a chunk.

[1] In this context, *time* does not mean the CPU time needed to calculate the match, but the time a human would need for the match according to the cognitive model.

The conflict resolution heuristic starts from assumptions on the probability P that the application of the current production leads to the goal and on the costs C of achieving that goal by this means. Moreover G is the time maximally available to fulfill the goal. The net utility E of the application of a production is defined as

$$E = PG - C. \tag{1}$$

We do not go into detail on how P, G and C are calculated. For the purposes of this paper, it is sufficient to note that G only depends on the goal, but not on the production.

To sum up, in ACT-R the choice of a production to apply is as follows:

1. The conflict set is determined by testing the match of the productions with the current goal.
2. The production p with the highest utility is chosen.
3. The actual instantiation of p is determined via the activations of the corresponding chunks. If no instantiation is possible (because of τ), p is removed from the conflict set and the algorithm resumes in step 2, otherwise the instantiation of p is applied.

ACT-R also provides a learning mechanism, called *production compilation*, which allows for the learning of new productions.

5 Dialog Planning

In the community of NLG, there is a broad consensus that it is appropriate to generate natural language in three major steps [41, 6]. First, a *text planner* determines what to say, that is, content and order of the information to be conveyed. Then, a *sentence planner* determines how to say it, that is, it plans the scope and the internal structure of the sentences. Finally, a *linguistic realizer* produces the surface text. In this classification, the dialog planner is a text planner for managing dialogs.

The dialog planner of *P.rex* plans the dialog by building a representation of the structure of the discourse that includes speech acts as well as relations among them. The discourse structure is represented in the declarative memory. The plan operators are defined as productions.

5.1 Discourse Structure

Drawing on *Rhetorical Structure Theory (RST)* [30], Hovy argues in favor of a single tree to represent a discourse [21]. He considers a discourse as a structured collection of clauses, which are grouped into segments by their semantic relatedness. The discourse structure is expressed by the nesting of segments within each other according to specific relationships (i.e., RST relations). According to his discourse theory, each segment essentially contains the communicative goal the speaker wants to fulfill with this segment and either one to several discourse segments with intersegment discourse relations or the semantic material to be communicated.

Similarly to Hovy's approach, we describe a discourse by a *discourse structure tree*, where each node corresponds to a segment of the discourse. The speech acts, which correspond to minimal discourse segments, are represented in the leaves. We achieve a linearization of the speech acts by traversing the discourse structure tree depth-first from left to right. Explicit representation of the discourse purpose allows for presentations using different styles. Moreover, discourse structure trees also account for restricted types of dialogs as well, namely certain types of interruptions and clarification dialogs. This is a necessary prerequisite for the system to represent user interactions and to appropriately react to them.

5.2 Speech Acts

Speech acts are the primitive actions planned by the dialog planner. They represent frozen rhetorical relations between exchangeable semantic entities. The semantic entities are represented as arguments to the rhetorical relation in the speech act. Each speech act can always be realized by a single sentence. We use speech acts in *P.rex* not only to represent utterances that are produced by the system, but also to represent utterances from the user in the discourse.

We distinguish two major classes of speech acts. First, *mathematical communicative acts (MCAs)* are employed to convey mathematical concepts or derivations. MCAs suffice for those parts of the discourse, where the initiative is taken by the system. Second, *interpersonal communicative acts (ICAs)* serve the dialog, where both the system and the user alternately take over the active role.

Mathematical Communicative Acts. *Mathematical communicative acts (MCAs)* are speech acts that convey mathematical concepts or derivations. Our class of MCAs was originally derived from *PROVERB*'s PCAs [22], but has been substantially reorganized and extended. We distinguish two classes of MCAs:

- *Derivational MCAs* convey steps of the derivation, which are logically necessary. Failing to produce a derivational MCA makes the presentation logically incorrect. The following is an example for a derivational MCA given with a possible verbalization:

  `(Derive :Reasons (`$a \in F \vee a \in G$`)  :Conclusion` $a \in F \cup G$
  `:Method` $\cup$-Lemma`)`
  "Since $a \in F$ or $a \in G$, $a \in F \cup G$ by the $\cup$-Lemma."

- Explanatory MCAs comment on the steps of a derivation or give information about the structure of a derivation. This information is logically unnecessary, that is, omission leaves the derivation logically correct. However, inclusion of explanatory MCAs makes it much easier for the addressee to understand the derivations, since these comments keep him oriented. For example, an MCA of type `Case` introduces a case in a case analysis:

  `(Case :Number 1 :Hypothesis` $a \in F$`)`
  "Case 1: Let $a \in F$."

Interpersonal Communicative Acts. MCAs, which only convey information to the dialog partner without prompting any interaction, suffice to present mathematical facts and derivations in a monolog. To allow for dialogs we also need *interpersonal communicative acts (ICAs)*, which are employed for mixed-initiative, interpersonal communication. In our taxonomization we distinguish four classes of ICAs: questions, requests, acknowledgments and notifications. Note that the user never enters speech acts directly into the system. Instead, the user's utterances are interpreted by the analyzer and mapped into the corresponding speech acts.

ICAs are especially important to allow for clarification dialogs. If the system failed to successfully communicate a derivation, it starts a clarification dialog to detect the reason for the failure. Then, it can re-plan the previously failed part of the presentation and double-check that the user understood the derivation. We shall come back to this issue in Section 5.4.

5.3 Plan Operators

Operational knowledge concerning the presentation is encoded as productions. Each production either fulfills the current goal directly or splits it into subgoals. Let us assume that the following nodes are in the current proof:

Label	*Antecedent*	*Succedent*	*Justification*
P_1	Δ_1	$\vdash \varphi_1$	J_1
		$\vdots$	
P_n	Δ_n	$\vdash \varphi_n$	J_n
C	Γ	$\vdash \psi$	$R(P_1, \ldots, P_n)$

An example for a production is:

(P1) IF the current goal is to show $\Gamma \vdash \psi$
and R is the most abstract known rule justifying the current goal
and $\Delta_1 \vdash \varphi_1, \ldots, \Delta_n \vdash \varphi_n$ are known
THEN produce MCA
`(Derive :Reasons (`$\varphi_1, \ldots, \varphi_n$`) :Conclusion` ψ `:Method` R`)`
insert it in the discourse structure tree
and pop the current goal

By producing the MCA the current goal is fulfilled and can be popped from the goal stack. An example for a production decomposing the current goal into several subgoals is:

(P2) IF the current goal is to show $\Gamma \vdash \psi$
and R is the most abstract known rule justifying the current goal
and $\Phi = \{\varphi_i | \Delta_i \vdash \varphi_i$ is unknown for $1 \leq i \leq n\} \neq \emptyset$
THEN for each $\varphi_i \in \Phi$, push the goal to show $\Delta_i \vdash \varphi_i$

Note that the conditions of (P1) and (P2) only differ in the knowledge of the premises φ_i for rule R. (P2) introduces the subgoals to prove the unknown premises in Φ. As soon as those are derived, (P1) can apply and derive the conclusion. Hence, (P1) and (P2) in principle suffice to plan the presentation

of a proof starting from the conclusion and traversing the proof tree towards its hypotheses. However, certain proof situations call for a special treatment. Assume that the following nodes are in the current proof:

Label	*Antecedent*	*Succedent*	*Justification*
P_0	Γ	$\vdash \varphi_1 \vee \varphi_2$	J_0
H_1	H_1	$\vdash \varphi_1$	HYP
P_1	Γ, H_1	$\vdash \psi$	J_1
H_2	H_2	$\vdash \varphi_2$	HYP
P_2	Γ, H_2	$\vdash \psi$	J_2
C	Γ	$\vdash \psi$	CASE(P_0, P_1, P_2)

A specific production managing such a case analysis is the following:

(P3) IF the current goal is to show $\Gamma \vdash \psi$
and CASE is the most abstract known rule justifying the current goal
and $\Gamma \vdash \varphi_1 \vee \varphi_2$ is known
and $\Gamma, H_1 \vdash \psi$ and $\Gamma, H_2 \vdash \psi$ are unknown
THEN push the goals to show $\Gamma, H_1 \vdash \psi$ and $\Gamma, H_2 \vdash \psi$
and produce MCA
`(Case-Analysis :Goal` ψ `:Cases (`φ_1, φ_2`))`
and insert it in the discourse structure tree

This production introduces new subgoals and motivates them by producing the MCA.

Since more specific rules treat common communicative standards used in mathematical presentations, they are assigned lower costs, that is, $C_{(\mathrm{P3})} < C_{(\mathrm{P2})}$ (cf. Equation 1 in Section 4.2).

Moreover, it is supposed that each user knows all natural deduction (ND) rules, since ND rules are the least abstract possible logical rules in proofs. Hence, for each production p that is defined such that its goal is justified by an ND rule in the proof, the probability P_p that the application of p leads to the goal to explain that proof step equals one. Therefore, since CASE is such an ND rule, $P_{(\mathrm{P3})} = 1$.

Note that the productions ensure that only those inference rules are selected for the explanation that are known to the user.

5.4 User Interaction

The ability for user interaction is an important feature of explanation systems. Moore and Swartout presented a context-sensitive explanation facility for expert systems that, on the one hand, allows the user to ask follow-up questions and, on the other hand, actively seeks feedback from the user to determine whether the explanations are satisfactory [37]. Mooney and colleagues emphasized that the user must be able to interrupt the explanation system at any time [36].

In *P.rex*, the user can interact with the system at any time. When the system is idle – for example, after starting it or after completion of an explanation – it waits for the user to tell it the next task. During an explanation, *P.rex* checks

after each production cycle if the user wishes to interrupt the current explanation. Each interaction is analyzed by the analyzer and passed on to the dialog planner as a speech act, which is included in the current discourse structure tree.

We allow for three types of user interaction in *P.rex*: A *command* tells the system to fulfill a certain task, such as explaining a proof. An *interruption* interrupts the system to inform it that an explanation is not satisfactory or that the user wants to insert a different task. In clarification dialogs, finally, the user is prompted to give *answers* to questions that *P.rex* asks when it cannot identify a unique task to fulfill. In this paper, we shall concentrate on interruptions.

Interruptions. The user can interrupt *P.rex* anytime to enter a new command or to complain about the current explanation. The following speech acts are examples for messages that can be used to interrupt the system:

`(too-detailed :Conclusion` C`)`
: The explanation of the step leading to C is too detailed, that is, the step should be explained at a more abstract level.

`(too-difficult :Conclusion` C`)`
: The explanation of the step leading to C is too difficult, that is, the step should be explained in more detail.

The Reaction to `too-detailed`. When the user complains that the derivation of a conclusion C was too detailed, the dialog planner checks if there is a higher level of abstraction on which C can be shown. If so, the corresponding higher level inference rule is marked as known, so that it is available for future explanations. Then, the explanation of the derivation of C is re-planned. Otherwise, the dialog planner informs the user, that there is no higher level available for presentation. This reaction of the system is encoded in the following two productions:

(P4) IF the user message is
`(too-detailed :Conclusion` C`)`
and the inference rule R was used to justify C
and there is an inference rule R' justifying C that is more abstract than R
THEN mark R' as known by the user
and push the goal to show C

(P5) IF the user message is
`(too-detailed :Conclusion` C`)`
and the inference rule R was used to justify C
and there is no inference rule R' justifying C that is more abstract than R
THEN produce ICA
`(Most-abstract-available :Rule` R`)`
and insert it in the discourse structure tree

An example dialog where the user complained that the original explanation of a proof was too detailed shall be given in Example 1 in Section 6.

The Reaction to `too-difficult`. When the user complains that the derivation of a conclusion C was too difficult, the dialog planner enters a clarification dialog to find out which part of the explanation failed to remedy this failure. The control of the behavior of the dialog planner is displayed in Figure 2. Note that every arrow in the figure corresponds to a production such that we cannot give the productions here due to space restrictions.

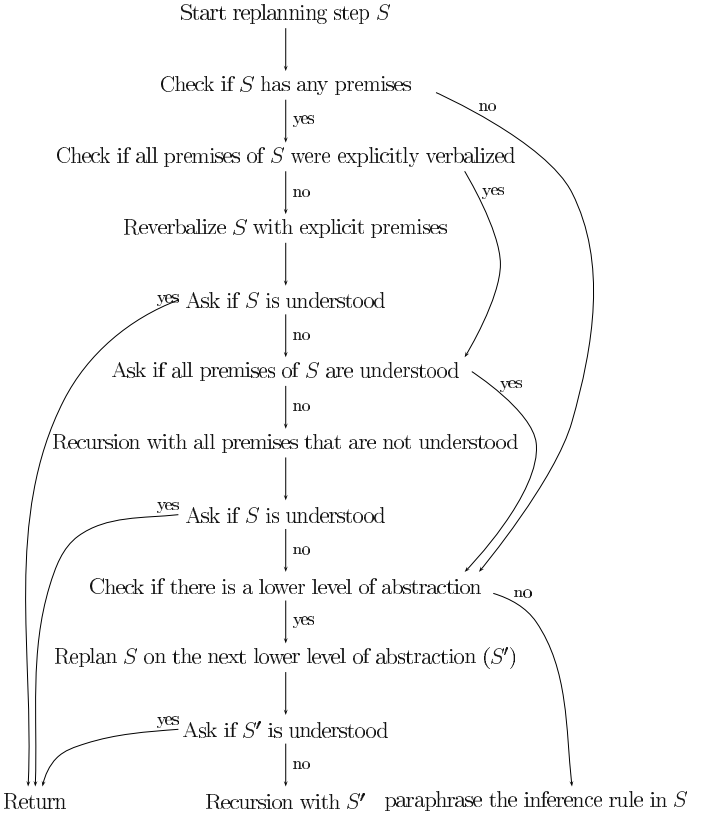


Fig. 2. The reaction of the dialog planner if a step S was too difficult.

To elucidate the diagram in Figure 2, an example dialog where the user complained that the original explanation of a proof was too difficult shall be given in Example 2 in the following section.

6 Example Dialogs

Let us examine more closely how *P.rex* plans the discourse with the help of two example dialogs. In both dialogs, *P.rex* explains the proof given in Figure 3. Note that this proof consists of two similarly proved parts with L_3 and L_7 as roots, respectively.

Label	*Antecedent*	*Succedent*	*Justification*
L_0		$\vdash a \in U \vee a \in V$	J_0
H_1	H_1	$\vdash a \in U$	HYP
L_1	H_1	$\vdash a \in U \cup V$	Def$\cup(H_1)$
H_2	H_2	$\vdash a \in V$	HYP
L_2	H_2	$\vdash a \in U \cup V$	Def$\cup(H_2)$
L_3		$\vdash a \in U \cup V$	$\cup$-Lemma(L_0)
			CASE(L_0, L_1, L_2)
L_4		$\vdash a \in F \vee a \in G$	J_4
H_5	H_5	$\vdash a \in F$	HYP
L_5	H_5	$\vdash a \in F \cup G$	Def$\cup(H_5)$
H_6	H_6	$\vdash a \in G$	HYP
L_6	H_6	$\vdash a \in F \cup G$	Def$\cup(H_6)$
L_7		$\vdash a \in F \cup G$	$\cup$-Lemma(L_4)
			CASE(L_4, L_5, L_6)

Fig. 3. A proof to be explained by *P.rex*.

Example 1. Let us consider the following situation:

- The current goal is to show the fact in L_3. The next goal on the stack is to show the fact in L_7.
- The rules HYP, CASE, and Def$\cup$ are known, the rule $\cup$-Lemma is unknown.
- The facts in L_0 and L_4 are known, the facts in H_1, L_1, H_2, L_2, H_5, L_5, H_6, and L_6 are unknown.

Since CASE is the most abstract known rule justifying the current goal, both decomposing productions (P2) and (P3) are applicable. Recall that the conflict resolution mechanism chooses the production with the highest utility E (cf. Equation 1 in Section 4.2). Since $P_{(\mathrm{P3})} = 1$ and $P_p \leq 1$ for all productions p, $P_{(\mathrm{P3})} \geq P_{(\mathrm{P2})}$. Since the application of (P2) or (P3) would serve the same goal, $G_{(\mathrm{P3})} = G_{(\mathrm{P2})}$. Since (P3) is more specific than (P2), $C_{(\mathrm{P3})} < C_{(\mathrm{P2})}$. Thus

$$E_{(\mathrm{P3})} = P_{(\mathrm{P3})}G_{(\mathrm{P3})} - C_{(\mathrm{P3})} > P_{(\mathrm{P2})}G_{(\mathrm{P2})} - C_{(\mathrm{P2})} = E_{(\mathrm{P2})}$$

Therefore, the dialog planner chooses (P3) for the explanation, thus producing the MCA

$$\texttt{(Case-Analysis :Goal } a \in U \cup V \quad \texttt{:Cases } (a \in U, a \in V)\texttt{)}$$

which can be realized as "To prove $a \in U \cup V$ let us consider the cases that $a \in U$ and $a \in V$," and then explains both cases.

Suppose now that the user interrupts the system throwing in that the presentation is too detailed. Then, the analyzer passes the speech act `(too-detailed :Conclusion` $a \in U \cup V$`)` to the dialog planner. Since the inference rule $\cup$-Lemma, which is more abstract than CASE, also justifies the conclusion, production (P4) applies. Hence, the inference rule $\cup$-Lemma is marked as known and the goal to show the fact in L_3 is again pushed onto the goal stack. Then, (P1) is the only applicable production. Since $\cup$-Lemma is more abstract than CASE and both are known, it is chosen to instantiate (P1). Hence, the dialog planner produces the MCA

(Derive :Reasons ($a \in U \vee a \in V$) :Conclusion $a \in U \cup V$:Method $\cup$-Lemma)

which can be verbalized as "Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the $\cup$-Lemma."

Since $\cup$-Lemma is now marked as known by the user, it can also be used for presentation in subsequent situations, for example, when L_7 is to be shown. The whole dialog takes place as follows:

***P.rex*:** In order to prove that $a \in U \cup V$ let us consider the following cases.
Case 1: Let $a \in U$. Then $a \in U \cup V$ by the definition of $\cup$.
Case 2: Let $a \in V$. That implies that $a \in U \cup V$ by the definition of $\cup$.
User: This derivation is too detailed.
***P.rex*:** Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the $\cup$-Lemma. Since $a \in F$ or $a \in G$, $a \in F \cup G$ by the $\cup$-Lemma.

To elucidate the behavior of the dialog planner as depicted in Figure 2, let us examine the following example:

Example 2. We now consider the following situation:

- The current goal is to show the fact in L_3. The next goal on the stack is to show the fact in L_7.
- the rules HYP, CASE, Def$\cup$, and $\cup$-Lemma are known,
- The facts in L_0 and L_4 are known, the facts in H_1, L_1, H_2, L_2, H_5, L_5, H_6, and L_6 are unknown.

This time, the only applicable production is (P1). Since $\cup$-Lemma is more abstract than CASE and both are known, it is chosen to instantiate (P1). Hence, the dialog planner produces the MCA

(Derive :Reasons ($a \in U \vee a \in V$) :Conclusion $a \in U \cup V$:Method $\cup$-Lemma)

which can be verbalized as "Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the $\cup$-Lemma."

Suppose now that the user points to this utterance and interrupts *P.rex* throwing in that this step was too difficult. The analyzer translates the user's interaction to the speech act (too-difficult :Conclusion $a \in U \cup V$). Now, the dialog planner enters the clarification dialog as displayed in Figure 2. Since all premises were explicitly mentioned, the system does not reverbalize the step, but asks whether all premises are understood, what the user affirms. Hence, the system checks whether there is a lower level of abstraction, at which the step can be presented. Since this is the case, *P.rex* replans the explanation of the step by marking the inference rule $\cup$-Lemma as unknown and pushing the goal to show the fact in L_3 onto the goal stack. Now, (P1) is not applicable, since $\cup$-Lemma is unknown, but (P2) and (P3) are applicable. Hence, the system proceeds as in the beginning of Example 1 and verbalizes the case analysis. Since $\cup$-Lemma is unknown, it is not used in subsequent situations either. Therefore, L_7 is also explained by a case analysis. The whole dialog takes place as follows:

***P.rex*:** Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the $\cup$-Lemma.
User: This step is too difficult.
***P.rex*:** Do you understand the premises?
User: Yes.
***P.rex*:** In order to prove that $a \in U \cup V$ let us consider the following cases.
Case 1: Let $a \in U$. That leads to $a \in U \cup V$ by the definition of $\cup$.
Case 2: Let $a \in V$. Then $a \in U \cup V$ by the definition of $\cup$.
Do you understand this step?
User: Yes.
***P.rex*:** In order to prove that $a \in F \cup G$ let us consider the following cases.
Case 1: Let $a \in F$. Therefore $a \in F \cup G$ by the definition of $\cup$.
Case 2: Let $a \in G$. Then $a \in F \cup G$ by the definition of $\cup$.

7 Conclusion

We reviewed research in proof presentation and presented the proof explanation system *P.rex*. Based on assumptions about the addressee's knowledge (e.g., which facts does he know, which definitions, lemmas, etc.), the dialog planner of *P.rex* chooses a degree of abstraction for each proof step to be explained. In reaction to the user's interactions, it enters clarification dialogs to revise its user model and to adapt the explanation. The architecture of the dialog planner can also be used to adapt content selection and explicitness reactively to the audience's needs. The rationale behind the architecture should prove to be useful for explanation systems in general.

However, in the current experimental stage, only a small set of user interactions is allowed. More elaborate interactions that call for more complex reactions are desirable. Therefore, empirical studies of teacher-student interactions in mathematics classes are necessary. Moreover, powerful natural language analysis and multi-modal generation is desirable.

P.rex is implemented in Allegro Common Lisp with CLOS and has been tested on dozens of input proofs. The implementation (currently Version 1.0) can be accessed via the *P.rex* homepage at `http://www.ags.uni-sb.de/~prex`.

Acknowledgments

Part of this project was funded by the Graduiertenkolleg Kognitionswissenschaft (doctoral program in cognitive Science) at Saarland University. I thank C. P. Wirth who read an earlier version of this work.

References

1. John R. Anderson and Christian Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum, 1998.
2. Peter B. Andrews. Transforming matings into natural deduction proofs. In *Proceedings of the 5th International Conference on Automated Deduction*, pages 281–292. Springer Verlag, 1980.

3. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In McCune [31], pages 252–255.
4. Bruno Buchberger. Natural language proofs in nested cells representation. In J. Siekmann, F. Pfenning, and X. Huang, editors, *Proceedings of the First International Workshop on Proof Transformation and Presentation*, pages 15–16, Schloss Dagstuhl, Germany, 1997.
5. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In Mark Stickel, editor, *Proceedings of the 10th Conference on Automated Deduction*, number 449 in LNCS, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag.
6. Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. In search of a reference architecture for NLG systems. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 77–85, Toulouse, France, 1999.
7. Alison Cawsey. Generating explanatory discourse. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, number 4 in Cognitive Science Series, pages 75–101. Academic Press, San Diego, CA, 1990.
8. Daniel Chester. The translation of formal proofs into English. *AI*, 7:178–216, 1976.
9. Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.
10. Yann Coscoy, Gilles Kahn, and Laurent Théry. Extracting text from proofs. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, number 902 in LNCS, pages 109–123. Springer Verlag, 1995.
11. B. I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ILF. In McCune [31], pages 57–60.
12. Andrew Edgar and Francis Jeffry Pelletier. Natural language explanation of natural deduction proofs. In *Proceedings of the 1st Conference of the Pacific Association for Computational Linguistics*, Vancouver, Canada, 1993. Centre for Systems Science, Simon Fraser University.
13. Michael Elhadad and Jacques Robin. Controlling content realization with functional unification grammars. In Robert Dale, Eduard Hovy, Dietmar Rösner, and Oliviero Stock, editors, *Aspects of Automated Natural Language Generation*, number 587 in LNAI, pages 89–104. Springer Verlag, 1992.
14. Amy Felty and Dale Miller. Proof explanation and revision. Technical Report MC-CIS-88-17, University of Pennsylvania, Philadelphia, PA, 1988.
15. Armin Fiedler. Mikroplanungstechniken zur Präsentation mathematischer Beweise. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 1996.
16. Gerhard Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, 39:176–210, 572–595, 1935.
17. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
18. Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and Eleventh Innovative Application of Artificial Intelligence Conference (IAAI-99)*, pages 277–284. AAAI Press, 1999.

19. Helmut Horacek. A model for adapting explanations to the user's likely inferences. *User Modeling and User-Adapted Interaction*, 7:1–55, 1997.
20. Helmut Horacek. Presenting proofs in a human-oriented way. In Harald Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI, pages 142–156. Springer Verlag, 1999.
21. Eduard H. Hovy. Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385, 1993.
22. Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
23. Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, pages 738–752, Nancy, France, 1994. Springer Verlag.
24. Xiaorong Huang and Armin Fiedler. Proof verbalization as an application of NLG. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 965–970, Nagoya, Japan, 1997. Morgan Kaufmann.
25. INLG. *Proceedings of the 7th International Workshop on Natural Language Generation*, Kennebunkport, ME, 1994.
26. Anne Kilger and Wolfgang Finkler. Incremental generation for real–time applications. Research Report RR-95-11, DFKI, Saarbrücken, Germany, July 1995.
27. Peter Kursawe. Transformation eines Resolutionsbeweises: Der erste Schritt auf dem Weg zum natürlichen Schließen. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1982.
28. Uri Leron. Structuring mathematical proofs. *The American Mathematical Monthly*, 90:174–185, 1983.
29. Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. PhD thesis, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.
30. William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. ISI Reprint Series ISI/RS-87-190, Univerisity of Southern California, Information Science Institute, Marina del Rey, CA, 1987.
31. William McCune, editor. *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, Townsville, Australia, 1997. Springer Verlag.
32. David D. McDonald. Natural language generation as a computational problem. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*. The M. I. T. Press, Cambridge, Massachusetts/London, 1984.
33. Andreas Meier. System description: TRAMP: Transformation of machine-found proofs into ND-proofs at the assertion level. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 460–464. Springer Verlag, 2000.
34. D. E. Meyer and D. E. Kieras. EPIC: A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, 104:3–65, 1997.
35. Dale Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, number 170 in LNCS, pages 375–303. Springer Verlag, 1984.
36. David J. Mooney, Sandra Carberry, and Kathleen McCoy. Capturing high-level structure of naturally occurring, extended explanations using bottom-up strategies. *Computational Intelligence*, 7:334–356, 1991.

37. Johanna D. Moore and William R. Swartout. A reactive approach to explanation: Taking the user's feedback into account. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence*, pages 3–48, Boston, MA, USA, 1991. Kluwer.
38. A. Newell. *Unified Theories of Cognition.* Havard University Press, Cambridge, MA, 1990.
39. Cécile Paris. The role of the user's domain knowledge in generation. *Computational Intelligence*, 7:71–93, 1991.
40. F. Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1987.
41. Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In INLG [25], pages 163–170.
42. Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, and Thomas Rist. Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, 63:387–427, 1993.
43. Marilyn A. Walker and Owen Rambow. The role of cognitive modeling in achieving communicative intentions. In INLG [25], pages 171–180.
44. Ingrid Zukerman and R. McConachy. Generating concise discourse that addresses a user's inferences. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1202–1207, Chambery, France, 1993. Morgan Kaufmann, San Mateo, CA.

Why Proof Planning for Maths Education and How?

Erica Melis

DFKI Saarbrücken, 66123 Saarbrücken, Germany
melis@dfki.de
http://www.ags.uni-sb.de/~melis

1 Introduction

Artificial Intelligence techniques have massively been applied for Intelligent Tutor systems (ITS), e.g., user modeling, error diagnosis, user adaptation, knowledge representation, and dialog techniques. In this paper, I will argue in favor of the application of another AI-technique in ITS, namely of proof planning, a methodology from automated theorem proving.

The attempt to employ proof planning for educational purposes is not self-evident. It requires an understanding of what mathematics education needs and what proof planning offers.

When I tried to convince teachers or mathematics professors of using proof planning for teaching and, in particular, of using our proof planner ΩMEGA, then most of them were very sceptical to begin with. Only slowly, when they had seen several examples, they finally could understand the general advantages and agree that using proof planning can be a goal for supporting learning in a constructivist direction[1]. What are the main reasons for the skepsis? *First* of all, handling current proof planners is not made at all for students who learn mathematics or even for teachers used to mathematical argumentation and this heavily inhibits the usability and discourages an actual usage. In particular, the user interface and the support for interactive theorem proving by feedback and other functionalities is still insufficient.

Secondly, to learn a proof planning language to handle yet another system requires an additional effort for teachers and for students.

Third and maybe most importantly, many teachers and even mathematicians are not used to think about teaching mathematical proofs and problem solving in a non-traditional way. They themselves have learned mathematics traditionally, by listening lectures, following a sequential description of proof steps in textbooks, and finally somehow constructing the representation of knowledge in their minds from many examples *implicitly* containing structures, methods, and heuristics used in the proof processes. Although modern didactical and cognitive theories and empirical evidences show that (deep) learning is little supported by the traditional style of teaching, good examples for a modern teaching of theorem

[1] 'constructivist' used in the epistemological sense of *constructing* knowledge in a user's mind [27] rather than the logical meaning of the word.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 364–378, 2005.

proving are rare. Geometrical proof is an exception because building hypotheses and visual clues are supported by dynamic geometry systems such as Geolog [13] and Cinderella [29].

Fourth, many mathematics professors do not believe that learning heureka steps can be supported by a system.

Given these heavyweight opinions, it takes a lot to convince teachers of the benefits a proof planning approach can have for learning. However, my experience shows that teachers and maths professors *can* be convinced by good examples from different mathematical areas that show how learning can be supported at points, where the teachers experienced bottlenecks for understanding and transfer of mathematical proofs in their courses. Once convinced, they even accept the additional effort. This has been similar for other mathematical tools such as calculators and computer algebra systems (CASs) for which one has to learn an input language. These systems are widely accepted by now even in schools.

Since good examples are crucial for convincing the potential users, this paper will not only discuss the benefits of the proof planning methodology for learning mathematics but also illustrated them with examples, in particular some I have been using successfully in my 'conviction activities'.

A warning before we go into medias res: although I shall advocate a support that makes explicit the heuristics and the common structure of a class of proofs, I do not want to replace beautiful singular heureka ideas and creative acts in mathematical problem solving and proving that we love so much and that create some of the beauty of mathematics that I admire personally. In the opposite, this paper investigates why common methods, structures, meta-level knowledge, and heuristics uncovered by the proof planning methodology can support the learning of standard textbook mathematics. In addition, I shall argue how an interactive system can improve the fun, motivation, self-responsibility, and self-guidance of learners – pedagogic goals that are generally pursued by any kind of advanced learning.

For Jörg I want to re-phrase and summarize: Machines can think mathematically, at least together with humans and therefore, AI-machines can support meat-machines in learning mathematics. And, in order to have a real impact, this requires an interdisciplinary effort from AI, Cognitive Science, and empirical pedagogy. Therefore, this article has an interdisciplinary flavor.

2 Why Is Improvement Necessary in the First Place?

To start with, let me reflect upon the current situation in mathematics teaching and upon the ways for improvement and new directions suggested by advanced educationalists and cognitive scientists. Subsequently, I will show *why* proof planning can contribute to some of the ingredients of a more appropriate learning and teaching; provide examples for *how* proof planning can be beneficial for learning mathematics; and finally discuss which improvements are necessary in order to make proof planning a methodology that real students and teachers will be able to use beneficially.

During the last decades, the mathematics pedagogy community recognized that students learn mathematics more effectively, if the traditional rote learning of formulas and procedures is supplemented with the possibility to explore a broad range of problems and problem situations [32, 18]. In particular, the international comparative study of mathematics teaching, TIMSS [4][2], has shown that teaching with an orientation towards active problem solving and an encouragement to find different solutions yields better learning results in the sense that the acquired knowledge is more readily available and applicable especially in new contexts. Moreover, several investigations show that a reflection about the problem solving activities and methods yields a meaningful learning [11, 2] that a training of meta-cognitive skills is a basis for meaningful and for lifelong learning [1].

There are at least four avenues to an improved mathematics learning: (1) the *active construction* of knowledge in the learner's mind, combined with a (2) systematic and structured teaching of *heuristic and systematic* knowledge for problem solving steps and *processes*, (3) the support of meta-cognitive reasoning, and (4) the appropriately structured presentation of problem solutions. The application of proof planning and its augmentation by a meta-cognitive cycle has the potential to contribute to all four ways of improvement as I shall show in the remainder of this article.

3 Active Learning

Active learning, as opposed to pure instruction, is one of the educational conclusions of the constructivist learning paradigm that goes back to Piaget [27] and Vygotsky [36]. The construction of knowledge in a student's mind requires to learn in a context, to be able to make mistakes and to learn from erroneous proof attempts, and to deeply understand the involved objects and their relationships. Moreover, the student has to experience "proving" as a process in which she engages during learning and "proof" as a (mere) product.

How can the learner's active participation be supported? As in other sciences, cognitive tools can provide a supporting machinery. The term *cognitive tool* was coined in [16] and denotes instruments explicitly supporting or representing cognitive processes and thus extending the limits of the human cognitive capacities, e.g., the working memory. When applied to learning, such tools can help, e.g., to remember, to practice, to hypothesize, to solve a problem. In particular, when

[2] The same seems to apply for the recent OECD study, PISA 2000, which looked at mathematical literacy of students measured in terms of student's capacity to

- recognize and interprete mathematical problems in everyday life
- translate these problems into a mathematical context
- use mathematical knowledge and procedures to solve problems
- interprete the results in terms of the original problem
- reflect on the applied methods and
- formulate and communicate the outcomes.

learning is difficult because it is too complex or because several things have to be done at the same time, these tools can help considerably. This is well-known for simulation tools [15], dynamic geometry systems, CASs, visualization by animations, and for tools that can impose a structure on a reasoning process [37, 34].

Let me summarize *why* proof planners and the integrated Computer Algebra Systems can support active learning of theorem proving and problem solving. They help the learner

- to *explore* a problem interactively and *directly experience* the result of a tentatively applied method or theorem. This is similar to what is possible in programming, where debugging is a most significant source of learning. The proof planner can *check the correctness* of the student's problem solving steps. It can help with feedback on where it is promising to explore and where dead ends are reached in a proof attempt;
- to *focus* on a particular subtask or skill in solving a problem. For example, the user of the system should not fail to calculate the limit of a function just because she cannot factorize polynomials but a CAS can perform this task. Similarly, the user should not fail to find the overall proof just because she cannot prove a minor subgoal;
- by presenting heuristics used, e.g. for intelligent backtracking, by explicating methods (see section 4), and also by providing an intelligently designed preselection and ordering of methods.
- to keep an overview of the proof attempt, e.g., list the not yet proved conjectures, present the resulting partial proof multi-modally.

4 Learning by Interactive Proof Planning

After the short answer on *why*, I want to explain *how* proof planning can help in learning.

A mere check of correctness could be performed by traditional automated theorem provers (ATPs), such as OTTER. However, there is more to proof planning which provides information that is essential for learning and for performing a mathematical proof: a sense of direction in searching for a proof plan, the theory-reasoning that helps to find instantiations for mathematical objects, and the use of mathematical methods which represent *mathematical* steps whose abstraction-level is usually different from the *logical* steps applied by an ATP.

An analysis of traditional textbook proofs gives rise to appreciate the information produced in proof planning. In such an analysis [17] Leron observed that the linear mathematical proofs occurring in most (text)books are just a *minimal* or even sub-minimal 'code' for transmitting mathematical knowledge that mature mathematicians are able to decode which is, however, missing important information, e.g. the proof idea. He found in his classes that many students are simply unable to decode those proofs and, therefore, their actions are reduced to meaningless manipulation of the 'code'.

The additional information that proof planning offers consists of methods and their explanation at different levels of abstraction, mathematical search heuristics, and the systematic construction of mathematical objects. This information is rarely taught and exercised in college-level mathematics, currently. We believe that teaching mathematical methods and proof know-how and know-when has to be introduced into mathematics teaching as an augmentation of the traditional teaching of axioms, theorems, and procedures. Indeed, first experiments suggest that instruction materials based on the description of proof planning *methods* yield a better subsequent problem solving performance than traditional (textbook-like) instruction material [23].

The idea to use proof planning in an educational context was also a reason for developing the Barnacle system [20], an extension of the proof planner CL^AM. It extended CL^AM mainly by an interface that is more accessible and tried to convey the rippling heuristic for proofs by mathematical induction.

4.1 Method Knowledge

In his foreword to *how to solve it* [28] Ian Stewart writes: n order to be able to select the relevant information, and make use of it, mathematicians spend a great deal of their time acquiring both, a broad background and a repertoire of more specific tricks.

In our research for proof planning, we tried to acquire those general as well as specific methods for a restricted area of mathematics [22]. Aside from such general 'background methods' as `Case-Split`, `UnwrapHyp`, and `TellCS` we designed the `ComplexEstimate` method whose idea is a modification and generalization of the limit heuristic in [5], as well as other estimation methods implicitly used in many proofs in calculus.

A simple variant of `ComplexEstimate` was used in the instruction material of the empirical research reported in [23] and mentioned above. An explanation of this method goes as follows:
`ComplexEstimate` proves the estimation of a complicated term $|b|$ by representing b as a linear combination $k * a\sigma + l$ [3] of a term a whose estimation is already known and by reducing the goal $|b| < \epsilon$ to simpler subgoals that contain $\mathbf{M}$ – a positive real number whose existence is postulated by `ComplexEstimate`. The subgoals are

1. $|k| \leq \mathbf{M}$,
2. $|a| < \epsilon/(2 * \mathbf{M})$,
3. $|l| < \epsilon/2$.

The rationale behind the reduction to the subgoals is contained in the proof schema of `ComplexEstimate` in which the subgoals, the Triangle Inequality, and monotony properties are employed to prove $|b| < \epsilon$.

For instance, in planning LIM+, at some point the goal
$|f(x) + g(x) - (l_1 + l_2)| < \epsilon$ is reduced by `ComplexEstimate` to

[3] where σ is a substitution.

(1) $|1| \leq \mathbf{M}$,
(2) $|f(x) - l_1| < \epsilon/(2*\mathbf{M})$,
(3) $|g(x) - l_2| < \epsilon/2$.

`ComplexEstimate` is not necessarily to be used in its full generality. It would make sense to vary the explanation (and maybe even the instantiation of the method in the actual proof process), if the coefficients k or l have the trivial values $k = 1$ or $l = 0$ because this makes the explanation easier to follow.

By making this method explicit the student can be supported in understanding the rationale behind the otherwise correct but senseless manipulation of the term b and in generalizing simple proofs that are special cases with $k = 1$ or $l = 0$ which might be solvable without knowing a systematic procedure. If equipped with this 'trick' and its rationale, it is easier for students to find proofs themselves as we have shown empirically [23].

4.2 Heuristic Knowledge

An analysis of Polya's famous heuristics for mathematical problem solving [28] shows that these heuristics are guidelines and very general strategies rather than rules. Dependent on the concrete situation, they have to be expanded to a set of concrete operations if possible [31]. Moreover, they comprise at least two levels of heuristics: problem solving (search) heuristics and meta-cognitive heuristics. In §6.1 we discuss how meta-cognitive heuristics can be used to support students in the future.

For problem solving Polya suggests to try the following complex heuristics (strategies):

- reformulate the problem
- find a special case to work on first; introduce an auxiliary variable or an auxiliary problem
- decompose the problem into subproblems

Some general problem solving heuristics such as 'decompose into subgoals', 're-represent goal', 'look for a common proof structure' are inherent in proof planning and methods already. Other proof heuristics are (or can be) represented by control rules in ΩMEGAs proof planner. Control rules have been used for all kinds of search control in ΩMEGA [21]. More recent ideas for guiding the search by control rules are due to Andreas Meier. They include, e.g., the instantiations of variables, and backtracking control. The backtracking control employs heuristic knowledge on when search branches are likely to fail and can enforce backtracking even if there are still applicable methods. It also can suggest, where to backtraqck to.

In this following, some more concrete search heuristics from proof planning that can be beneficially employed for learning, are discussed. In particular, rippling [14, 8] and the introduction of a case split.

Rippling is a search heuristic for systematic difference reduction. In the proof planner CL^AM it is performed based on an annotated logic calculus that handles

annotated terms and uses annotated matching[4]. More specifically, a skeleton-annotation indicates the commonalities between the induction hypothesis and the induction conclusion (the 'skeleton') and a context-annotation indicates the difference between the induction hypothesis and the induction conclusion (the 'context'). Rippling is essentially a difference-reducing rewriting technique that preserves the skeleton, i.e., the commonalities. Such a difference reduction works in particular for equational proofs and proofs by mathematical induction. An attempt to teach rippling to students has been made by Helen Lowe [19].

Another typical mathematical heuristic that is captured in control rules in the proof planner ΩMEGA suggests the introduction of a case split. It analyzes the overall proof attempt rather than a local search heuristic. It heuristically suggests a repair the proof attempt by introducing a case split. For the introduction of a case split, lets have a look at the proof of the following theorem[5].

Theorem 1. *A convergent sequence of real numbers is bounded.*

We zoom into the proof process, where – under the assumption $n > k_1$ – the inequality $|x_n - lim| < e_1$ has been employed to prove the boundedness, i.e., $|x_n| \leq B$ for $2 \cdot |lim| \leq B$.
In this situation, $n > k_1$ has still to be proved or, if this is difficult or impossible as here, a *heuristic* says that a case split $(n > k_1 \vee \neg n > k_1)$ has to be introduced and the subproof obtained so far under the condition $n > k_1$ provides the proof branch for the first case $n > k_1$. A student can learn this heuristic, if she is informed by the proof planner.

The second branch resulting from the case split requires to prove $|x_n| \leq B$ under the assumption $\neg n > k_1$. Since k_1 is a natural number, the new assumption can be split into the cases $n = 1 \ldots n = k_1$. The proof can then be completed with further restricting $|x_1| \leq B, \ldots |x_{k1}| \leq B$ in addition to $2 \cdot |lim| \leq B$, which finally yields $B = max(|x_1|, \ldots |x_{k1}|, 2 \cdot |lim|)$.

4.3 Support for Constructing Mathematical Objects

Many mathematical proofs require the construction of mathematical objects. For instance, in order to prove that a function $f(x)$ converges to l, if x converges to a, for each arbitrarily small positive real number ϵ a positive real number δ has to be constructed that meets certain requirements; or in order to show that there are infinitely many prime numbers, the proof by contradiction assumes that there exist only finitely many prime numbers whose biggest one is p, and for the contradiction a prime number is constructed which is bigger than p.

Typically, such a construction is a difficult part of the proof. Students may lack a proper technique or support to determine an object satisfying all the – maybe many and maybe somewhat hidden – requirements [17]. Traditional teaching does not provide support for this task. It just delivers correct instances

[4] Rippling can also be represented by control rules as shown in an unpublished BlueNote [25].
[5] Theorem 3.2.2 in [3].

out-of-the-blue rather than providing techniques for systematically constructing the objects. For a real understanding of a proof and meaningful learning it is essential to watch and understand which constraints for an object occur in the proof process and are *collected in a store*, and then to understand the *search* for an object that satisfies all the collected constraints.

ATPs use unification as a technique for constructing objects satisfying equality requirements, that is, domain-independent constraints. In addition, proof planning can integrate constraint solving for constructing objects with certain domain-specific constraints. By readably displaying the constraint store, the proof planner informs the student of the restrictions of an object and can support the creative act of constructing an object, as shown with the systematic restriction of B in the example of §4.2.

The following example[6] contains information about proof steps, meta-reasoning, and the constraint store (single entries of the constraint store are indicated by a box surrounding them).

Note that a computational presentation of the example would be more convincing since the process character of theorem proving can be presented and illustrated by introducing parts of the non-linear proof at different ends and hierarchical levels of the proof and can be accompanied by explanations of the methods, alternative search branches, and information about the constraint store. A computational presentation could offer useful features by a hierarchical and hypertext presentation, by a separate window that displays the constraint store, etc. The presented sequence of steps is not necessarily the sequence of their final appearance in a top-down proof. Only in the end, this can yield a final proof presentation (hiding some information again, if desired). The paper form of this article, however, prevents such a presentation of the proof process.

4.4 Example

Theorem 2. *If $f : I \mapsto \mathbf{R}$ has a derivative at $c \in I$, then f is continuous at c.*

1. By expanding the definitions, the proof assumption can be re-stated (forward application) as
   $\forall\epsilon_1(\epsilon_1 > 0 \rightarrow \exists\delta_1(\delta_1 > 0 \wedge \forall x_1(|x_1 - c| < \delta_1 \rightarrow x_1 \neq c \rightarrow |\frac{f(x_1)-f(c)}{x_1-c} - f'(c)| < \epsilon_1)))$,
2. the goal can be re-stated (backward application) as
   $\forall\epsilon(\epsilon > 0 \rightarrow \exists\delta(\delta > 0 \wedge \forall x(|x - c| < \delta \rightarrow |f(x) - f(c)| < \epsilon)))$.
3. The goal is reduced to $|f(x) - f(c)| < \epsilon$ yielding the assumptions $|x - c| < \delta$ and $\boxed{\epsilon > 0}$.
4. The subformula $|\frac{f(x_1)-f(c)}{x_1-c} - f'(c)| < \epsilon_1$ is extracted from the assumption which leaves $|x_1 - c| < \delta_1$ and the 'not so important' condition $x_1 \neq c$ as subgoals.
5. The subgoal $|x_1 - c| < \delta_1$ is proved by the assumption $|x - c| < \delta$ and this sends $\boxed{x = x_1}$ and $\boxed{\delta \leq \delta_1}$ to the constraint store.

[6] Theorem 6.1.2 in [3].

6. `ComplexEstimate` reduces the goal $|f(x)-f(c)| < \epsilon$ to the following subgoals for which the existence of a positive real number M is assumed
   (a) $|x - c| \leq M$
   (b) $|\frac{f(x)-f(c)}{x-c} - f'(c)| < \frac{\epsilon}{2 \cdot M}$
   (c) $|(x - c) \cdot f'(c)| < \frac{\epsilon}{2}$.
   The first subgoal can be proved by assuming $\boxed{\delta \leq M}$, the second can be proved by assuming $\boxed{\epsilon_1 \leq \frac{\epsilon}{2 \cdot M}}$, the third can be proved by a case split on $f'(c) = 0 \vee f'(c) \neq 0$ and assuming $\boxed{\delta \leq \frac{\epsilon}{2 \cdot |f'(c)|}}$ in the second case.
7. So far, the condition $x \neq c$ is not proved. Now it can be discovered by the user or by the proof planner that $x \neq c$ does not hold generally, hence cannot be proved. As in the example of §4.2, the meta-reasoning of the proof planner can suggest to introduce a `Case-Split` $(x \neq c \vee x = c)$ into the proof attempt and to take the subproof obtained so far – which needed $x \neq c$ – as the proof branch for the first case.
8. The second branch assuming $x = c$ has to be tackled separately. Its subproof is simple because in this case $0 = |f(x) - f(c)| < \epsilon$ holds.

5 Structured Presentation

For learning of mathematical proofs, Leron and others [17, 9] have shown that the traditional sequential presentation of proofs which has been used in most textbooks and courses is inappropriate for a real understanding and for learning proof skills. The more appropriate, structured (and multi-modal) presentation with more relevant information yields longer proofs because it makes explicit the complexity inherent in the proof anyway. For learning, however, it is useful to explicate the proof ideas, proof methods, and other information as empirical research and experience has shown.

In [24] we have shown how to construct a hierarchically structured proof presentation from a schematic verbalization of proof planning methods. An additional information dimension is delivered by the contraint solving in proof planning:

The construction of objects belongs to the information that is relevant and that can be made explicit in a hierarchical proof plan presentation. The constructions of objects is a frequent and often a tricky part of proofs. There are two ways of presenting such proofs: the 'classical' that simply defines the solution's object and shows that this object satisfies the constraints (typically introduced by "let.." or similar phrases) and a presentation that uses the collected constraints to search for an object (verbally this can be introduced by a phrase such as "suppose, we had already found a solution-object, what would it look like?"). The second presentation may not be as concise and 'elegant' as the first one, but it contains additional information that is important for meaningful learning that enables students to prove theorems on their own later on. In a multi-modal presentation, the display of the constraint store that contains the constraints on solution-objects can provide the relevant information.

6 Future Work: Polya and Friends

Although the user can already browse a partial proof plan during her proof attempt, the heuristic reasoning that can guide the proof attempt is not yet presented, even though it is essential for learning. This information will be communicated to the learner in a user-adaptive way.

In addition, we believe that a mixed-initiative system[7] is more appropriate what an average student can handle properly while concentrating on the mathematics rather than on handling the system. The following lists some of the features that will be targeted for a re-design of LOUI for a truly interactive and mixed-initiative use of proof planning.

- The input should be as similar to mathematical language as possible.
- The user has to be able to easily and in an intuitive way choose, backtrack, and tentatively try the application of a strategy (including backtracking etc.), a method, and of parameters etc.
- The user should easily be able to investigate the success or failure of a method and get feedback about repair opportunities.
- Different levels of detail of a proof plan should be inspectable.
- In the presentation of proof attempts, parts such as formulas, terms, and methods have to carry *semantics*[8].
- A proof verbalization needs to show the structure of the proof and the goal structure
- To support a user, highlighting of proof steps in focus (or hiding those not in focus) is needed in order to avoid cognitive overload by too high a complexity.

Moreover, the language for communicating with the user should approximate the commonly used mathematical (not so much logical) language. This way, the additional effort that is necessary to learn the language is not in vain but produces one of the skills belonging to the repertoire of mathematical work anyway. Examples of systems that use such languages are Mizar, ISETL [10], and a preliminary attempt by Schmidt [30].

In addition to the discussed changes of the current proof planner and its GUI, we shall augment the actual proof planning with radically new facilities described next.

6.1 Polya: Support of Meta-cognition

The actual theorem proving process is merely one part of the problem solving process in which exploration and producing conjectures plays a crucial role [7, 33]. This does not only apply for learning but also for the mathematician's work as, e.g., described by Buchberger's creativity spiral and in [6] that identifies the following activities in mathematical theorem proving

[7] See [12, 35] for an introduction to mixed-initiative (planning) systems.

[8] Carry semantics in the sense, that behind the presentation the semantics is represented and can be used for manipulations such as drag'n drop, for making it an assumption, etc.

- production of a conjecture (exploration of the problem situation)
- formulation of the statement according to conventions
- identification of appropriate arguments and link to the existing theory, exploration of (limits of) validity of the conjecture
- discovery or selection of proof methods and mathematical objects
- organization of the arguments into a coherent proof that is acceptable to current mathematical standards (for the communication of the proof).

Polya suggests a student to proceed with problem solving in a larger context that involves the phases of:

1. understanding the problem,
2. devising a plan for a solution,
3. executing (and debugging) the plan,
4. analyzing the success or failure of the plan.

In an explorative environment an additional start phase would comprise the invention of an hypothesis. For guidance, Polya suggests to ask the following questions

- what is known and what is unknown?
- do you remember a similar problem?
- find the connection between the known and the unknown
- carry out the plan and check each step
- is there an alternative way to solve the problem?
- examine the solution, can you use the result or the method for another problem?

Polya's heuristics for problem solving caught attention in the early years of AI and their implementation was targeted. In 1981, Allen Newell summarized why these attempts were not successful. He explained that Polya's heuristics are too general and each represents a whole class of heuristics [26]. Consequently, for an automated problem solving system it may be too difficult even today to realize such a heuristic guidance.

However, Polya's heuristics for meta-reasoning can be used for supporting interactive learning. Such a support for the student's meta-cognition is desirable because cognitive research [32] has indicated that a large part of what comprises competent problem solving behavior consists of the ability to guide, monitor, reflect, and assess the own problem solving process. It also indicates that students are poor at this, partly because these abilities and the introspection into the reasoning are rarely trained or discussed in traditional teaching.

A *meta-level cycle* as depicted in Figure 1 can help students who do not have those meta-cognitive skills yet. We plan to implement such a supportive cycle in a POLYA module. This module for supporting meta-cognitive activities will in some sense be similar to ThinkerTools [37], a tool for inquiry learning in experimental sciences. ThinkerTools provides the cycle: question, hypothesize, investigate, analyze, model, and evaluate. Of course, we expect quite a few changes that are mostly relevant for the mathematical problem solving as opposed to physics or biology experiments. Moreover, our meta-cognitive support will, be user-adaptive in the sense, that the student will be supported depending on her own abilities and needs.

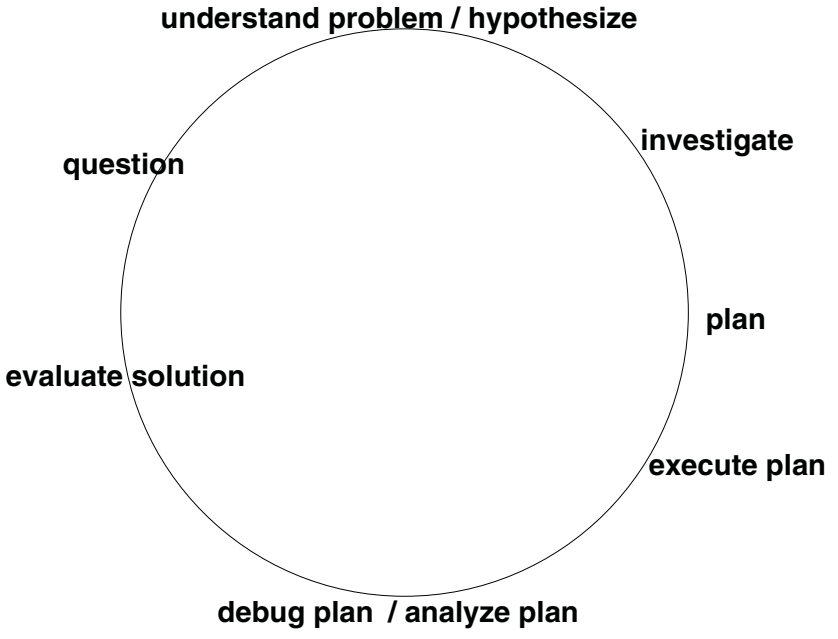


Fig. 1. The meta-level cycle.

6.2 Proof Planning by Programming

A use of proof planning as a cognitive tool that is geared towards constructive learning could be based on 'programming' of proof planning methods and proofs by students. For experience in this direction we refer to the work of the mathematicians Dubinsky and Leron who replace pure lecture by constructive, interactive methods involving programming and cooperative learning. They report an radically increased amount of meaningful learning for average students [18].

Different from other mathematical textbooks, their algebra course [10] is based on the constructivist belief that, before students can make sense of any presentation of abstract mathematics, they need to engage in mental activities which construct the base for future verbal explanations. Notions such as Co-sets and quotient groups become more meaningful to the students when definitions, examples, proofs etc. are closely related to activities than just presented in a lecture. Their course requires to actively *do* things (mainly programming) and to *discuss* them with the fellow students. It asks students to formulate a conjecture, test it, and try to give an explanation. This includes an exploration with a computer until the understanding of a topic is satisfying, as opposed to the traditionally practiced attitude to avoid mistakes and immediately correct faulty solutions.

In order to achieve this goal with proof planning, a relatively simple programming language for methods has to be designed that is close to mathematical language that students have to learn anyway.

7 Conclusion

We recognized proof planning as a methodology that is useful for a successful learning of mathematical proof. Therefore, we want to use a proof planner as a user-adaptive cognitive tool for learning mathematical problem solving, in particular mathematical theorem proving. The first step towards this goal has been the integration of the currently proof planner of the ΩMEGA system into the web-based learning environment ACTIVEMATH `http://www.activemath.org` for working through example proofs and for interactive proof exercises. In addition to this proof tool, the learner can benefit from other facilities of ACTIVEMATH such as an ontological overview of a mathematical area (concept map), a user-adaptive choice of examples and exercises, suggestion mechanisms, etc.

We described which support already exists and how the proof planner and its GUI will be modified and enhanced in order to ease its use and to support all phases of mathematical problem solving/proving as they occur in learning and in mathematicians' work. This 'future work' description deals with some of the essential objectives of the Mippa project and invites other researchers to collaborate for these objectives.

References

1. Gutachten zur Vorbereitung des Programms "Steigerung der Effizienz des mathematischen-naturwissenschaftlichen Unterrichts. Bund-Länder-Kommission, 1997. Materialien zur Bildungsplanung und zur Forschungsförderung.
2. V. Aleven, K.R. Koedinger, and K. Cross. Tutoring answer explanation fosters learning with understanding. In S.P. Lajoie and M. Vivet, editors, *Artificial Intelligence in Education*, pages 199–206. IOS Press, 1999.
3. R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis*. John Wiley& Sons, New York, 1982.
4. J. Baumert, R. Lehmann, M. Lehrke, B. Schmitz, M. Clausen, I. Hosenfeld, O. Köller, and J. Neubrand. *Mathematisch-naturwissenschaftlicher Unterricht im internationalen Vergleich*. Leske und Budrich, 1997.
5. W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3(1):27–60, 1972.
6. P. Boero. Argumentation and mathematical proof: A complex, productive, unavoidable relationship in mathematics and mathematics education. Proof Newsletter, 1999.
7. P. Boero, R. Garutti, and M.A. Mariotti. Some dynamic mental processes underlying producing and proving conjectures. In *Proceedings of PME-XX*, volume 2, pages 121–128, 1996.
8. A. Bundy, F. van Harmelen, A. Ireland, and A. Smaill. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.
9. R. Catrambone and K.J. Holyoak. Learning subgoals and methods for solving probability problems. *Memory and Cognition*, 18(6):593–603, 1990.
10. E. Dubinsky and U. Leron. *Learning Abstract Algebra with ISETL*. Springer-Verlag, 1993.

11. M.T.H. Chi et al. Self-explanation: How students study and use examples in learning to solve problems. *Cognitive Science*, 15:145–182, 1989.
12. G. Ferguson, J. Allen, and B. Miller. Trains-95: Towards a mixed-initiative planning assistant. In B. Drabble, editor, *Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77, 1996.
13. G. Holland. *Geolog-Win*. Dümmler, 1996.
14. D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *Proceedings of 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.
15. W:R: Joolingen and T. Jong. Design and implementation of simulation-based discovery environments: the SMISLE solution. *Journal of ARtificial Intelligence and Education*, 7:253–277, 1996.
16. S. Lajoie and S. Derry, editors. *Computers as Cognitive Tools*. Erlbaum, Hillsdale, NJ, 1993.
17. U. Leron. Heuristic presentations: the role of structuring. *For the Learning of Mathematics*, 5(3):7–13, 1985.
18. U. Leron and E. Dubinsky. An abstract algebra story. *American Mathematical Monthly*, 102(3):227–242, March 1995.
19. H. Lowe, A. Bundy, and D. McLean. The use of proof planning for co-operative theorem proving. Research Paper 745, Department of AI, 1995.
20. H. Lowe and D. Duncan. Xbarnacle: Making theorem provers more accessible. In W. McCune, editor, *Proceedings of the Fourteenth Conference on Automated Deduction (CADE-14)*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 404–408. Springer, 1997.
21. E. Melis. AI-techniques in proof planning. In *European Conference on Artificial Intelligence*, pages 494–498, Brighton, 1998. Kluwer.
22. E. Melis. The "limit" domain. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence in Planning Systems*, pages 199–206, 1998.
23. E. Melis, Ch. Glasmacher, C. Ullrich, and P. Gerjets. Automated proof planning for instructional design. In *Annual Conference of the Cognitive Science Society*, pages 633–638, 2001.
24. E. Melis and U. Leron. A proof presentation suitable for teaching proofs. In S.P. Lajoie and M. Vivet, editors, *9th International Conference on Artificial Intelligence in Education*, pages 483–490, Le Mans, 1999. IOS Press.
25. Erica Melis and Julian Richardson. Separation of control and logic in rippling and unwraphyp, 1998.
26. A. Newell. The Heuristic of George Polya and its Relation to Artificial Intelligence. Technical Report CMU-CS-81-133, Carnegie-Mellon-University, Dept. of Computer Science, Pittsburgh, Pennsylvania, U.S.A., 1981.
27. J. Piaget. *Equilibration of Cognitive Structures*. Viking, New York, 1977.
28. G. Polya. *How to Solve it*. Princeton University Press, Princeton, 1945.
29. J. Richter-Gebert and U.H. Kortenkamp. *The Interacitive Geometry Software Cinderella*. Springer-Verlag, 1999.
30. P. Schmidt. Preparing oral examinations of mathematical domains with the help of a knowledge-based dialogue system. In *Proceedings of Ed-Media*, 2001.
31. A.H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, New York, 1985.
32. A.H. Schoenfeld. *Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics*, chapter 15. McMillan Publ.Company, New York, 1992.

33. M. Simon. Beyond inductive and deductive reasoning: The search for a sense of knowing. *Educational Studies in Mathematics*, 30:197–210, 1996.
34. D. Suthers, A. Weiner, J. Connely, and M. Paolucci. Belvedere: Engaging students in critical discussion of science and public policy issues. In *7th World Conference on Artificial Intelligence in Education, AIED-95*, pages 266–273, 1995.
35. Manuela M. Veloso. Towards mixed-initiative rationale-supported planning. In A. Tate, editor, *Advanced Planning Technology*, pages 277–282. AAAI Press, May 1996.
36. L. Vygotsky. *Thought and Language*. MIT press, Cambridge, MA, 1986. originally published 1962.
37. B.Y. White and J.R. Frederiksen. *Inquiry, Modeling, and Metacognition: Making Science Accessible to all Students*. Lawrence Erlbaum, 1998.

Towards MultiMedia Instruction in Safe and Secure Systems

Bernd Krieg-Brückner

Bremen Institute of Safe and Secure Systems, Universität Bremen,
Postfach 330440, D-28334 Bremen
bkb@Informatik.Uni-Bremen.DE

Abstract. The aim of the MMiSS project is the construction of a multimedia Internet-based adaptive educational system. Its content will initially cover a whole curriculum in the area of Safe and Secure Systems. Traditional teaching materials (slides, handouts, annotated course material, assignments and so on) are to be converted into a new hypermedia format, integrated with tool interactions for formally developing correct software; they will be suitable for learning on campus and distance learning, as well as interactive, supervised, or co-operative self-study. Coherence and consistency are especially emphasised, through extensive semantic linking of teaching elements, and through a process model borrowed from the theory of formal software development, enlarging the knowledge base with the help of version and configuration management, to ensure "sustainable development", i.e. continuous long-term usability of the contents.

1 Aims

The aim of the MMiSS project (*MultiMedia instruction in Safe and Secure Systems*), which is supported by the German Ministry for Research and Education, bmb+f, in its programme *"New Media in Education"* from 2001 to 2003, is to set up a multimedia Internet-based adaptive educational system, covering the area of Safe and Secure Systems. Thanks to a consistent integration of hypermedia course materials and formal programming tools, teaching in this area will attain a level hitherto impossible in this form. The system will be as suitable for learning on campus and for distance-learning with its associated management of assignments, as it is for interactive, supervised, or co-operative self-study.

The system is to be introduced step by step, over the duration of the project, into the normal teaching activities of the project partners: Universität Bremen (Krieg-Brückner, Eckert [now at Darmstadt], Gogolla, Kreowski, Lüth, Peleska, Roggenbach, Schlingloff [now at HU Berlin], Schröder, Shi et al.), FernUniversität (Distance-University) Hagen (Poetzsch-Heffter, Kraemer et al.), Universität Freiburg (Basin, Wolff et al.), Ludwig-Maximilians-Universität München (Wirsing, Kroeger, Merz et al.), and Universität des Saarlandes (Hutter, Melis, Siekmann, Stephan et al.). However, as the "Open-Source" model is to be used and teaching materials and tools are to be made freely available, a much greater

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 379–395, 2005.

national and international take-up is to be expected. To assist this, a MMiSS Forum is to be founded with German, international, and industrial members, to evaluate the emerging curriculum and assist its development and distribution. The Advisory Board shall advise the project from a scientific as well as an industrial perspective, with a view to future applications.

The area of *"Safe and Secure Systems"* has in the last few years become increasingly important. Software is increasingly used to control safety-critical embedded systems, in aeroplanes, spaceships, trains and cars, and electronic trading over the Internet, with its associated security risks, is rapidly expanding; all this requires qualitatively and quantitatively better training. To go with the planned deployment at universities, a number of well-known German companies have already expressed, through the various industrial contacts of the project partners, an interest in measures for further in-house training.

At the core of the system is the hypermedial adaptation of a series of classes or lectures on the development of Safe and Secure Systems. The lecturers should be able to store various sorts of course material, such as overheads, commentary, bibliographies, books, lecture notes, exercises, animations and so on, and retrieve them again for use in teaching. The system provides a formal framework for the integration of teaching materials based on a *semantic structure (ontology)* and enables fast directed access to individual teaching elements. An initial collection of teaching materials is already available and should be further hypermedially developed as part of the project. It covers the use of *formal* methods in the development of (provably) *correct* software. Highlights include data modelling using algebraic specifications; modelling of distributed reactive systems; handling of real-time with discrete events; and the development of hybrid systems with continuous technical processes, so-called *safety-critical systems*. The curriculum also covers informal aspects of modelling, and introduces into the management of complex developments and *security* .

The system will also contain a meta-database, containing methodological, ontological and paedagogical knowledge about the contents. The teaching materials should, where possible, be available in several different variants. It should be left to the teachers, or the students, to choose between variants, according to the educational or application context. For example reactive systems could be modelled with either process algebras or Petri-nets.

An important educational aspect is to teach about the possibilities and limits of formal tools. Tools for formal software development should be integrated in the system, to illustrate and intensify the contents to be taught. Thus students doing assignments can use the system to test their own solutions, while gathering experience with non-trivial formal tools. The integration of didactic aspects with formal methods constitutes a new quality of teaching. It will become possible, for the first time in formal methods, both to present a variety of formal tools as a subject for teaching, and to use them as a new medium. Thus an algorithm can for example be simultaneously developed, visualised, and verified.

The goal of applying the new system in as many universities and companies as possible, and the fact that the area of Safe and Secure Systems will continue

to develop in future, requires the highest level of *flexibility, extensibility and reusability* of the content. It should be possible to incrementally extend or adapt content and meta-data, to suit the teacher's individual requirements, and to keep them up-to-date.

As the individual parts of the curriculum rely on each other, there is a network of *semantic dependencies*, which the system should be able to administer; thus it must at the least handle version- and configuration-management. The ontology additionally allows better support for orientation and navigation within the content. It should also form the basis for adaptation to the user, for example by learning from exercises which concepts the students have understood, and adapting future assignments accordingly.

The formalisation of semantic dependencies means that the system can help *maintain the consistency* of the content. Definitions must be coordinated to suit each other; the removal or adaptation of part of the material may force the removal or adaptation of all dependent concepts. In formal software development, a similar problem has to be solved: there are also semantic dependencies between different parts of a development, for example between specification and implementation. Some of the project partners have already developed techniques for the administration of such dependencies as things change, and implemented them in development tools. Here we perceive an important *synergy* between expertise in formal software development – and support tools – and the demands of long-term sustainable administration of consistent multimedia materials in an efficient and productive educational system.

2 Sustainable Development of the Content

The problem of "sustainable development", i.e. how to continuously develop and maintain multimedia educational content, is to a large extent unsolved:

> To help realise complex systems, tools are needed to support the development process from initial design to maintenance. ... Tools are also becoming ever more important, to co-ordinate team-work and guarantee consistency during development and beyond. The existing tools have substantial deficiencies. Support is especially lacking for the early stages of development, as is a suitable methodological framework. ... The commercially available systems ... offer a wide range of possibilities, though the ... results are hardly understandable or maintainable. ... Generation and reuse of previously developed components in a new project is as good as not supported. A further grave problem is the deficient or inadequate support of quality control during development. ... This leads later to maintenance problems, as with current software systems. ... The story is similar with the development of educational systems, for which the development methods and tools in use today correspond to the state of the art 20 or 30 years ago. [35]

In the MMiSS project, the elimination of these deficiencies is a priority.

In teaching practice there is a series of specific problems in the area of formal methods, for example:

- the adequate communication of abstract mathematical concepts;
- the communication of course material which has a complex structure, is often presented in a non-uniform way, and develops dynamically;
- the integration of practical aspects, such as process models [15] and tools.

Teaching Material for Safe and Secure Systems. The area of Formal Methods, the basis for the content to be developed during the project, is established in academia and on the threshold of coming into the industrial mainstream. It is differentiated into a variety of competing alternatives and orthogonal, potentially complementary approaches. Like many mathematical theories, the different methods have a complex internal structure. Due partly to the rapid development of the last 15 years, there also remains a certain lack of uniformity in the presentation of the theoretical foundations, with corresponding consequences for teaching. For the content, the project will address standardisation as a priority in the short term, work out approaches for integration, and devise principles for the comparison and presentation of alternatives. We will now sketch previous work of the project partners in this direction, reflected in the comprehensive teaching material that is already available.

Several project members were involved in the bmb+f project KORSO (*"Correct Software"* [11]), which laid the foundations for this co-operation; of these, several have been working for many years on Algebraic Specification [10, 12, 4] in a rather closed-knit international community, funded for many years by the EU as ESPRIT WG COMPASS [22] and migrating eventually into the IFIP WG 1.3 (Foundations of System Specification) The *Common Framework Initiative for Algebraic Specification and Development (CoFI)* [27, 13] of IFIP WG 1.3, which originated from COMPASS, aimed at the development of an internationally standardised family of specification languages [2, 14, 34, 28]. CASL, the core language of this family, shall be a standard in the project for all teaching content concerned with mathematical foundations, algebraic specification and data modelling; it is well-supported by tools [26, 25], and its development methodology receives increased attention [30–32]; its link to the functional programming language Haskell as a target is well under way [34].

Work at Ludwig-Maximilians-Universität München and at Universität Bremen will be important for integrating the content with which we are concerned here into the whole subject of software development. This work aims to build bridges between Formal Methods, and those informal methods and languages which are in practice now a de facto standard, such as UML and Java. Techniques for specification and verification of object-oriented programs have been developed at FernUniversität Hagen.

The situation is less unified in the area of the formal treatment of concurrent reactive distributive systems, up to and including (hard) realtime and hybrid systems. One possibility is the hierarchy of languages established at Universität Bremen, which stretches from the widely-used language CSP [18, 33, 37] via Timed-CSP [24] and HybridCSP [1] to hybrid automata and the duration

calculus, to be combined with CASL [29]. The foundations of temporal logic have been analysed at Ludwig-Maximilians-Universität München and used there and at Universität Bremen in the teaching of model-checking. At Universität Freiburg, it was demonstrated how decidable monadic second-order logics can be used to model and to reason about such systems (e.g. [8]). Within courses on "Software Techniques", "Testing" and "Proofs and Modelling", a wide range of content on the subject of "Integrating Formal Methods into the Software Design Process" (cf. e.g. [9]) has been created.

The proof system INKA, the VSE-method and its derivatives [5–7], developed at Universität des Saarlandes (DFKI), combine development methods for abstract datatypes with temporal logic. Educationally, VSE has principally been used in industrial seminars. During the adaption of content in this area, the aim is to further work out existing approaches for integration, and to delimit and classify alternative methods as they apply to particular applications.

Coherent and Consistent Teaching Materials. One problem with the development of a national curriculum on "Safe and Secure Systems" is the wide variety of different and partly competing approaches. Here a unifying approach, at an international level, presents itself, via the specification language CASL. The proposed system may be instrumental in spreading such standardising approaches, and, via New Media education, create an new identity in the field. In an analogous way, the restriction, at first, to a few established and well-supported languages and tools for reactive and hybrid systems should lead to coherence in the curriculum. Initial experience from industrial training has been very promising. Up to now the preparation of content has been done locally from the specialised viewpoint of individual teachers; the comprehensive consistent integration of content will overcome this, and so contribute to a uniform understanding of the whole area throughout Germany, and, as a perspective, beyond.

Semantic Linking of the Content. Many beginners find the subject matter very complicated at first, because of the many dependencies between the various fundamental formalisms. It is tiresome and time-consuming to communicate conceptual dependencies and conceptual analogies to students. The proposed system can play a decisive role, by providing a hierarchy of concepts (an ontology) throughout the whole material, making the complexity manageable for students as well as teachers.

The author of content will be able to assemble teaching units from a structured system of individual modules and elements, by using the structural and semantic relations explicit in the representation, such as pre-conditions, cross-references, related units and alternatives. Thus content can be prepared by different people with different goals and requirements, but together and as part of the same repository, possibly in different variants and views.

Students will also be able to side-step a prescribed order of presentation for course content, navigate by themselves and make use of related materials as their own needs dictate. Rapid access to semantically related concepts and theories will significantly help users in forming an overall picture.

It is expected that this project will influence other areas, such as Mathematics or other areas in Computing Science, in the short run, and so lead to a persistent improvement in the teaching methodology of interrelated theories.

Reusability and Extensibility. One major problem with the preparation of teaching materials of any kind is that the adaptations necessary for each teacher and each year make reuse of older material almost impossible. It is often necessary to completely restructure a course to integrate new developments and results into hand-outs and overheads. Thus a major goal of the proposed system is to guarantee users the highest degree of flexibility, extensibility and reusability of the materials stored within it. It should be easy for teachers to combine different materials, even from different authors, into a whole, and for students to use alternative material. The planned mechanisms to support this, such as version and configuration-management, consistency-preservation, and a tool-supported development methodology, will also be available for other similar systems, and are expected to substantially improve the long-term development of coherent and consistent teaching content in the New Media.

Extensible Knowledge-Base. The speed at which knowledge develops is a special problem in the areas of Computing Science and *Safe and Secure Systems*. It is imperative to be able to continuously extend and modify the stored knowledge. This leads to consistency problems, in particular for multimedia presentations. For example, it must be clearly specified whether a cross-reference (hyperlink) refers to the newest version (whatever it is), or to some specific older version; these could be lost or outdated because of modifications to the referenced content. This is especially important in the context of Formal Methods: a referenced definition must fit into the application context which may not necessarily be compatible. For example, the name of the term defined by the other author may be different, or, worse, the other author may use the same name for an entity that has a subtly different semantics. The system to be constructed shall solve this by keeping track of semantically different entities, independently of their apparent name in a specific context, and by storing additional metadata in the knowledge-base. Semantics and functionality of knowledge are to be separated from representation.

There is also the problem of granularity. An element in the content may be an entire lecture, a particular topic, a overhead (or something structurally equivalent to it) on a subtopic, or indeed a single definition or theorem. This problem is to be solved by structuring the teaching materials into a semantically-based hierarchy, reached via, and defined in their granularity by, the ontology.

Semantic Relations in the Development Graph. Figure 1 shows, as an example of the structure, a section of a document containing mathematical definitions, theorems and proofs (with connecting texts); a similar situation arises with overhead transparencies for lectures. Texts, on the other hand, contain embedded *formal* components (theorems, formulas, proof-scripts) which can of themselves be processed by corresponding systems. A textual nesting (the *"is*

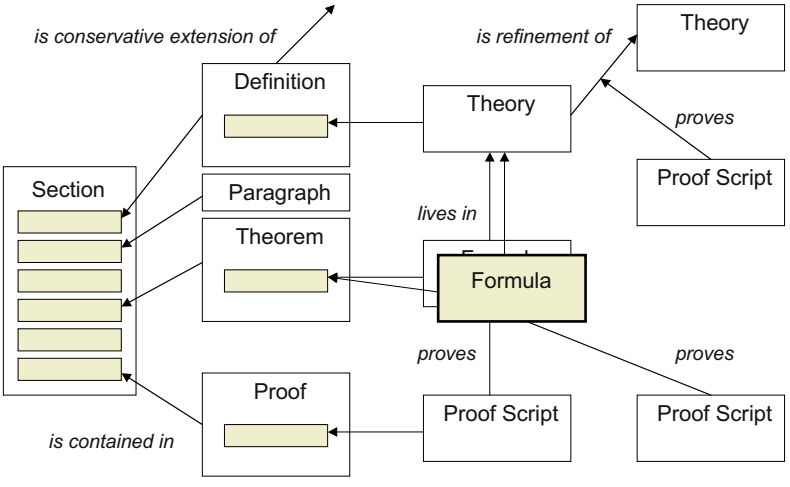


Fig. 1. Document Structure and Development Graph.

contained in" relation) yields at first a tree structure. This is extended by *semantic relations*, defined explictly by the user or implicitly by the system. For formal components this structure is evident; a formula representing a theorem to be proved *lives in* a theory; a proof-script that proves this theorem within a proof-system is subordinate to this theorem, or in general to a relation containing a proof obligation (*proves*). In the course of development, alternatives arise, for example an alternative proof in the example; this, like earlier versions, must be preserved so that it is possible to return to it. Thus there is in general a *Development Graph* containing one or more *formal* development graphs as subgraphs.

Development Methodology. Semantic approaches from software engineering and, in particular, Formal Methods, can cure the hitherto unsolved problem of how to develop sustainable multimedia teaching content. Here the development methodology of *(stepwise) refinement* is already known (compare with *"is refinement of"* in Figure 1); this could for example be applied to working out the materials with more precision or in more detail. The important point is that a reference to this activity of refinement is preserved. Another concept borrowed from Formal Methods is the so-called *conservative extension*, which preserves the original content so that a reference to the extended version remains valid for the original meaning. An example of this is a theory whose axioms are kept, but which is extended by further properties derived as theorems (compare with *"is conservative extension of"* in Figure 1). This concept has a well defined verifiable semantics, which naturally cannot be guaranteed for textual, or other multimedia, content. Thus consistency must be preserved through discipline among the developers, rather than formal proofs. In any case we can, as a *semantic* relation, distinguish conservative extension from a *real* change: the latter forces all dependent content to be reworked (this can be automatically recognised and communicated to the authors), while a conservative extension does not do.

Version and Configuration Management. An important dimension for Development Graphs is that of versions and their administration. Filtered *views* (realised by the graph-visualisation system daVinci [17, 16]) should help the user. Usually only the *current version* is of interest and all earlier ones are not shown; however, an option should make alternatives to a version visible. It should be possible to select between *variants*, such as the language used (for example "British English" or "German"), the formalism (for example "CASL" or "CSP"), or the level of detail (for example "Lecture Notes", i.e. overheads augmented by comments and explanations, suitable for study after presentation in class), and so on. The notion of a *consistent configuration* is important here; all objects related to a particular selected object should belong to the same version, or at least to a semantically compatible one. The document actually displayed (a subgraph) should in a well-defined sense be *complete*; thus when for example the formalism"CASL" is chosen as a variant, examples should generally be available in the formalism"CASL", and similarly when a particular level of detail is chosen. It is then possible to freeze a configuration as a publicly-accessible *edition*. All these functions, especially verifying consistency and completeness, should be supported by the system.

Scenarios and Roles. The knowledge-base is to be read, enriched or extended by different groups of people, according to the educational context. As sketched above, we recommend a semantics-driven process model for the development of multimedia educational content in which the different scenarios and rôles of those involved are differentiated:

- The *author* provides the initial groundwork (such as overheads), supplements it as required with animations or tool demonstrations, reacts to feedback from students and teachers, adds commentary, and expands it to create hypermedially-related teaching material, such as lecture notes or courses for distance learning (whether tutored or not).
- The *teacher* uses the teaching material stored in the knowledge base for teaching on campus, adapts it to his or her specific requirements (by selection or extension), and stores it back in the system.
- The *tutor* also uses the existing material, but in a different teaching situation; s/he compiles explanatory commentary, answers questions, puts frequently asked questions and answers together, sets and corrects assignments, and so on.
- The *student* uses the (prepared) teaching materials for a review after class; for self-study of the fundamentals or of additional background material; for assignments; and so on. The system helps to navigate or leads through the materials. It also contains (meta) information to support the selection of material according to the student's progress.
- The *system developer* extends the underlying system, incorporating existing and recently developed tools, especially for Safe and Secure Systems, into the development system.
- The *tool developer* works on particular tools, particularly for authoring.

– The *administrator* manages the system and cares especially for version and configuration management both of the content and the system itself. S/he moderates editions and arranges their distribution, including ones for particular user groups (such as authors, teachers, students attending particular courses); creates user groups and manages them; supports the distribution and installation of system versions.

2.1 A MultiMedia Platform for Educational Content

To support didactically worthwhile multimedia training that is genuinely interactive and cognitively adequate, powerful support systems must be developed, particularly in formal areas (Mathematics, Formal Methods); they should be adaptable to the user. Up to now, there are few such systems; the Springer-Verlag's interactive textbooks represent the first steps in this direction. These textbooks all belong to the first generation, which has no KI-methods such as user-modelling and diagnosis, learning, knowledge-representation, distributed architecture (multiple agents), and which makes little use of results from Cognitive Psychology or the theory of Education (cf. also the recent efforts of ActiveMath [23] at Universität des Saarlandes).

The ability to structure theories hierarchically, compare alternative approaches, combine complementary approaches usefully, and abstract away differences in presentation is, sometimes with difficulty, to be found among experts, but hardly among students. Currently, teaching of formal methods is predominantly characterised by being based on (or restricted to) the "local theory environment". Methodologically, classical methods such as lectures (with little or no interaction) and exercise sessions predominate. A comprehensive inter-relation of content is therefore impossible without co-operation and system support during creation. It is often a problem just to combine two, in principle complementary, but in detail differently constructed, textbooks. There is no support for recombination and further development of content, a particular problem given the rapid development of the subject area.

Structure of the Support System. The support system should have an open architecture and accommodate various user models. This requires, in the simplest case, a coarse static classification by user category or rôle in the learning process, such as Diploma or Master's student, student still learning the basics, (external) student in further training, or industrial user; such a classification should be universally introduced. It is also intended to take advantage of the opportunities available for educational systems which dynamically adapt to the progress of the user. Thus the proposed system is divided into components, which are presented to the user in a view depending on the scenario:

– The *authoring system* contains various tools for the preparation, semantic linking and extension of content.
– The *teaching system* serves primarily to support teaching on campus, but is also suitable for tutored distance-learning.
– The *learning system* contains materials for students and supports various learning situations.

- The *development system* for Safe and Secure Systems permits the integrated use of tools for demonstrations and exercises.
- The *assignment system* manages assignments; solutions to exercises are dispatched for correction, the corrections administrated, and the corrected solutions returned to the students.

A detailed architecture will be designed on the basis of the process model and the methodology, which in turn serve as basis for the implementation of the system. In particular the architecture will specify the individual components and how they communicate. It will also be necessary to consider how development tools fit into the system, the various supporting formats for encoding overheads or assignments, as well as the technical representations of ontological and paedagogical knowledge.

Knowledge-Base. All the content should be stored in the knowledge-base, which will administer the above-mentioned Development Graph with various views, including version and configuration management. Content is to be stored in its primary format, as well as possibly in automatically derived formats (for example texts should be stored in the LaTeX, XML and OMDoc [19, 20] formats), if possible distributed. In particular, the content developed during the project should be made available on special archive servers, while students are to have personal knowledge-bases, in which examples can be explored, annotations made, or exercises solved before being sent to a tutor. The personal knowledge-base can also serve as a local copy (or cache) for the students, making them less dependent on their local network.

Standards. Standards are decisive for the technical coupling, but also for semantic integration. Therefore current standardisation attempts in Mathematics and Formal Methods should be adhered to.

The educational content, the description of the meta-structures and the internal communication of the software systems are to be based on the new Internet standard XML. Embedded structuring elements are to be tagged with the special formalism used, such as CASL for the integration of structured algebraic specifications, or input formats for the formal software systems and visualisation components involved in the project; this way, these elements can be analysed by appropriate tools. Formal content should be adapted to the XML dialect OMDoc (OpenMath Documents [19, 20]), which is an extension of the OpenMath standards. OMDoc allows materials to be presented in a series of formats, such as LaTeX, DVI and PostScript for printed documentation, HTML for interactive books or browsable presentations, or MathML for special handling of mathematical formulae.

3 Learning Environment and Communicative Elements

For each rôle (among others authors, teachers and students) the system appears as an individual environment. In the following we will consider the learning environment and its elements in more detail. The learning environment supports the students in various situations:

- selecting (or generating) learning materials from the knowledge base;
- studying the course material interactively;
- communicating with tutors and other students;
- perfoming exercises, practicing and experimenting;
- administrating assignments; evaluating progress.

Tutored and Co-operative Learning, Assignments. Experience at FernUniversität (Distance-University) Hagen shows that the New Media are very good at supporting tutored and co-operative learning. This new way of learning has not yet been generally accepted in the other universities, except in isolated experiments. This will change through the continual availability and extensibility of the system to be constructed by the MMiSS project; its extensive deployment should bring about a new quality of learning.

Modern communication technology permits asynchronous and documented discussion of questions, problems and solutions within structured content-specific discussion-forums (similar to newsgroups) which should be integrated with the course material. In particular such forums can be used for efficient tutoring. It is also possible to realise different levels of visibility (for example, visible for a whole group and its tutors, or only for a particular subgroup); this is known to increase the students' willingness for co-operation.

In *tutored learning*, a tutor is available on the net to answer questions about the content, the use of the system and its tools, or assignments – either synchronously ("talk") or asynchronously (via electronic mail). *Co-operative learning* usually implies a group of students who co-operate in studying a large amount of content (for example a major course) together. A number of learning situations, described by different metaphors, should be supported, for example

- *newspaper stand*: latest information from the teaching staff is distributed
- *café*: a few participants meet in unmoderated synchronous conversation
- *market place*: many communicate asynchronously, for example all participants in a course.

The organisation of assignments poses additional problems. Exercises and sample solutions must be handed out (perhaps for quite different areas and levels of expertise of groups of participants in a course); it is also necessary to administer the forwarding of solutions to the tutors, returning the corrected exercises, and so on. At FernUniversität Hagen, Six's team have developed a tool to address these problems, WebAssign which has already been used in a large number of courses with occasionally more than a thousand students each. This system will be integrated into the current project.

The *added value* of this new form of education is that it raises the quality of learning and makes tutoring more efficient and effective, particularly in a subject area with *as many students* as Computing Science. Experience at Universität Bremen has also shown that alternative forums for interaction and communication improve the students' willingness for and enjoyment of co-operation and help them express themselves; optional anonymity can be useful here. Thus new chances for learning arise and the students are *motivated* by their own sense of

success. *Women-only learning scenarios* (for example a women-only café) become possible. Another advantage of having support for tutored or co-operative learning is that there is very good feedback about student progress, and criticism about the materials can easily be forwarded to the tutors and authors; this clearly assists *quality-control*. The suitability of such approaches to the area of Safe and Secure Systems and Formal Methods becomes even more plausible as the objects of study (specifications, proofs) are per se of a written nature.

Adaptive Learning Environment. Since the system to be constructed should be available to a large user community, we can expect a variety of different application scenarios: on campus learning, self-study, tutored distance-learning, preparation and subsequent assessments of industrial projects, and so on. The educational environment should therefore contain an adaptive user-modelling component, and be oriented to universally understood metaphors when guiding the user; this will significantly increase user acceptance. The learning environment should be able to generate a personalised document from the knowledge-base, covering a special subject-area and configured according to a particular personal profile, with a variety of interaction possibilities. Document generation is to be based on the ontology and dependencies between the terms and the concepts and methods to be learnt. By this personalisation, the learning materials, examples, assignments and the way in which the knowledge is presented can be adapted to the student's state of knowledge and requirements. This adaptability entails on one hand a more individualised support, on the other a flexible re-use of teaching materials.

The foundation for the *user-adaptive generation* of learning elements is a general, partly semantic (and thereby reusable) knowledge representation. To adapt to the profile, goals and the context of the user, such meta-data are acquired in a user model and can be updated for use in a pedagogical presentation planner. The user model will contain, for example, the user's status as a student or industrial trainee, the level of detail the user requires, or whether the user is preparing for an examination.

Interactive Learning. For the study of a content package, the learning environment provides the technical content in an adequate multimedia presentation, for example texts, graphics, pictures, animations, simulations, audio- and video-sequences, and semantic hyperlinks between them. A *navigational aid* leads the students through the content, based on the onotology. Little exercises provide self-tests to *check* the progress.

Within the learning environment, the presentation of the material can be closely interrelated with other functionality; for example the assignments can be referenced directly from the teaching content. This also applies to the embedding of software-development tools. The learning environment also offers direct access to the course-specific communication and evaluation mechanisms.

Support for Assignments and Practical Work. The learning environment will include an assignment component; this will integrate and simplify *embeddings of software-development tools* specific to the course. It will authenticate

students, administrate their solutions and keep track of course marks. Various types of exercises shall be supported:

- multiple choice; exercises with textual or graphical answers;
- creation/modification of specifications, proofs or programs with tool support;
- solution of exercises by dialog with an interactive system.

Evaluation. The learning environment offers, on all those levels we have considered, the possibility of including *support for its evaluation* in the content. As part of the presentation, students can answer questions on the course and add *commentary*. It will also be possible to analyse all the students' responses to an exercise. Points of view publicly put forward as part of communication between students or with the tutors are also potentially useful material for evaluation. The learning environment will provide technical support for managing this data.

4 Embedding of Tools for Safe and Secure Software

The integration of existing formal software development tools shall make teaching more flexible and dynamic. Various interaction levels should be possible, such as "movie-demos" of tools; replays of developments in the tool; completion and extension of developments using the tool; independent working on exercises; working on a project as a team.

Use of Software-Development Tools in Teaching. An important part of training is the familiarisation with computerised tools for developing Safe and Secure Systems. Tools and their methodical use in practical scenarios have, so far, only been integrated into courses in an isolated way, at the moment mostly in the form of complementary tutored exercise sessions. A complete integration into the curriculum has yet to happen, not least because of the general restriction of teaching content to locally-available tools, and the difficulty of providing general methodical integration into the content while avoiding too much detail only of interest in the special case.

As well as educating people in the use of tools, we are also concerned with the methodical improvement of teaching from the point of view of educational theory; an increased use of animations, visualisation and active experimentation can considerably reduce the difficulty of grasping abstract concepts. The user can be aided in make knowledge explicit through an experimental and explorative approach to problem-solving. As is known from Cognitive Psychology, this is an important basis for learning.

As well as integrating tools into teaching content, we should also consider the integration of different tools, and the re-use of developments (represented in a common language) in another (tool-) context; without this coherent teaching is not possible. The project partners have developed (and continue to develop) numerous tools, and demonstrated these in practical applications, up to and including co-operation in large commercial software projects. Above all two approaches that have been developed are relevant. The bmb+f-supported development of UniForM (*Universal development environment for Formal Methods* [21]) at Universität Bremen supports the close interaction and integration of tools.

The MathWeb architecture and the OMDoc format [19, 20], both developed at Universität des Saarlandes, support the loose coupling of tools and the common representation of formal development. Both partners cooperate in developing support for development graphs, their visualisation and administration [7].

Integrated Development Tools. Several formal development tools must be integrated in the learning system, if knowledge of how to use them is to be adequately communicated. This entails consistency problems not just between the tools, but also with the other content. The system must hence be configurable, and support input and output in the most popular formats on the basis of new standards.

The tool support for Formal Methods includes editing, syntactic and static semantic analysis and visualisation of specifications, their animation, interactive proof-development, fully-automated decisions procedures and test-procedures. In addition to systems which address one or other of these tasks, there are also development tools which integrate several sub-systems and so provide general support for formal development. The existing tools differ in their functionality and their fundamental formal approach.

This potential should be exploited in teaching, by complementing the passive absorption of teaching content with active explorative components. We see here the beginnings of a new, decisive approach to improving teaching. Up to now, it is too often the case that what is taught is only of limited applicability, and only limited understanding can actually be said to have been attained. For Formal Methods in particular, current quantitatively and qualitatively limited methods doing assignments (proofs cannot really be worked out) provide no solution.

Because the whole subject area should be covered here, and we aim to be able to adapt to individual special needs and further tool development, the semantically consistent integration of tools and their technical coupling is of high importance; for application-oriented training it is indispensable.

5 Outlook

The project has made good progress during its first year. Many lectures have been converted to the initial LATEX-oriented input format, with good quality output as overhead transparencies in PDF-format. This material is now awaiting further coordination and refinement, as well as semantic interlinking using development graphs in the repository. The Development Manager, and other editing and authoring tools, are well under way towards completion.

Acknowledgement

We are grateful to the members of the Advisory Board, V. Lotz (Siemens AG, München), H. Reichel (TU Dresden), W. Reisig (HU Berlin), D.T. Sannella (University of Edinburgh), and M. Ullmann (BSI [Federal Institute for Security in Information Technology], Bonn), for their advice, and to G. Russel for his help with the manuscript.

References

1. Peter Amthor. *Structural Decomposition of Hybrid Systems – Test Automation for Hybrid Reactive Systems*. PhD thesis. Universität Bremen, 1999. Monographs of the Bremen Institute of Safe Systems 13. Shaker.
2. Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella and Andrzej Tarlecki. CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, to appear 2003.
3. Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella and Andrzej Tarlecki. (eds.). CASL– the CoFI Algebraic Specification Language: Tutorial Introduction, Language Summary, Formal Definition, Basic Data Types. (submitted).
4. Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner (eds.). *Algebraic Foundations of System Specification*. IFIP State-of-the-Art Reports, Springer 2000.
5. Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. INKA 5.0: a logic voyager. *Proc. 16th Intl. Conference on Automated Deduction*, Trento. *LNAI* volume 1632, pages 207–211. Springer, 1999. (see also www.dfki.de/vse/.)
6. Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. Towards an Evolutionary Formal Software Development Using CASL. In Christine Choppy, Didier Bert, and Peter Mosses (eds.): Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT'99, Chateau de Bonas, France. *LNCS* volume 1827, pages 73–88. Springer, 2000.
7. Serge Autexier and Till Mossakowski. Integrating HOLCASL into the development graph manager MAYA. In: A. Armando (ed.). Frontiers of Combining Systems, 4th International Workshop. *LNCS* volume 2309, pages 2–17. Springer, 2002.
8. David A. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *Formal Methods in Systems Design*, 13(3):255–288, November 1998.
9. David A. Basin and Bernd Krieg-Brückner. Formalization of the Development Process. In [4]. 521–562.
10. Michel Bidoit, Hans-Jörg Kreowski, Pierre Lescanne, Fernando Orejas, and Donald Sannella (eds.). *Algebraic System Specification and Development: A Survey and Annotated Bibliography*, *LNCS* volume 501. Springer 1991.
11. Manfred Broy and Stefan Jähnichen (eds.). *KORSO: Methods, Languages, and Tools for the Construction of Correct Software – Final Report*, *LNCS* volume 1009. Springer, 1995.
12. Maura Cerioli, Martin Gogolla, Hélène Kirchner, Bernd Krieg-Brückner, Zhenyu Qian, and Markus Wolf (eds.). *Algebraic System Specification and Development: Survey and Annotated Bibliography.* 2nd edition, 1997. Monographs of the Bremen Institute of Safe Systems 3. ISBN 3-8265-4067-0. Aachen: Shaker, 1998.
13. CoFI. The Common Framework Initiative for algebraic specification and development, electronic archives. Notes and Documents accessible at http://www.cofi.info.
14. CoFI Language Design Task Group. CASL – The CoFI Algebraic Specification Language – Summary. in [13].
15. Carla Freericks. Open-Source Standards on Software Process: A Practical Application. In K. Jakobs (ed.). *IEEE Communications Magazine, Vol. 39, No. 4* (2001) 116–123. See also www.tzi.de/gdpa/
16. Michael Fröhlich. *Inkrementelles Graphlayout im Visualisierungssystem daVinci.* Dissertation. Monographs of the Bremen Institute of Safe Systems 6. ISBN 3-8265-4069-7. Shaker, 1998.

17. Michael Fröhlich and Mattias Werner. *The interactive Graph-Visualization System daVinci – A User Interface for Applications.* Informatik Bericht Nr. 5/94 (1994). Universität Bremen. Up-to-date documentation: www.tzi.de/ daVinci.
18. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.
19. Manfred Kohlhase. *OMDoc: Towards an OpenMath representation of mathematical documents.* SEKI Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. www.mathweb.org/ilo/omdoc/
20. Manfred Kohlhase. OMDoc: Towards an Internat Standard for the Administration, Distribution and Teaching of Mathematical Knowledge. *Proc. Artificial Intelligence and Symbolic Computation. LNAI.* Springer, 2000.
21. Bernd Krieg-Brückner, Jan Peleska, Ernst-Rüdiger Olderog, and Alexander Baer. The UniForM Workbench, a Universal Development Environment for Formal Methods. In: J. M. Wing, J. Woodcock, and J. Davies (eds.): *FM'99, Formal Methods.* Proceedings, Vol. II. *LNCS* Volume 1709, pages 1186-1205. Springer, 1999.
22. Bernd Krieg-Brückner. Seven Years of COMPASS. In *11th Workshop on Specification of Abstract Data Types, Joint with the 8th* COMPASS *Workshop, Oslo*, *LNCS* volume 1130, pages 1–13. Springer, 1996.
23. Erica Melis, Eric Andres, Georgi Goguadse, Paul Libbrecht, Martin Pollet, and Cartsen Ulrich. ActiveMath: System description. In Johanna D. Moore, Carol Redfield, and W. Lewis Johnson (eds.): *Artificial Intelligence in Education.* IOS Press (2001) 580–582.
24. Oliver Meyer. *Structural Decomposition of Timed CSP and its Application in Real-Time Testing.* PhD thesis. Universität Bremen, 2001. (To appear in Monographs of the Bremen Institute of Safe Systems. Logos Verlag.)
25. Till Mossakowski. CASL: From Semantics to Tools. In S. Graf and M. Schwartzbach (eds.) *Tools and Alogorithms for the Construction and Analysis of Systems*, Proceedings TACAS 2000. *LNCS* volume 1785, pages 93–108. Springer, 2000.
26. Till Mossakowski, Kolyang, and Bernd Krieg-Brückner. Static semantic analysis and theorem proving for CASL. In *12th Workshop on Algebraic Development Techniques, Tarquinia*, *LNCS* volume 1376, pages 333–348. Springer, 1998. (For the Bremen CoFI Tools see http://www.tzi.de/cofi.)
27. Peter D. Mosses. CoFI: The Common Framework Initiative for Algebraic Specification and Development. In *TAPSOFT '97: Theory and Practice of Software Development*, *LNCS* volume 1214, pages 115–137. Springer, 1997.
28. Horst Reichel, Till Mossakowski, Markus Roggenbach, and Lutz Schröder. CoCASL – Proof support for co-algebraic specification. In *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT'02*, *LNCS*. Springer (accepted for presentation).
29. Markus Roggenbach. CSP-CASL – A new Integration of Process Algebra and Algebraic Specification. In *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT'02*, *LNCS*. Springer (accepted for presentation).
30. Markus Roggenbach and Till Mossakowski. What is a good CASL specification? In *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT'02*, *LNCS*. Springer (accepted for presentation).
31. Markus Roggenbach and Lutz Schröder. Towards Trustworthy Specifications I: Consistency Checks. In M. Cerioli and G. Reggio (eds.). *Recent Trends in Algebraic Development Techniques, 15th International Workshop, WADT'01, Genova*, *LNCS* volume 2267. Springer. 305-327.

32. Markus Roggenbach and Lutz Schröder. Towards Trustworthy Specifications II: Testing by Proof. In *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT'02*, *LNCS*. Springer (accepted for presentation).
33. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International Series in Computer Science, 1998.
34. Lutz Schröder and Till Mossakowski. HasCASL: Towards integrated specification and development of Haskell programs. In *Proc. AMAST 2002*, *LNCS*. Springer (to appear).
35. Forschergruppe SofTecNRW. *Studie über Softwaretechnische Anforderungen an multimediale Lehr- und Lernsysteme*. Sept. 1999. See also: www.uvm-nw.de, [36].
36. G. Engels, U. Kelter, R. Depke, and K. Mehner. Unterstützende Angebote der Softwarebegleitgruppe. E. E. Doberkat et al. (eds.). *Multimedia in der wirtschaftswissenschaftlichen Lehre – Erfahrungsbericht*. LIT Verlag, Münster (2000) 27–56.
37. Haykal Tej and Burkhart Wolf. A Corrected Failure-Divergence Model for CSP in Isabelle/HOL. In *Proc. FME 97 – Industrial Applications and Strengthened Foundations of Formal Methods*. *LNCS*, volume 1313. Springer (1997) 318-337.

The Impact of Models in Software Development

Manfred Broy

Institut für Informatik, Technische Universität München,
D-80290 München Germany
broy@in.tum.de
http://wwwbroy.informatik.tu-muenchen.de

Abstract. Software construction is essentially a modeling task. The most important decisions in software development are decisions that deal with modeling. The better, the more adequate and more powerful the available modeling paradigms are the easier the program development task is and the better its results are. In the following we describe the role of models in program development and show how closely the issue of modeling is related to the so-called formal methods in program development. We give a number of arguments the usage of mathematical models in software construction and back them up by some detailed examples.

1 Motivation

Software development is still one of the most complex and powerful tasks in engineering. Just by formulating the right programs we obtain engineering artifacts that can control systems, calculate results, communicated messages and illustrate and animate all kinds of information. Since programs are – implicitly or explicitly – based on models and since well-chosen models are the best way to understand software, modeling is a crucial step in software construction.

In all disciplines models play a prominent role. In physics, mathematics has provided lots of models. The same holds for many engineering disciplines. Economy works with models; biology works more and more with models, chemistry works with models. Constructing models is at the heart of science.

In Informatics modeling is even more crucial. Developing software is more or less nothing than developing the right models finally represented in the right notation such that they can be executed on computing machinery.

2 On the Nature of Software Development

Still we have the ongoing discussion what the essence of software development really is. Is it an engineering task? Is it an art, a science, a handicraft or just a simple profession? Of course, there are many views onto program development, more scientific ones or more pragmatic ones. We study and discuss two extreme views in the following:

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 396–406, 2005.

- Academic view: Software development always means the construction of a formal/mathematical/logical model – therefore it is a formal activity. Software is a mathematical object, formally specifiable and verifiable.
- Pragmatic view: Software development is an art and a craft; it proceeds by esoteric lore, by stepwise improvement, by trial and error. It has to be changed and redesigned as well as tested over and over again. It is unreliable and had to predict.

Thus, software is an artifact, complex, unreliable and unpredictable.

Off course, both views are extreme and therefore hardly fully correct. But both views contain valuable insights into the nature of software development. We claim that we have to integrate both views to obtain a real discipline of software engineering. Only if we manage to have a compromise between that both views in a smart way, software development can be improved into a scientifically well founded engineering discipline.

2.1 "Formal Methods" and "Software Engineering"

The scientific community has invested lots of time and effort into so called formal methods. In formal methods the idea is that the task of software development including specification, its incremental design, its implementation and its verification can be done completely within a logical and mathematical theory. This is a striking idea, full of interesting scientific issues and insights. However, practitioners often consider formal methods inadequate, too expensive, too difficult, and "not at all practical".

On the other side the state of the art in pragmatic and practical software development is still far from being satisfactory. Practical software development is often considered being "ad hoc", "immature", uncontrollable, and "not an engineering discipline".

Therefore a very interesting question is how we can find a good compromise between the rigorous scientific approach to programming and the pragmatic practical approach. One idea of course is the use of well-chosen, sufficiently formal models. Programming means in any case using explicitly or implicitly models. We claim that it is important to identify the used model very explicitly and that this is of great practical advantage since finally formal methods provide a rich tool kit of methods.

2.2 Modeling in Software Engineering

Systematic development of distributed interactive software systems needs basic system models. Description techniques are to provide specific views and abstractions of systems such as:

- the data view,
- the interface view,
- the architecture view,
- the distribution view,

- the process view,
- the interaction view,
- the deployment view,
- the state transition view.

All these views have to be captured by carefully chosen models. The development of systems concentrates on working out these views that lead step by step to an implementation.

2.3 Software Development as Modeling Tasks

Software development can be seen as a sequence of modeling tasks. From the very beginning, when analyzing and understanding a problem domain we start to work towards finding the right models. This goes on and on when we analyze the use cases and their specifications, the software architecture, the modularization of the system and its implementation. Software development includes the modeling and description of various aspects, such as:

- application domains, their data structures, laws, and processes,
- software requirements, based on data models, functions, and processes,
- software architectures, their structure and principles,
- software components, their roles, interfaces, states, and behaviors,
- programs and modules, their internal structure, their runs and their implementation,
- test cases, their generality and usefulness.

If models are so important in software and systems engineering, a central question of course is, what is a model in software engineering?

1. An annotated graph or diagram?
2. A collection of class names, attributes, and method names?

In engineering, a model is always given by a collection of formulas, diagrams, and tables as well as text expressed in some notation with a well-understood mathematical theory! In analogy, software engineering needs mathematical modeling theories of digital systems – algebra, logic, model theory! Logic provides a unifying frame for that!

3 The Role of Diagrams

Practical software engineers often prefer the use of diagrams to textual notation using formulas and programming languages. The reason is quite obvious. They assume that diagrams are more telling, easier to understand and better to grasp. Whether this holds actually true leads into a long, controversial discussion.

Nevertheless in some applications certainly diagrams are helpful. However, on the long run diagrams are only helpful if they are based on a proper theory of understanding. Well-chosen models can provide such an understanding.

3.1 Practice Today: Diagrams

In practice, today we find many diagrammatic methods for modeling and specification (SA, SADT, SSADM, SDL, OMT, UML, ROOM, ..., see [22, 12, 3, 19, 20, 23]) in software and systems engineering. Especially UML has gained much attention. The idea of universal modeling languages is a great one – but a closer look shows, however, how ad hoc most of these "methods" especially as found in UML are. At best, they reflect essential engineering insights in engineering software applications. Never have practical diagrammatic modeling been justified on the basis of a comprehensive mathematical foundation. In contrast, only after the languages where published scientists work had to define and explain the ad hoc constructs of modeling language post mortem.

3.2 Limitations of Diagrams

By a look at the state of the art today we see that a lot of diagrams are used without a proper theory and without a good support of understanding. To underline this remark we mention three bad examples (see also [24]):

- UML and its statecharts dialect with its endless discussions about its semantics.
- ehavior specification of interfaces of classes and components in object oriented modeling techniques in the presence of callbacks.
- Concurrency and co-operation: Most of the practical methods especially in object orientation seem to live in the good old days of sequential programming and do not properly address the needs of highly distributed, mobile, asynchronously co-operating software systems.

We need proper theories and methodological insights (see [21]) to overcome these shortcomings.

3.3 From Logic to UML Back to Logic

It is a disaster for academic informatics that it did not manage to design a modeling language that is used as widely as UML. The vision however, remains – an academic, scientific view on modeling! How can we achieve that? We start from foundations: A tractable scientific basis, understanding, and theory for modeling, specifying, and refining in programs, software and system. On that basis we identify powerful models supporting levels of abstractions, multi-view modeling, domain modeling. This leads to comprehensive description techniques based on these foundations. Thus we gain a family of justified engineering methods based on these foundations and finally a flexible process model combining these methods.

All this is the necessary prerequisite for a comprehensive tool support in software development including validation, consistency checks, verification, and code generation by algorithms and methods justified by the theories. Finally we arrive at modeling and its theory as an integral part of software construction as an engineering discipline.

4 An Example: MSC Semantics

In this section we treat as an example of diagrams and the corresponding theory the example of message sequence charts (see [14–16, 18, 17, 1, 2, 5, 6, 8–10, 13]). We very shortly discuss issues of message sequence charts for a comprehensive understanding of the issues we prefer to comprehensive paper by [5].

4.1 What Is an MSC

A message sequence chart (MSC) is first of all an example of an interaction and communication trace of a system. Using sets of message sequence charts naively in that sense may help to get a first understanding of the requirements of a system. But it will never lead to a useful and comprehensive requirements specification for a complex system. The reason is obvious. Giving a bunch of example traces can never specify a system in a sense that expresses what the system must do and what a system must not. A set of traces only gives some ideas what a system might do.

However, if we replace simple messages by variables and carefully distinguish between input and output according to the idea that input gives a certain context for a trace. Then the message sequence chart fixes a reaction in that context. Therefore the message sequence charts can be seen just as a set of equations as we will point out in more detail in the following.

4.2 The Role of MSCs in the Development Process

A message sequence chart is just a diagram of a form as it is shown in figure 1. The diagrams specify the communication sequences between the components of system architectures. Figure 1 shows a system architecture and figure 2 shows the interpretation of a message sequence charts.

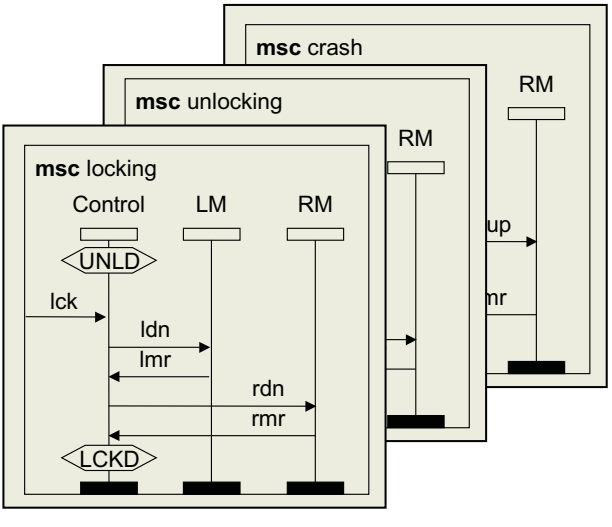


Fig. 1. An Example of a Message Sequence Chart.

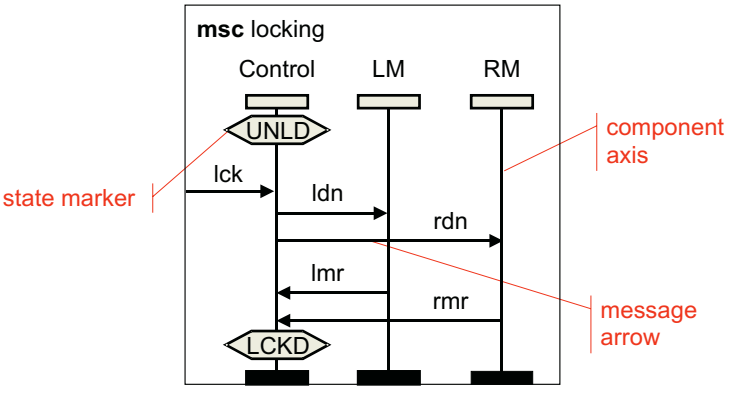


Fig. 2. A Message Sequence Chart with its Constituents.

The methodologically interesting question is what the role of message sequence charts is in the development process. In fact, we see many methodological roles which can only be covered by message sequence charts provided we can give a proper semantics to message sequence charts.

Our ambitious goal is a seamless, methodologically founded integration of MSCs into the development process for distributed, reactive systems.

In the connection of the methodological use of message sequence charts and its theoretical foundations we have to deal with a number of difficult questions in connection with message sequence charts: first of all, it is always important to understand to which properties of a system message sequence charts refer to. This can only be explained if we have a clear understanding of a system model. In the following we give a simple system model and show how message sequence charts can be understood as equations in term of the system model.

We are in particular interested in the following questions:

- What is the formal meaning of
  - a single MSC?
  - a set of MSCs?
- for the individual components of a system?
- How can several MSCs be combined?
- What properties do MSCs specify?
- What is the methodological role of MSCs?
- Can MSCs serve as a precise and comprehensive specification method?

The system class that we consider are distributed, reactive systems. First we have to select a model for such systems before we can fix the meaning of MSCs.

We use the Focus model (see [7]). There a system consists of

- named components (with local state)
- named channels

driven by global, discrete clock over which the components communicate via data streams.

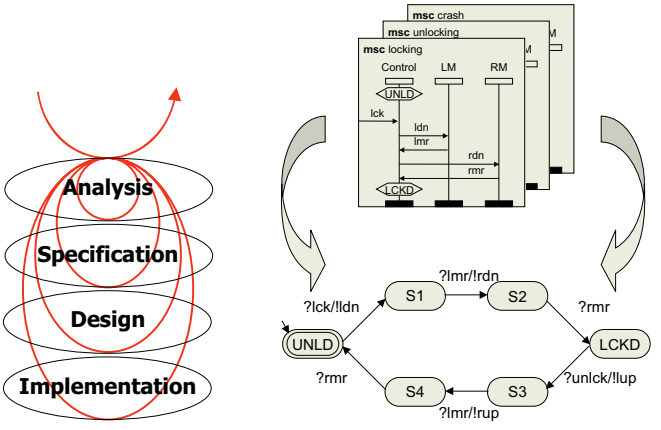


Fig. 3. The Role of MSCs in the Development Progress.

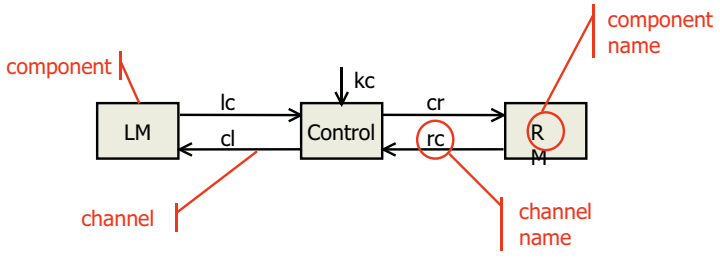


Fig. 4. The System Model.

The basic model is based on timed streams. This leads to a semantic model for black-box-behavior.

On each channel that connects two components there flows a stream of messages in a time frame as shown in Fig. 5.

4.3 MSC Semantics for Deterministic Systems – An Example

To show how easy it is to give a powerful meaning to MSCs we look at a simple example.

The two MSCs in Fig. 6 are translated into the following equations:

$$f_{TR}(\langle a : m\rangle) = \langle b : ready\rangle$$
$$f_{TR}(\langle a : m\rangle\hat{}\langle c : Y\rangle\hat{}x) = \langle b : ready\rangle\hat{}\langle d : Y\rangle\hat{}\langle b : m\rangle\hat{}f_{TR}(x)$$
$$f_{TR}(\langle a : m\rangle\hat{}\langle c : N\rangle\hat{}x) = \langle b : ready\rangle\hat{}\langle d : N\rangle\hat{}f_{TR}(x)$$

This translation is based on the understanding that for every thread incoming arrows denote input and outgoing arrows denote output. Here we assume the component is deterministic in the sense that it reacts to every input pattern by a uniquely defined output pattern. For this case we assume that after each pattern

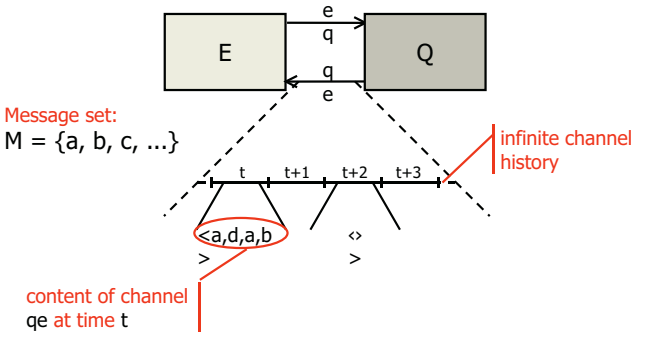


Fig. 5. Streams as Models of the Interaction between Components.

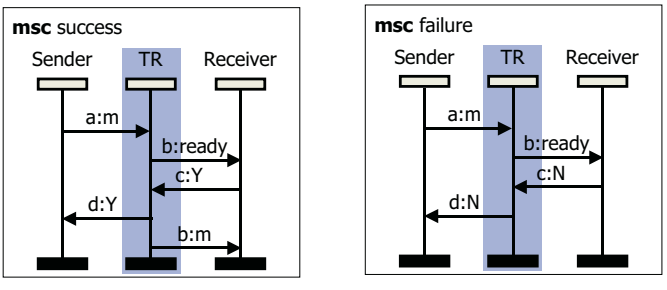


Fig. 6. Two MSCs.

of behaviour the components are in their initial state again. If the behaviour is more specific, states can be introduced explicitly, too (see [5]).

We get a seamless support of the development process as outlined in figure 7 to support software development.

As our simple example above shows, being able to give a theory for message sequence charts in terms of an appropriate system model we can get a very concise interpretation of message sequence charts. This goes up to a point where message sequence charts can just be seen as a graphic representation of logical equations. This provides a major step in the understanding diagrams and software development and gives the basis also for very comprehensive tool support along the lines of AutoFocus.

4.4 Property Specification with MSCs

In the end we can use message sequence charts now as a precise description of logical properties of system represented by diagrams. So message sequence charts can be seen as a means of formal specifications of systems. This leads to a helpful compromise and finally to an amalgamation of pragmatic graphical techniques and formal models. It shows that a proper theoretical foundation of a diagrammatic method leads to a much better, better supported programming methodology that combines the advantages of formal methods and practical approaches to software development (see also [11]).

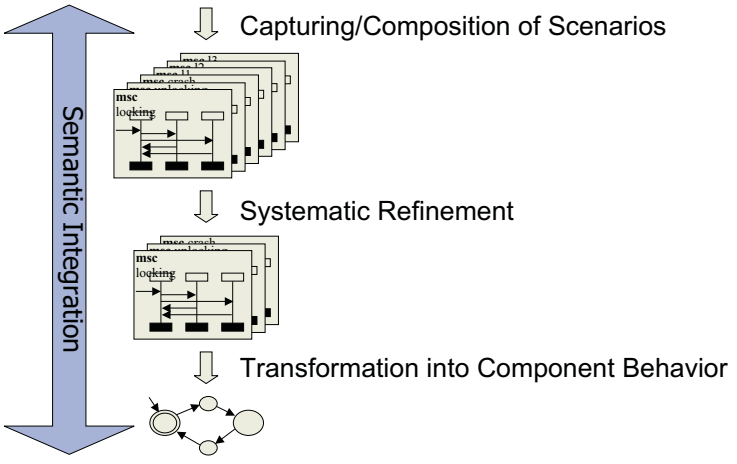


Fig. 7. MSCs in Software Development.

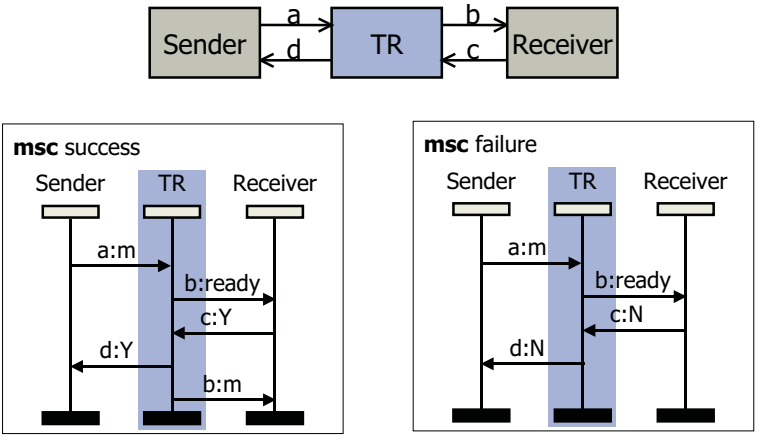


Fig. 8. System Architecture and MSCs.

5 Summary and Outlook

Software development is a difficult and complex engineering task. It would be very surprising if such a task could be carried out properly without a proper theoretical framework. It would at the same time being quite surprising if a purely scientifically theoretical framework would be the right approach for the practical engineer. The result has to be a compromise as we have shown between formal techniques and theory on one side and intuitive notations based on diagrams. Work is needed along those lines including experiments and feedback from practical applications. But as already our example shows a lot is to be gained that way.

Theory and practical understanding are the key to mature software development. To achieve that we need a much deeper and more intensive interaction between researchers working on the foundations, the designers of practical engineering methods and tools, the programmers and engineers in charge of practical solutions, and application experts modeling application domains.

Successful work does not only require the interaction between these types of people – it also needs *hybrid* people that have a deep understanding in all three of these areas.

Acknowledgements

It is a pleasure to thank Ingolf Krüger and Radu Grosu for stimulating discussions and helpful remarks on draft versions of the manuscript.

References

1. R. Alur, G.J. Holzmann, D. Peled: An Analyzer for Message Sequence Charts. Software – Concepts and Tools 17 (1996), 70-77
2. H. Ben-Abdallah and S. Leue: Syntactic analysis of Message Sequence Chart specifications. Tech Report 96-12, Department of Electrical and Computer Engineering, University of Waterloo, 1996.
3. G. Booch: Object Oriented Design with Applications. Benjamin Cummings, Redwood City, CA, 1991
4. M. Broy: Towards a formal foundation of the specification and description language SDL. Formal Aspects of Computing 3, 1991, 21-57
5. M. Broy: The Essence of Message Sequence Charts. Keynote Speech. In: Proceedings of the International Symposium on Multimedia Software Engineering, 11-13 December 2000, IEEE Computer Society 2000, 42-47
6. M. Broy, C. Hofmann, I. Krüger, M Schmidt: A Graphical Description Technique for Communication in Software Architectures. Technische Universität München, Institut für Informatik, TUM-I9705, Februar 1997
   URL: http://www4.informatik.tu-muenchen.de/reports/TUM-I9705, 1997. Also in: Joint 1997 Asia Pacific Software Engineering Conference and International Computer Science Conference (APSEC'97/ICSC'97)
7. M. Broy, K. Stølen: Specification and Development of Interactive Systems: FOCUS Focus on Streams, Interfaces, and Refinement. Springer 2001
8. J.M.H. Cobben, A. Engels, S. Mauw, M.A. Reniers: Formal Semantics of Message Sequence Charts. Eindhoven University of Technology, Departement of Computing Science, Technical Report CSR 97/19
9. W. Damm, D. Harel: Breathing Life into Message Sequence Charts. Weismann Insitute Tech. Report CS98-09, April 1998, revised July 1998, to appear in: FMOODS'99, IFIP TC6/WG6.1 Third International Conference on, Formal Methods for Open Object-Based Distributed Systems, Florence, Italy, February 15-18, 1999
10. P. Graubmann, E. Rudolph, J. Grabowski. Towards a Petri Net based semantics definition for message sequence charts. In O. Faergemand and A. Sarma, editors, Proceedings of the 6th SDL Forum, SDL'93: Using Objects, October 1993

11. G. J. Holzmann, D. A. Peled, M. H. Redberg: Design Tools for Requirements Engineering. Bell Labs Technical Journal, Winter 1997, 86-95
12. I. Jacobsen: Object-Oriented Software Engineering. Addison-Wesley, ACM Press 1992
13. I. Krüger, R. Grosu, P. Scholz, M. Broy: From MSCs to statecharts. In: Proceedings of DIPES'98, Kluwer, 1999
14. ITU-T (previously CCITT) (March 1993) Criteria for the Use and Applicability of Formal Description Techniques. Recommendation Z. 120, Message Sequence Chart (MSC), 35pgs.
15. ITU-T. Recommendation Z.120, Annex B: Algebraic Semantics of Message Sequence Charts. ITU-Telecommunication Standardization Sector, Geneva, Switzerland, 1995.
16. P.B. Ladkin, S. Leue. Interpreting Message Sequence Charts. Technical Report TR 101, Department of Computing Science, University of Stirling, 1993.
17. P.B. Ladkin, S. Leue. Interpreting Message Flow Graphs. Formal Aspects of Computing, 7(5): 473-509, 1995.
18. S. Mauw, M.A. Reniers. An algebraic semantics of basic message sequence charts. The Computer Journal, 37(4): 269-277, 1994.
19. B. Selic, G. Gullekson. P.T. Ward: Real-time Objectoriented Modeling. Wiley, New York 1994
20. J. Rumbaugh: Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs: New Jersey 1991
21. B. Rumpe: Formale Methodik des Entwurfs verteilter objektorientierter Systeme. Ph.D. Thesis Technische Universität München, Fakultät für Informatik 1996. Published by Herbert Utz Verlag
22. Specification and Description Language (SDL), Recommendation Z.100. Technical report, CCITT, 1988
23. G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language for Object-Oriented Development, Version 1.0, RATIONAL Software Cupertino
24. P. Zave, M. Jackson: Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology, January 1997

Formal Software Development in MAYA

Dieter Hutter and Serge Autexier*

German Research Center for Artificial Intelligence,
Stuhlsatzenhausweg 3, D-66123 Saarbruecken, Germany
{autexier,hutter}@dfki.de

Abstract. The formal development of industrial-size software is an error-prone and therefore an evolutionary process. Verifying formal specifications usually reveals hidden errors causing the change of parts of the specification. Also adding new functionality will result in changes of the specification which always endangers the verification work already done. In this paper we describe the system MAYA which maintains formal developments. The MAYA-system supports an evolutionary formal development since it allows users to specify and verify developments in a structured manner, incorporates a uniform mechanism for verification in-the-large to exploit the structure of the specification, and maintains the verification work already done when changing the specification. MAYA relies on development graphs as a uniform representation of structured specifications, which enables the use of various (structured) specification languages to formalize the software development. Moreover, MAYA allows the integration of different theorem provers to deal with the actual proof obligations arising from the specification, i.e. to perform verification in-the-small.

1 Introduction

Formal methods are used in the software development process to increase the security and safety of software. The software systems as well as their requirement specifications are formalized in a textual manner in some specification language like CASL [4] or VSE-SL [7]. The specification languages provide constructs to structure the textual specifications to ease the reuse of components. Exploiting this structure, e.g. by identifying shared components in the system specification and the requirement specification, can result in a drastic reduction of the proof obligations, and hence of the development time which again reduces the overall project costs.

Creating the arising proof obligations in a naive way by postulating all parts of the security requirements as theorems of the system design would result in umpteen redundant proof obligations relating to common datastructures. Exploiting the given (graph-) structure of specifications allows one to reveal this redundancy. In [2] we proposed the use of development graphs to represent defined and postulated properties of formal specifications in a logical way. We

* This work was supported by the German Ministry for Education and Technology (BMBF).

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 407–432, 2005.

will introduce a calculus $\mathcal{DG}$ to verify postulated properties. The calculus rules decompose conjectures between specifications into conjectures between parts of the specification and check whether some of those are already subsumed by the specification structure. We denote this activity by *verification in-the-large*. Those conjectures that can neither be further decomposed nor subsumed give rise to the proof obligations that must actually be tackled by some theorem prover, which is denoted by *verification in-the-small*.

However, the logical formalization of software systems is error-prone. Since even the verification of small-sized industrial developments requires several person months, specification errors revealed in late verification phases pose an incalculable risk for the overall project costs. An *evolutionary formal development* approach is absolutely indispensable. In all applications so far development steps turned out to be flawed and errors had to be corrected. The search for formally correct software and the corresponding proofs is more like a *formal reflection* of partial developments rather than just a way to assure and prove more or less evident facts. Revealed flaws give rise to changes of the specification and to the need for an update of all the proof work done before. Loosing this work would be an incalculable risk of the overall project costs for large verification tasks that arise in practice. Hence, we introduce a management of change based on the notion of development graphs to incrementally adjust existing proofs to a changed specification, while preserving as much information about proven conjectures as possible.

We start with a general overview of how MAYA supports the formal development of software in Sect. 2. In Sect. 3 we introduce the formal notion of development graphs that underly the MAYA system. Sect. 4 describes the general methodology to define translation of an input specification given in some specification language $\mathcal{L}$ into development graph. The methodology is illustrated by providing the definition of the translation of CASL-specifications into development graphs. Sect. 5 is concerned with the computation of differences between specifications and how to update their logical representation inside the development graph. Sect. 6 presents the management of change for both, verification in-the-large and verification in-the-small, while we discuss its implementation in MAYA and related work in Sects. 7 and 8.

2 General Overview

A user interacts with the MAYA-system via a formal specification language. Such a formal specification gives rise to a logical modeling of the specification and the proof obligations arising from commitments made inside the specification. Examples of such commitments are that theories satisfy specific properties specified inside so-called security models or that basic specifications imply specific properties, so-called theorems. Translating the textual specification into a structured logic representation, which we call a development graph, proof obligations are denoted either as so-called theorem links between theories, indication that both theories are related to each other wrt. a specific property or as so-called theorems representing lemmata inside a particular theory. The problem of establishing

properties between theories is dealt with inside the MAYA-system utilizing the overall structure of the graph until we end up with elementary proof obligations which are tackled by external theorem provers.

The user performs changes of her specification always on the textual representation, which gives rise to the problem of tracking the changes in the textual specification to arising changes in the corresponding logical representation and last not least also in changes or adaptation of the proofs. Changed textual specifications are translated into their logical counterpart. The analysis which parts of the specification have changed is done on the logical level. The result of the analysis is an operational description of how to adjust the existing development graph such that it fits the changed specification. The adjustment of the proof work is based on the operation description incorporating precompiled knowledge how individual operations will affect the validity of proofs.

3 Development Graphs

In order to define development graphs we start with a short recapitulation of the basics of logics as they are given, for instance, in [11]. Thereby the notion of a logic is based on the notions of an *institution* and an *entailment system*.

An *institution* $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consists of a category of signatures **Sign**, two functors **Sen** and **Mod** giving respectively the set of valid sentences $\mathbf{Sen}(\Sigma)$ and the models $\mathbf{Mod}(\Sigma)$ for some signature, and a satisfaction relation $\models_\Sigma \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$ for each signature Σ. An *entailment system* $\mathcal{E} = (\mathbf{Sign}, \mathbf{Sen}, \vdash)$ consists of a category **Sign** of *signatures*, a functor $\mathbf{Sen}\colon \mathbf{Sign} \longrightarrow \mathbf{Set}$ giving the set of *sentences* over a given signature, and entailment relations $\vdash_\Sigma \subseteq |\mathbf{Sen}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ with the following properties:

1. *Reflexivity:* for any $\varphi \in \mathbf{Sen}(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
2. *Monotonicity:* if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
3. *Transitivity:* if $\Gamma \vdash_\Sigma \varphi_i$, for $i \in I$, and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
4. $\vdash$-*Translation:* if $\Gamma \vdash_\Sigma \varphi$, then for any $\sigma\colon \Sigma \longrightarrow \Sigma'$ in **Sign**, $\sigma[\Gamma] \vdash_{\Sigma'} \sigma(\varphi)$.

A logic is then defined as a 5-tuple $\mathcal{LOG} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \vdash, \models)$ such that: (1) $(\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ is an institution (denoted by $inst(\mathcal{LOG})$), (2) $(\mathbf{Sign}, \mathbf{Sen}, \vdash)$ is an entailment system (denoted by $ent(\mathcal{LOG})$), and (3) the following *soundness condition* is satisfied: for any $\Sigma \in |\mathbf{Sign}|$, $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ and $\varphi \in \mathbf{Sen}(\Sigma)$, $\Gamma \vdash_\Sigma \varphi$ implies $\Gamma \models_\Sigma \varphi$. Throughout the rest of the paper, we will work with an arbitrary but fixed logic $\mathcal{LOG} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \vdash, \models)$.

Structured specifications are represented on a logical basis as development graphs. The nodes of such a graph represent individual theories. Definition links are used to specify theory inclusions (with respect to some morphism) between different theories. The axiomatic specification of a single theory is distributed to the subgraph of the corresponding node since the definition of the theory of a node depends on the local axioms attached to the node combined with the axioms or theories of the nodes imported by definition links.

In order to formulate proof obligations denoting properties between different theories (*verification in-the-large*) we introduce so-called theorem links. These

links are similar in appearance to definition links but do not influence the theories denoted by the nodes. Formally we define

Definition 1. *A* development graph $\mathcal{S}$ *is a directed graph* $\langle \mathcal{N}, \Psi \rangle$*, where*

- $\mathcal{N}$ *is a finite set of nodes. Each node* $N \in \mathcal{N}$ *is a pair* (Σ_l^N, Φ_l^N) *consisting of a* **local signature** Σ_l^N *and a set of* **local axioms** $\Phi_l^N \subset \mathbf{Sen}(\Sigma^N)$ *of* N*.*
- $\Psi = \Psi_D \uplus \Psi_T$ *is a finite set of directed links between elements of* $\mathcal{N}$ *consisting of an acyclic*[1] *set* Ψ_D *of* **definition links** *and a set* Ψ_T *of* **theorem links***. Each link from a node* M *to a node* N *in* Ψ *is either* **global** *(denoted* $M \xrightarrow{\sigma} N$*) or* **local** *(denoted* $M \xrightarrow{\sigma} N$*) and is annotated with a signature morphism* $\sigma : \Sigma^M \to \Sigma^N$*.*

For all $N \in \mathcal{N}$ *the* **signature** Σ^N *of* N *is given by:*

$$\Sigma^N = \Sigma_l^N \cup \{\sigma(f) | f \in \Sigma^M, M \xrightarrow{\sigma} N \in \Psi_D\} \cup \{\sigma(f) | f \in \Sigma_l^M, M \xrightarrow{\sigma} N \in \Psi_D\}$$

For the implementation, we represent a signature morphism σ by a set of finite pairs (f_{in}, f_{out}) with $\sigma(f) = g$ if there is a pair $(f, g) \in \sigma$ and $\sigma(f) = f$ otherwise.

The proof theoretical semantics of a development graph is given by the following definition:

Definition 2. *Let* $\mathcal{S} = \langle \mathcal{N}, \Psi \rangle$ *be a development graph and* $\Delta \subseteq \Psi$*,* Δ *acyclic. Let* $N \in \mathcal{N}$*, then the* **theory** $Th_\Delta(N)$ *of* N **relative** *to* Δ *is defined by*

$$Th_\Delta(N) = \left[\Phi_l^N \cup \bigcup_{K \xrightarrow{\sigma} N \in \Delta} \sigma(Th_\Delta(K)) \cup \bigcup_{K \xrightarrow{\sigma} N \in \Delta} \sigma(\Phi_l^K) \right]^{\vdash_{\Sigma^N}}$$

where $[\Gamma]^{\vdash_{\Sigma^N}}$ *denotes the closure of* Γ *under the entailment relation* $\vdash_{\Sigma^N}$*. The* **theory** $Th(N)$ *of* N *is defined as* $Th_{\Psi_D}(N)$*.*

Fig. 1 presents a development graph for lists LIST and stacks STACK over arbitrary elements and their respective instantiations to lists NATLIST and stacks

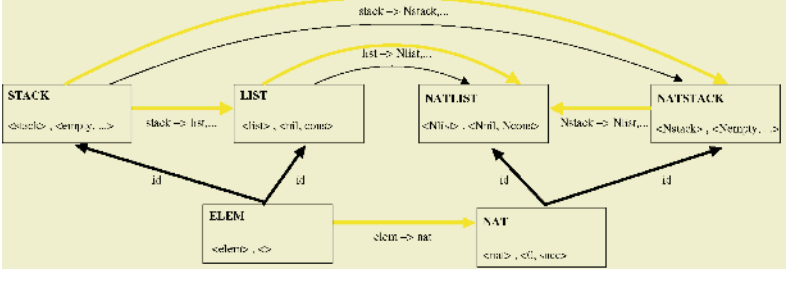


Fig. 1. Structured Specifications of NATLIST.

[1] A set of links is acyclic iff the graph denoted by these links is acyclic.

NatStack over natural numbers. While we included the local signatures of the nodes in the figure we have omitted the local axioms because of shortage of space. The theories of generic lists List and generic stacks Stack are defined with the help of a theory Elem, indicated by the global definition links from Elem to List and Stack, and local axioms in List and Stack specifying that List and Stack are freely generated. The global theorem link between Stack and List represents the proof obligation, that Stack can be implemented by List.

The theories of lists and stacks of natural numbers, NatList and NatStack, are instantiations of generic lists and stacks with natural numbers Nat. Thus, both NatList and NatStack import Nat via a global definition link and the local axioms of List and Stack respectively via local definition links. The global theorem links between List and NatList, and Stack and NatStack denote the proof obligations that NatList and NatStack are respective instances of List and Nat. The global theorem link between Elem and Nat denotes the proof obligation that the actual parameter Nat satisfies the requirements of the formal parameter Elem. The proof obligation that NatStack can be implemented by NatList is represented by a global theorem link from NatStack to NatList.

This toy example illustrates how the important concepts from structured specifications are represented with development graphs. In practice the system and requirement specifications and hence the resulting development graph are much larger. A development graph of a typical size is sketched in Fig. 2.

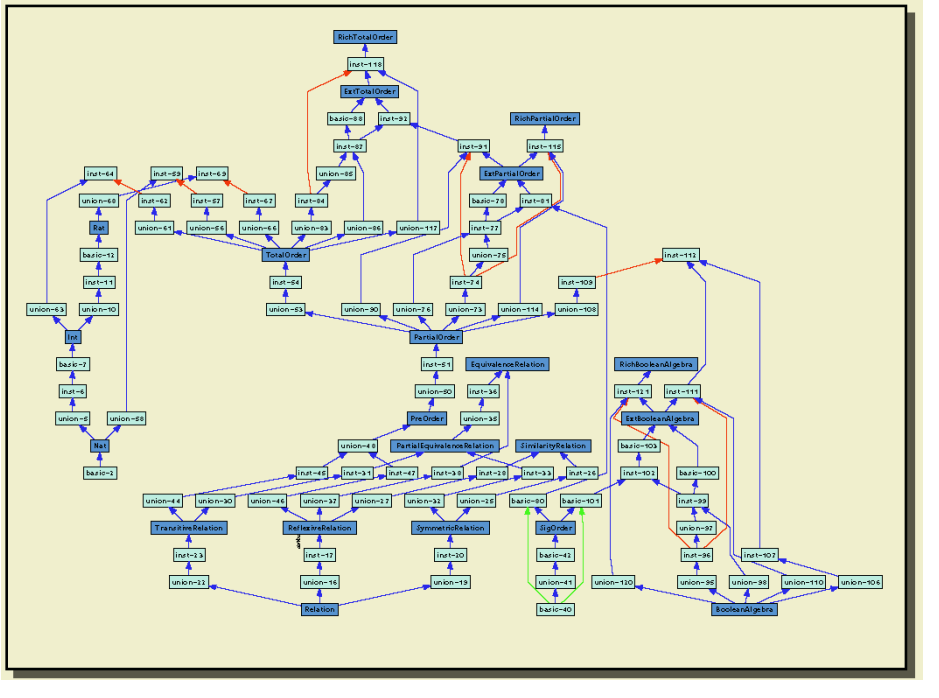


Fig. 2. Example of development graphs for real software engineering problems.

4 Translating Specifications into Development Graphs

In formal software development, specification languages like CASL [4] or VSE-SL [7] are used to describe a system and its requirements as well as proof obligations arising from the requirement that the system must satisfy these requirements. The notion of development graph from Sect. 3 provides a uniform representation of general structured formal developments. The representation is independent of any specification language, which eases the use of different specification languages as input format. To use a specification language $\mathcal{L}$ as input for MAYA we must define a mapping from $\mathcal{L}$-specifications into the representation language of development graphs. Roughly speaking, such a translation of some $\mathcal{L}$-specification S works as follows:

- it maps basic (unstructured) parts of the specification, like the specification of simple abstract datatypes, into a collection of (local) axioms within some theory node,
- it translates the structuring operations of the specification language $\mathcal{L}$ into the notion of definition links, and
- it reformulates proof obligations given in the specification either into theorem links connecting corresponding theories or into conjectures considered as lemmata of a specific theory in the development graph.

Obviously, the definition of such a translation entails the requirement to prove the adequacy of the translation of $\mathcal{L}$-specifications into the notion of development graphs.

In the following we illustrate the translation of specifications into development graphs by giving the rigorous definition of the mapping of CASL-specifications into development graphs.

4.1 Translation of CASL into Development Graphs

For the translation of CASL into development graphs we restrict ourselves to a subset of the CASL-language, namely CASL-specifications without the structuring operations hiding and freeness (cf. [4]) and without architectural specifications. We use the CATS-parser [12] to parse the specifications, which also provides encoding of the logical parts of specifications from the CASL-logic into different target logics. Since the logic underlying the actual implementation of development graphs in MAYA is many-sorted higher-order logic, we use second-order logic without subsorting as the target logic. Furthermore, the CATS-parser performs a static analysis of the specifications.

Basic specifications constitute the nucleus of CASL-specifications. Consisting of a (local) signature and a set of axioms, basic specifications are translated into the higher-order logic with the help of the CATS-parser which provides second-order logic encodings. *Structured specifications* are used to combine *basic specifications* with the help of structuring operations, like for example extension (**then**), union (**and**), or to actualize parameterized specifications.

A CASL-specification itself consists of a list of specification parts which are either *named specifications*, *named views*, or *fitting views*, which are constructed

with the help of structured specifications. We will describe these constructs in more details lateron.

To translate a CASL-specification, we define a top-level translation function τ_{CASL} that iterates over the specification parts and that iteratively constructs the corresponding development graph. The major difficulty of this translation is the encoding of the so-called *linear visibility constraint*, which is implicitly given by the CASL-semantics: the semantics of a specification part depends on its global environment which depends on previously parsed specification parts. Thus besides a list of CASL-specifications, τ_{CASL} requires the global environment as an additional argument that provides information about translated specifications and views before parsing the actual list of CASL-specifications.

τ_{CASL} returns the actualized development graph enlarged by nodes and links corresponding to the parsed specification list and provides the new global environment. Inside this translation information we accumulate for instance the information how named specifications or named views have been translated. Lateron we make use of this information to perform actualizations. Formally, we define τ_{CASL} by recursion as follows:

$$\begin{gathered}\tau_{CASL}(\langle\rangle, \mathcal{S}, \mathcal{P}) := (\mathcal{S}, \mathcal{P}) \\ \tau_{CASL}(\langle \textit{spec-part}, \textit{restlist}\rangle, \mathcal{S}, \mathcal{P}) := \tau_{CASL}(\textit{restlist}, \mathcal{S}', \mathcal{P}') \\ \textit{with } (\mathcal{S}', \mathcal{P}') := \tau_{part}(\textit{spec-part}, \mathcal{S}, \mathcal{P})\end{gathered}$$

τ_{part} is the corresponding translation function for the individual specification parts. It takes as arguments a structured specification *spec-part*, a development graph $\mathcal{S}$, and a translation information $\mathcal{P}$. It provides a pair $(\mathcal{S}', \mathcal{P}')$ where $\mathcal{S}'$ is a new development graph that includes the subgraph resulting of the translation of the structured specifications and $\mathcal{P}'$ is the updated translation information.

Since CASL specification parts are constructed with the help of structured specifications, we will introduce a third translation function τ to translate structured specifications into development graphs. τ takes as argument the specification *Spec*, the development graph $\mathcal{S}$ and translation information $\mathcal{P}$. It returns a triple $(\mathcal{I}', \mathcal{S}', O')$ with $\mathcal{S}'$ being the development graph updated with the translated *Spec*. The linear visibility constraints for structured CASL-specifications defines *how* specification parts are visible when parsing the next specification. Translating this requirement in terms of development graphs, the development graphs of previous specifications have to be imported to the graph of the specification part under consideration. Therefore τ returns also a set $\mathcal{I}'$ of nodes denoting the import interface to the global environment and a node O' which corresponds to the exported global environment.

4.2 Named Specifications

A named specification in CASL is of the form

$$\textbf{spec } SN[SP_1]\ldots[SP_n] \textbf{ given } SP'_1, \ldots, SP'_m = SP \textbf{ end}$$

where SN is the name of the specification, $SP_1, \ldots, SP_n$ are the parameter specifications, $SP'_1, \ldots, SP'_m$ specifications that are visible inside the parameter

specifications, and SP is the body of the named specification. For the definition of the translation, we use the translation function τ for specification bodies. This function takes three arguments: (1) the CASL-specification body to translate, (2) the actual development graph, (3) the actual translation information. This function returns a triple $\langle \mathcal{I}, \mathcal{S}, O \rangle$, containing the new development graph $\mathcal{S}$, the theory nodes $\mathcal{I}$ which import the visible environment of the argument specification body, and the theory node O which exports the new visible environment.

In order to satisfy the visibility rules from CASL the translation is done according to the following steps

1. The union of the "given" specifications $SP'_1, \ldots, SP'_m$ is translated:

$$\langle \mathcal{I}_0, \mathcal{S}_0, O_0 \rangle := \tau(SP'_1 \textbf{ and} \ldots \textbf{and } SP'_m, \mathcal{S}, \mathcal{P})$$

2. Parameter specifications SP_i are translated for all i, $1 \leq i \leq n$ by:

$$\langle \mathcal{I}_i, \mathcal{S}_i, O_i \rangle := \tau(SP_i, \mathcal{S}_{i-1}, \mathcal{P})$$

   and new definition links are inserted to import the output theory node O_0 into all elements of all $\mathcal{I}_i$ with $1 \leq i \leq n$: Let $\mathcal{S}_n$ be $\langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ then

$$\mathcal{S}_{n+1} := \langle \mathcal{N}, (\Psi_D \cup \{O_0 \xrightarrow{\lambda} N \mid N \in \mathcal{I}_i \wedge 1 \leq i \leq n\}) \uplus \Psi_T \rangle$$

3. Next, the body SP of the named specification is translated by:

$$\langle \mathcal{I}_{n+2}, \mathcal{S}_{n+2}, O_{n+2} \rangle := \tau(SP, \mathcal{S}_{n+1}, \mathcal{P})$$

   and new definition links are added to import the parameters $O_1, \ldots, O_n$ into each element N of $\mathcal{I}_{n+2}$ obtain by translating the body. Let $\mathcal{S}_{n+2}$ be $\langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$ then

$$\begin{aligned} &\langle \mathcal{I}_{name}, \mathcal{S}_{name}, O_{name} \rangle := \\ &\quad \langle \mathcal{I}_0, \langle \mathcal{N}', (\Psi'_D \cup \{O_i \xrightarrow{\lambda} N \mid N \in \mathcal{I}_{n+2} \wedge 1 \leq i \leq n\}) \uplus \Psi'_T \rangle, O_{n+2} \rangle \end{aligned}$$

As translation information we store that the CASL-specification of name SN has the top-level output node O_{name}, and add the information about the translation of the parameter specifications ($\langle \mathcal{I}_i, O_i \rangle$) with $1 \leq i \leq n$ as well the output node O_0 of the given part. This is used for the translation of instantiations of SN.

The final result of the translation of the named specification definition is then

$$\begin{aligned} &\tau_{part}(\textbf{spec } SN[SP_1] \ldots [SP_n] \textbf{ given } SP'_1, \ldots, SP'_m = SP \textbf{ end}, \mathcal{S}, \mathcal{P}) \\ &\qquad := \langle \mathcal{S}_{name}, \mathcal{P} \cup [SN, O_{name}, (\langle \mathcal{I}_1, O_1 \rangle, \ldots, \langle \mathcal{I}_n, O_n \rangle), O_0] \rangle \end{aligned}$$

4.3 Views

A named view in CASL is of the general form

$$\textbf{view } VN\ [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m\ :\ SP \textbf{ to } SP'\ =\ SM \textbf{ end}. \quad (1)$$

$[SP_1] \ldots [SP_n]$ **given** $SP_1, \ldots, SP_m$: SP represents a specification similar to the definition of named specifications. The view constitutes the proof obligation that the models of this specification can be mapped to models of SP' using the signature morphism given by SM.

To translate a named view we translate a dummy named specification

$$\textbf{spec}\ SN\ [SP_1] \ldots [SP_n]\ \textbf{given}\ SP_1, \ldots, SP_m\ =\ SP$$

which results as described in the previous paragraph in

$$\langle \mathcal{S}_{name}, \mathcal{P} \cup [SN, O_{name}, (\langle \mathcal{I}_1, O_1 \rangle, \ldots, \langle \mathcal{I}_n, O_n \rangle), O_0] \rangle.$$

Next we translate the structured specification SP' by

$$\langle \mathcal{I}', \mathcal{S}', O' \rangle := \tau(SP', \mathcal{S}_{name}, \mathcal{P})$$

and add a global theorem link from O_{name} to O' with the morphism SM. The final result of τ_{part} on (1) consists of this new development graph and the parameter information for the named view. Let $\mathcal{S}' = \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$ then

$$\begin{array}{l} \tau_{part}(\textbf{view}\ VN\ \ldots \textbf{end}, \mathcal{S}, \mathcal{P}) := \\ \quad \langle \langle \mathcal{N}', \Psi'_D \uplus (\Psi'_T \cup \{O_{name} \xrightarrow{SM} O'\}) \rangle, \\ \quad\ \mathcal{P} \cup \{[VN, (O_{name}, O'), (\langle \mathcal{I}_1, O_1 \rangle, \ldots, \langle \mathcal{I}_n, O_n \rangle), O_0]\} \rangle \end{array}$$

4.4 Structured Specifications

We now define τ for basic specifications and each of the structuring operations in CASL.

Basic Specifications. A basic specification is a pair (Σ, Φ) of a signature Σ and a set of second-order logic axioms Φ. We create a new node in the development N with local signature $\Sigma_l^N := \Sigma$ and local axioms $\Phi_l^N := \Phi$ and add it to the development graph. The node N is both the node where the actual visible environment shall be imported into as well as the node that contains the visible environment "after" parsing the basic specification.

$$\tau((\Sigma, \Phi), \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) := \langle \{N\}, \langle \mathcal{N} \cup \{N\}, \Psi_D \uplus \Psi_T \rangle, N \rangle$$

Translations. A translation is of the form SP **with** SM, where SP is a structured specification and SM a symbol morphism. Let

$$\langle \mathcal{I}', \mathcal{S}', O' \rangle := \tau(SP, \mathcal{S}, \mathcal{P})$$

with $\mathcal{S}' = \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$ and let N be a new node in $\mathcal{N}'$ with empty local signature and axioms. We add this new node to the new development graph and import the top-level node O' from SP into N via a global definition link with morphism SM.

$$\tau(SP\ \textbf{with}\ SM, \mathcal{S}, \mathcal{P}) := \langle \mathcal{I}', \langle \mathcal{N}' \cup \{N\}, (\Psi'_D \cup \{O' \xrightarrow{SM} N\}) \uplus \Psi'_T \rangle, N \rangle$$

Extensions. There are two kinds of extensions in CASL, namely SP **then** SP' and SP **then %implies** SP'. The first is a *regular* extension of SP by SP', while the second denotes a *conservative* extension, i.e. it is in fact a conjecture that all axioms in SP' are theorems in the theory of SP.

– *Regular Extensions* are translated as follows. Let

$$\begin{aligned}\langle \mathcal{I}', \mathcal{S}', O' \rangle &:= \tau(SP, \mathcal{S}, \mathcal{P}) \\ \langle \mathcal{I}'', \langle \mathcal{N}'', \Psi''_D \uplus \Psi''_T \rangle, O'' \rangle &:= \tau(SP', \mathcal{S}', \mathcal{P})\end{aligned}$$

then

$$\tau(SP \textbf{ then } SP', \mathcal{S}, \mathcal{P}) := \langle \mathcal{I}', \langle \mathcal{N}'', (\Psi''_D \cup \{O' \xrightarrow{\lambda} I'' \mid I'' \in \mathcal{I}''\}) \uplus \Psi''_T \rangle, O'' \rangle$$

– *Conservative Extensions:* The CASL-semantics requires from a conservative extension SP **then %implies** SP' that SP' is a *basic specification* without local signature. Thus, let

$$\begin{aligned}\langle \mathcal{I}, \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, O \rangle &:= \tau(SP, \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) \text{ and} \\ \langle \{N\}, \langle \mathcal{N}' \cup \{N\}, \Psi'_D \uplus \Psi'_T \rangle, N \rangle &:= \tau(SP', \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, \mathcal{P})\end{aligned}$$

Then the translation of this conservative extension consists of adding the local axioms of N as local lemmata to O and returning $\langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$.

$$\tau(SP \textbf{ then \%implies } SP', \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) := \langle \mathcal{I}, \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, O \rangle$$

where O is the updated O.

Union. A union of specifications in CASL is of the form SP **and** SP'. Let

$$\begin{aligned}\langle \mathcal{I}, \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, O \rangle &:= \tau(SP, \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) \text{ and} \\ \langle \mathcal{I}', \langle \mathcal{N}'', \Psi''_D \uplus \Psi''_T \rangle, O' \rangle &:= \tau(SP', \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, \mathcal{P})\end{aligned}$$

In order to represent the union of the specifications, we add a new empty theory node N to $\mathcal{N}''$ and import both O and O' into N via global definition links.

$$\begin{aligned}\tau(SP \textbf{ and } SP', \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) := \\ \langle \mathcal{I} \cup \mathcal{I}', \langle \mathcal{N}'' \cup \{N\}, (\Psi''_D \cup \{O \xrightarrow{\lambda} N, O' \xrightarrow{\lambda} N\}) \uplus \Psi''_T \rangle, N \rangle\end{aligned}$$

The theory nodes to import the global environment is the union of both $\mathcal{I}$ and $\mathcal{I}'$, while the visible environment "after" the union is the global signature of the new node N.

Closed Specifications. They are of the form **closed**$\{SP\}$. The semantics is that the global environment is not visible inside SP, but shall still be visible "after" **closed**$\{SP\}$ together with the environment generated from SP. Thus, the translation of the closed specification consists in creating a new empty node N, import the environment from SP into N via a global definition link, and returning N has both the import and output node for the global environment. Thus, if $\langle \mathcal{I}, \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle, O \rangle := \tau(SP, \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P})$, then

$$\tau(\textbf{closed}\{SP\}, \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle, \mathcal{P}) := \langle N, \langle \mathcal{N}' \cup \{N\}, (\Psi'_D \cup \{O \xrightarrow{\lambda} N\}) \uplus \Psi'_T \rangle, N \rangle$$

Actualization. An actualization in CASL is of the general form

$$SN\ [SP_1\ \textbf{fit}\ SM_1]\dots[SP_n\ \textbf{fit}\ SM_n].$$

Its semantics is that, the formal parameter of the formerly declared named specification SN are instantiated with the SP_i and the "**given**"-specifications of SN is imported into the actual parameters SP_i. This is only sound if the actual parameter fit the formal parameter theories modulo the morphisms SM_i. A parameter information for SN is $[SN, O, (\langle \mathcal{I}'_1, O'_1\rangle, \dots, \langle \mathcal{I}'_n, O'_n\rangle), O_I]$, where O is the top-level theory for SN, $\langle \mathcal{I}'_i, O'_i\rangle$ the information about input and output theories of the parameter theories, and O_I the top-level theory node that is imported into the parameters. Given this parameter information for the named specification SN, let

$$\begin{aligned}\langle \mathcal{N}_0, \Psi_D^0 \uplus \Psi_T^0\rangle &:= \langle \mathcal{N}, \Psi_D \uplus \Psi_T\rangle\\ \tau(SP_i, \langle \mathcal{N}_{i-1}, \Psi_D^{i-1} \uplus \Psi_T^{i-1}\rangle, \mathcal{P}) &:= \langle \mathcal{I}_i, \langle \mathcal{N}_i, \Psi_D^i \uplus \Psi_T^i\rangle, O_i\rangle\\ &\qquad \text{for all } 1 \le i \le n\end{aligned}$$

Then we import the "**given**" environment theory O_I into each theory in $\mathcal{I}_i$, for all $1 \le i \le n$:

$$\langle \mathcal{N}_n, (\Psi_D^n \cup \{O_I \xrightarrow{\lambda} I \mid I \in \bigcup_{i=1}^{n} \mathcal{I}_i\}) \uplus \Psi_T^n\rangle$$

We further encode the soundness condition required by **fit** by introducing global theorem links from each O'_i to O_i with morphism SM_i:

$$\langle \mathcal{N}_n, (\Psi_D^n \cup \{O_I \xrightarrow{\lambda} I \mid I \in \bigcup_{i=1}^{n} \mathcal{I}_i\}) \uplus (\Psi_T^n \cup \{O'_i \xrightarrow{SM_i} O_i \mid 1 \le i \le n\})\rangle$$

Finally, we create the node N_I to encode the instantiated theory: This node imports globally the top-level node N for SN, as well as the top-level nodes O_i of the actual parameter theories.

$$\begin{aligned}\langle \mathcal{N}_n \cup \{N_I\},\ &(\Psi_D^n \cup \{O_I \xrightarrow{\lambda} I \mid I \in \textstyle\bigcup_{i=1}^{n} \mathcal{I}_i\} \cup \{N \xrightarrow{SM} N_I, O_1 \xrightarrow{\lambda} N_I, ..., O_n \xrightarrow{\lambda} N_I\})\rangle\\ &\uplus(\Psi_T^n \cup \{O'_i \xrightarrow{SM_i} O_i \mid 1 \le i \le n\})\end{aligned}$$

where $SM := \bigcup_{i=1}^{n} SM_i$.

This completes the definition of the translation of structured specification.

4.5 Fitting Views

A fitting view is of the form $VN\ [SP_1]\dots[SP_n]$, where VN is the name of a view. The translation of this fitting view is analogously to the translation of an actualization of a named specification, except that an additional global theorem link from the actualized theory to the top-level theory node obtained for the target SP of the view SN is inserted.

5 Difference Analysis and Basic Operations

Due to its evolutionary nature, (formal) software development can be seen as a chain of specifications $Spec_1, Spec_2, \ldots$ which corresponds to a chain of development graphs $DG_1, DG_2, \ldots$ such that DG_i is the logical representation of the specification $Spec_i$. Working on the verification side we try to verify the various proof obligations within a particular development graph, say DG_i. Changing the specification to $Spec_{i+1}$ and compiling it into its logical representation DG_{i+1}, we loose all information about previous proof work, which is stored in DG_i, at first. Hence, the idea is to incrementally adjust DG_i and its annotated proofs until the resulting development graph DG_{i+1} denotes a logical representation of $Spec_{i+1}$. Two problems have to be solved to implement this approach:

First, we need a set of operations which allow us to modify development graphs in such a way that as much proof work as possible can be reused from the previous development graph. We call these operations, that manipulate individual links, theories or axioms, *basic operations*.

Second, we have to compute the differences between two specifications $Spec_i$ and $Spec_{i+1}$ and translate these differences into a sequence of basic operations to be performed on the development graph DG_i in order to obtain DG_{i+1}.

5.1 Basic Operations

To allow for a reuse of proof work, basic operations have to be as granular as possible. Since development graphs consists of nodes and links, basic operations allow one to modify single nodes or links. In principle each of these individual objects can be inserted, deleted or modified. As nodes are composed of a local signature and local axioms, the modification of nodes is done by insertion, deletion or modification of signature entries or local axioms. Formally the set of basic operations consists of the following functions that take, between others, a development graph $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ as argument and return a new development graph $\mathcal{S}'$:

Nodes: $ins_{node}(N, \mathcal{S})$ inserts a new (isolated) node N to $\mathcal{N}$, and $del_{node}(N, \mathcal{S})$ removes a node N from $\mathcal{N}$ and deletes also all links in Ψ_D and Ψ_T connected to N.

Links: $ins(N, M, \sigma, Type, \mathcal{S})$ inserts a link to Ψ as a global/local definition/theorem link depending on the value of $Type$. $del(L, \mathcal{S})$ removes the link L from $\Psi_D \uplus \Psi_T$, and $ch(L, \sigma, \mathcal{S})$ replaces the morphism of the link L by σ.[2]

Local Signature: $ins_{sig}(f, N, \mathcal{S})$ inserts the symbol f into the local signature of N, where f can be either a sort, a constant, or a function. $del_{sig}(f, N, \mathcal{S})$ removes the symbol f from the local signature of N.

Local Axioms: $ins_{ax}(N, Ax, \mathcal{S})$ inserts the local axiom Ax into the node N, $del_{ax}(f, N, \mathcal{S})$ deletes the local axiom Ax from the node N. $ch_{ax}(N, Ax, Ax', \mathcal{S})$ replaces the local axiom Ax by the new local axiom Ax' in the node N.

[2] There are no operations to change the source or target node of a link. In this case the old link must be deleted and a new link is inserted.

For each basic operation the manner how it affects the development graph is known. This knowledge is exploited by the proof transformation techniques, that adapt the proofs of old global proof obligations to the new global proof obligations. We will describe these techniques in the Sect. 6.

Starting with a legal development graph, the application of basic operations may result in inconsistent intermediate states. A typical example is the insertion of a new function symbol into the source node of a link. Then in general, the morphism attached to the link has to be adjusted to cope with the new symbol. Therefore we allow for intermediate inconsistent states of the development graph and delay the update of the proof work until we reach a consistent state which is indicated by calling a special *update*-function initiating a consistency check and a propagation of the proof work done so far.

5.2 Computing Differences

When computing differences between specifications, the question arises how to define the granularity up to which differences are determined between the old and the new development graph. Note that along a scale of granularity levels for difference analysis the worst granularity level is the one only stating that the whole global proof obligation changed, in which case the proof transformation consists of redoing the whole proof, whereby any information about established conjectures are lost.

The overall aim is to enable the preservation of as many validated conjectures during the transformation of the old proof to the new development graph. The recorded information establishing the validity of a conjecture consists of proofs for those conjectures. However, not every theorem prover returns a proof object. In that case, we must assume that any axiom available at prove time might have been used during the proof. Thus, the information about a proof contains at least a set of axioms. If any of those is deleted or changed, the proof gets invalid. The implication is that we have to determine the difference between the old and new development graph at least on the level of axioms.

The axioms are build from the available signature symbols, like sorts, constants and functions. In order to maintain a sound development graph, we must also be able to determine the differences between signatures. As presented in Sect. 3, the signature of some node is defined from the local signature defined on that node and the signatures of the nodes imported via definition links, after application of the morphism attached to those links.

To determine the differences of signatures and axioms between two development graphs requires first to define an equivalence relation between graphs that identifies nodes and links. This problem has no optimal solution and hence we rely on some heuristics checking their equivalence. In principle two nodes are equivalent if their local signature and axioms are equal as well as their respective incoming definition links. However, this equivalence relation is to strict for our purpose, since if we added or deleted an axiom to some node, its old and new version are not identified. Thus, instead of performing an equality check, we perform a similarity check on nodes, that is based on the number of shared local signature

symbols as well as the similarity of the incoming definition links. Applying that similarity check results in an equivalence relation associating nodes and links of the old to nodes and links in the new development graph.

The equivalence relation is the basis to determine the differences between both graphs. From it we determine (1) which nodes have been deleted or added, (2) which local signature symbols and axioms have been deleted or added to some node, and (3) how the morphisms of links have changed.

5.3 Heuristic Determination of Similarities

In this Section we describe the heuristic implemented in MAYA which is used to compute the similarities between two versions of development graphs. The conducted experiments showed that it is sufficiently reliable for our needs. The similarity is expressed by a *mapping* among theory nodes and links. Formally, given two development graphs $\langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ and $\langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$, the mapping is a triple $(\mapsto_N, \mapsto_D, \mapsto_T)$, such that $\mapsto_N \colon \mathcal{N} \hookrightarrow \mathcal{N}'$, $\mapsto_D \colon \Psi_D \hookrightarrow \Psi'_D$, and $\mapsto_T \colon \Psi_T \hookrightarrow \Psi'_T$ are partial functions.

The heuristic to construct such a mapping works as follows. We start with partial functions $\mapsto_N, \mapsto_D$, and $\mapsto_T$ that have an empty domain. Then, for each node N in $\mathcal{N}$ we determine the "most similar node" in $\mathcal{N}'$ that is not in the image of $\mapsto_N$. If there is such a node N', we extend $\mapsto_N$ by setting $\mapsto_N (N) := N'$; otherwise N has no similar theory in $\mathcal{N}'$. Finally, for each link in Ψ_D (resp. Ψ_T) we determine the "most similar link" in Ψ'_D (resp. Ψ'_T), and extend $\mapsto_D$ (resp. $\mapsto_T$) accordingly, if such a link exists.

The whole heuristic is based on the notion of similarities of two nodes and links. These two notions are defined by mutually recursion and make use of already established mappings between old and new theory nodes and links. The similarity of nodes and links is some value from [0..1] and is denoted by $\mathit{Similarity}(N, N')$ for nodes and $\mathit{Similarity}(l, l')$ for links. The values are estimated as follows:

- *Similarity of Nodes.* Let $N \in \mathcal{N}$ and $N' \in \mathcal{N}'$ be two nodes and $(\mapsto_N, \mapsto_D, \mapsto_T)$ the actual mapping.
  - If $\mapsto_N (N) = N'$, then $\mathit{Similarity}(N, N') := 1$.
  - If both N and N' have a *defined name*, like for example if they are both the top-level theory nodes obtained for some named CASL-specification, then if those names are equal, then $\mathit{Similarity}(N, N') := 1$, otherwise $\mathit{Similarity}(N, N') := 0$.
  - If none of them has a defined name, then we take the average of on the one hand the similarity between the local sorts in Σ_l^N and $\Sigma_l^{N'}$, and on the other hand the similarity between the sets of incoming definition links into N and those for N'.
  - Otherwise, $\mathit{Similarity}(N, N') := 0$.
- *Similarity of Links.* We illustrate the computation for definition links, i.e. links from Ψ_D and Ψ'_D. The computation is analogously for theorem links. Let $l \in \Psi_D$ and $l' \in \Psi'_D$ be two definition links and $(\mapsto_N, \mapsto_D, \mapsto_T)$ the actual mapping.

- If one is a global link while the other is not, then $Similarity(l, l') := 0$.
- Otherwise, if they have the same morphism, then we set the similarities of the two links to be the average of the similarity between the source nodes and the similarity of the target nodes.
- If they don't have the same morphism, then the similarity is 0.

6 Maintaining Proof Work

The development graph represents a justification-based truth maintenance system for structured specifications. Based on underlying theorem provers it provides justifications for proof obligations (encoded as theorem links) and is able to remember and adjust derivations which were computed previously. There are two different types of justifications corresponding to the verification in-the-large and to the verification in-the-small which both have to be updated each time the graph is changed. In the following we describe this propagation of proof work for the verification in-the-large and the verification in-the-small separately.

6.1 The DG-Calculus

The theory of a node N depends on theories of all nodes connected to N via (global) definition links. Local definition links hide the theories of underlying subnodes. The next definition specifies possible paths to include the theory or the local axioms of the source node to the theory of the target node.

Definition 3. *Let Ψ be a set of links.*

- *Ψ contains a* **global path** $N_1 \overset{\sigma}{\blacktriangleright}_{\Psi} N_k$ *from N_1 to N_k via a morphism σ if there is either a sequence of links $N_1 \overset{\sigma_1}{\blacktriangleright} N_2, N_2 \overset{\sigma_1}{\blacktriangleright} N_3 \ldots N_{k-1}, \overset{\sigma_{k-1}}{\blacktriangleright} N_k$ in Ψ with $\sigma = \sigma_1 \circ \ldots \circ \sigma_{k-1}$ or $N_1 = N_k$ and σ is the identity function.*
- *Ψ contains a* **local path** $N_1 \overset{\sigma}{\blacktriangleright}_{\Psi} N_k$ *from N_1 to N_k via a morphism σ if there is a sequence of links $N_1 \overset{\sigma_1}{\longrightarrow} N_2, N_2 \overset{\sigma_1}{\blacktriangleright} N_3 \ldots N_{k-1}, \overset{\sigma_{k-1}}{\blacktriangleright} N_k$ in Ψ with $\sigma = \sigma_1 \circ \ldots \circ \sigma_{k-1}$.*

Given a development graph $\langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$, the definition links Ψ_D are used to specify the semantics, i.e. theory, of the individual nodes. Ψ_T constitutes the proof obligations inside the graph. In the following we define when a development graph satisfies these proof obligations:

Definition 4. *Let $\mathcal{S} = \langle \mathcal{N}, \Psi \rangle$ be a development graph and $\Delta \subseteq \Psi$ be acyclic. Δ* **satisfies** *a link $M \overset{\sigma}{\blacktriangleright} N \in \Psi$ (or $M \overset{\sigma}{\longrightarrow} N \in \Psi$ resp.) iff $\sigma(Th_\Delta(M)) \subseteq Th_\Delta(N)$ (or $\sigma(\Phi_l^M) \subseteq Th_\Delta(N)$ resp.). Δ* **satisfies** *a set Γ of links if it satisfies all elements in Γ.*

A development graph $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ is **verified** *iff Ψ_D satisfies Ψ_T.*

A global definition link includes the theory of the source node into the theory of the target node while a local definition link includes only the local axioms of the source node. Due to the $\vdash$-translation property of the underlying entailment relation, any global definition link starting at the target node of such a link will

export this imported theory or axioms in turn to other theories. Theorem links which are satisfied by the definition links can be treated in the same manner as definition links:

Lemma 1. *Let $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ be a development graph and let Ψ_D satisfy a set of links Δ. Then the following holds:*

1. $N \overset{\sigma}{\blacktriangleright}_{\Psi_D \uplus \Delta} M$ *implies* $\sigma(Th(N)) \subset Th(M)$ *and*
2. $N \overset{\sigma}{\rightarrowtriangle}_{\Psi_D \uplus \Delta} M$ *implies* $\sigma(\Phi_i^N) \subset Th(M)$

Proof. We sketch the proof for global paths which is done by induction on the length of the path:

Base Case: If the path is empty, then σ is the identity and $N = M$. Thus $\sigma(Th(N)) \subset Th(M)$ holds trivially.
Step Case: As an induction hypothesis we assume that if $N \overset{\sigma'}{\blacktriangleright}_{\Psi_D \uplus \Delta} K$ then $\sigma'(Th(N)) \subset Th(K)$. Let $K \overset{\sigma''}{\longrightarrow} M \in \Psi_D \uplus \Delta$. Thus, $\sigma''(Th(K)) \subset Th(M)$ (Def. 2 or Def. 4 resp.) and therefore $\sigma''(\sigma'(Th(N))) \subset Th(M)$. □

In order to verify a development graph we introduce a calculus $\mathcal{DG}$ operating on links to perform a so-called *verification in-the-large* and providing a *local decomposition rule* to establish elementary relations between theories by usual theorem proving, which we call *verification in-the-small*.

Definition 5 (Calculus $\mathcal{DG}$). *The calculus $\mathcal{DG}$ is a sequent-style calculus. Sequents are of the form $\Gamma \vdash \Delta$, where Γ, Δ are sets of links. A sequent $\Gamma \vdash \Delta$ holds iff Γ satisfies Δ. The sequent calculus rules of $\mathcal{DG}$ are:*

Axiom (AX)*:*
$$\frac{}{\Gamma \vdash \emptyset}$$

Global Decomposition (GD)*:*
$$\frac{\Gamma \vdash N \xrightarrow{\sigma} M, \bigcup_{K \xrightarrow{\rho} N \in \Gamma} \{K \xrightarrow{\sigma \circ \rho} M\}, \bigcup_{K \xrightarrow{\rho} N \in \Gamma} \{K \xrightarrow{\sigma \circ \rho} M\}, \Delta}{\Gamma \vdash N \xrightarrow{\sigma} M, \Delta}$$

Local Decomposition (LD)*:*
$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash N \xrightarrow{\sigma} M, \Delta} \quad \textit{if for all } \phi \in \Phi_i^N : \sigma(\phi) \in Th_\Gamma(M)$$

Global subsumption (GS)*:*
$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash N \xrightarrow{\sigma} M, \Delta} \quad \textit{if } N \overset{\sigma}{\blacktriangleright}_{\Gamma \cup \Delta} M$$

Local subsumption (LS)*:*
$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash N \xrightarrow{\sigma} M, \Delta} \quad \textit{if } N \overset{\sigma}{\rightarrowtriangle}_{\Gamma \cup \Delta} M$$

Theorem 1. *Let $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ be a development graph and $\Delta \subseteq \Psi_T$. Then, $\Psi_D \vdash \Delta$ is derivable in the deduction system $\mathcal{DG}$ iff Ψ_D satisfies Δ.*

Proof (of Theorem 1).

Soundness: We induce on the length n of the deduction:

Base Case: let $n = 1$. Thus, $\Delta = \emptyset$ and $\mathcal{S}$ satisfies Δ trivially.
Step Case: let $n > 1$ and $\Psi_D \vdash \Delta'$ be the immediate predecessor of $\Psi_D \vdash \Delta$. As induction hypothesis we assume that $\mathcal{S}$ satisfies Δ'. We do a case split according to the applicable rules:

GD: Hence $N \xrightarrow{\sigma} M \in \Delta'$, $K \xrightarrow{\sigma\circ\rho} M \in \Delta'$ for all $K \xrightarrow{\rho} N \in \Psi_D$ and $K \xrightarrow{\sigma\circ\rho} M \in \Delta'$ for all $K \xrightarrow{\rho} N \in \Psi_D$. Since $\mathcal{S}$ satisfies Δ', we know that $\sigma(\Phi_l^N) \subseteq Th(M)$, $\sigma(\rho(Th(K))) \subseteq Th(M)$ for all $K \xrightarrow{\rho} N \in \Psi_D$ and $\sigma(\rho(\Phi_l^K)) \subseteq Th(M)$ for all $K \xrightarrow{\rho} N \in \Psi_D$. Thus, from the $\vdash$-translation property of the underlying entailment relation we get

$$\left[\sigma(\Phi_l^N) \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \sigma(\rho(Th(K))) \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \sigma(\rho(\Phi_l^K))\right]^{\vdash_{\Sigma^N}} \subseteq Th(M).$$

Hence, $\sigma(\left[\Phi_l^N \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \rho(Th(K)) \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \rho(\Phi_l^K)\right]^{\vdash_{\Sigma^M}}) \subseteq Th(M)$ holds due to the $\vdash$-translation property, i.e. $\sigma(Th(N)) \subseteq Th(M)$ and thus Ψ_D satisfies $N \xrightarrow{\sigma} M$.
LD: $\sigma(\phi) \in Th(M)$ for all $\phi \in \Phi_l^N$ implies that Ψ_D satisfies $N \xrightarrow{\sigma} M$ and thus $\mathcal{S}$ satisfies Δ.
GS: Since $N \xrightarrow{\sigma}_{\Psi_D \cup \Delta'} M$ holds and $\mathcal{S}$ satisfies Δ' we know that $\sigma(Th(N)) \subset Th(M)$ holds, i.e. $\mathcal{S}$ satisfies $N \xrightarrow{\sigma} M$.
LS: Since $N \xrightarrow{\sigma}_{\Psi \cup \Delta'} M$ and $\mathcal{S}$ satisfies Δ' we know that $\sigma(\Phi_l^N) \subset Th(M)$ holds, i.e. $\mathcal{S}$ satisfies $N \xrightarrow{\sigma} M$.

Completeness: Suppose, Ψ_D satisfies Δ. Since the development graph $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ is acyclic with respect to Ψ_D we define the *depth* of a node $N \in \mathcal{N}$ as the length of the longest path of links in Ψ_D from some leaf node in $\mathcal{N}$ to N. We induce on the multiset *depths*(Δ) of depths of the source nodes in Δ.

Base Case: If *depths*(Δ) $= \emptyset$ then $\Delta = \emptyset$ and $\Psi_D \vdash \emptyset$ holds by rule AX.
Induction Step: Let *depths*(Δ) $\neq \emptyset$. As an induction hypothesis suppose the conjecture holds for all Δ' which are smaller than Δ with respect to the multiset-ordering on *depths*.

- Let $N \xrightarrow{\sigma} M \in \Delta$ with $depth(N) = max(depths(\Delta))$. Since Ψ_D satisfies $\Delta - \{N \xrightarrow{\sigma} M\}$, applying the induction hypothesis yields $\Psi \vdash \Delta - \{N \xrightarrow{\sigma} M\}$. As Ψ_D satisfies $N \xrightarrow{\sigma} M$, we know that $\sigma(\Phi_l^N) \subset Th(M)$ holds and apply rule LD to deduce finally $\Psi_D \vdash \Delta$.
- Let $N \xrightarrow{\sigma} M \in \Delta$ with $depth(N) = max(depths(\Delta))$. For all links $K \xrightarrow{\rho} N \in \Psi_D$ $\sigma(\rho(Th(K))) \subset Th(M)$ holds. Analogously for all links $K \xrightarrow{\rho} N \in \Psi_D$ holds $\sigma(\rho(\Phi_l^K)) \subset Th(M)$. Thus, $\mathcal{S}$ satisfies both $\bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\}$ and $\bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\}$. Applying the induction hypothesis yields $\Psi_D \vdash (\Delta - \{N \xrightarrow{\sigma} M\}) \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\} \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\}$. Since $\mathcal{S}$ satisfies also $N \xrightarrow{\sigma} M$ we use the argumentation of the first case to deduce $\Psi_D \vdash (\Delta - \{N \xrightarrow{\sigma} M\}) \cup \{N \xrightarrow{\sigma} M\} \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\} \cup \bigcup_{K \xrightarrow{\rho} N \in \Psi_D} \{K \xrightarrow{\sigma\circ\rho} M\}$ and apply rule GD to derive $\Psi_D \vdash \Delta$. □

The $\mathcal{DG}$-calculus is based on an oracle to check if $\sigma(\phi) \in Th_{\Psi_D}(M)$ holds. In general $Th_{\Psi_D}(M)$ is an infinite set of formulas and we need a finite axiomatiza-

tion for it. It is well-known that structured specifications excluding hiding[3] are flatable. The following lemma describes the finite axiomatization of the theory of a node:

Lemma 2. *Let $\mathcal{S} = \langle \mathcal{N}, \Psi \rangle$ be a development graph and let the* **axiomatization** *of some node $N \in \mathcal{N}$* **relative** *to some $\Delta \subseteq \Psi$, Δ acyclic, be defined by*

$$\Phi^N_\Delta = \Phi^N_l \cup \bigcup_{K \xrightarrow{\sigma} N \in \Delta} \sigma(\Phi^K_\Delta) \cup \bigcup_{K \xrightarrow{\sigma} N \in \Delta} \sigma(\Phi^K_l)$$

Then, $Th_\Delta(N) = \left[\Phi^N_\Delta\right]^{\vdash_{\Sigma^N}}$ holds for all $N \in \mathcal{N}$.

Proof. Directly from Def. 2 and the $\vdash$-translation property of the underlying entailment relation. □

To verify the proof obligations on local theorem links, which we call *verification in-the-small*, we make use of standard theorem provers like Isabelle [17] or Inka 5.0 [1]. The reader is referred to [3] for a description how development graph and theorem provers are technically connected.

6.2 Verification In-the-large

Verification in-the-large is concerned with the reduction of the overall problem of verifying an development graph $\mathcal{S}$ to the problem of proving as few as possible proof obligations denoted by local theorem links. Verification-in-the-large is done with the help of the $\mathcal{DG}$-calculus. Obviously, applying global and local subsumption rules as often as possible will reduce the number of arising proof obligations in-the-small. To support the maintenance of the proofs in-the-large, MAYA provides explicit proof objects for the $\mathcal{DG}$-calculus. Theorem links are annotated with explicit proof objects, which are instances of the $\mathcal{DG}$-calculus rules. Each $\mathcal{DG}$-calculus rule reduces the proof of a theorem link to the problem of proving a set of other theorem links. Thus, the proof object of a theorem link is distributed through the development graph and only the first inference step, the so-called *local proof object*, is stored at the theorem link while the remaining part always coincides with proof objects of other theorem links.

Definition 6. *Let $\psi = N \xrightarrow{\sigma} M$ then*

- *$pr_\psi := GD(\psi_0, \langle \psi'_1, \ldots, \psi'_n \rangle, \langle \psi''_1, \ldots, \psi''_m \rangle)$ is a local proof object. pr_ψ is locally valid iff $\psi_0 = N \xrightarrow{\sigma} M$, $\{\psi'_1, \ldots, \psi'_n\} = \bigcup_{K \xrightarrow{\rho} N \in \Gamma} \{K \xrightarrow{\sigma \circ \rho} M\}$ and $\{\psi''_1, \ldots, \psi''_m\} = \bigcup_{K \xrightarrow{\rho} N \in \Gamma} \{\psi \xrightarrow{\sigma \circ \rho} M\}$*
- *$pr_\psi := GS(\psi_1, \ldots, \psi_n)$ is a local proof object. pr_ψ is locally valid iff $\psi_1, ..., \psi_n$ constitutes a relation $N \rightarrowtail^{\sigma} M$.*

[3] See [14] for an extension of development graphs by hiding which translates proof obligations in theories based on hiding to proof obligations in theories without hiding.

Let $\psi = N \xrightarrow{\sigma} M$ *then*

- $pr_\psi := LS(\psi_1, \ldots, \psi_n)$ *is a local proof object.* pr_ψ *is locally valid iff* $\psi_1, ..., \psi_n$ *constitutes a relation* $N \overset{\sigma}{\rightarrowtail} M$.
- $pr_\psi := LD(\sigma, (Ax_1, \Phi_1), \ldots, (Ax_k, \Phi_k))$ *is a proof object where each* Φ_i *is either an atom NoProof, ProofExists or a set of triples* (τ, K, Ω) *with* $\Omega \subset \Phi_l^K$. pr_ψ *is locally valid iff for all* (Ax_i, Φ_i) *with* $1 \leq i \leq n$, $(\bigcup_{(\tau,K,\Omega)\in\Phi_i} \tau(\Omega)) \vdash \sigma(Ax_i)$ *and for all triple* $(\tau, K, \Omega) \in \Phi_i$ $K \overset{\tau}{\rightarrowtail} M$ *holds.*

$\Psi(pr_\psi)$ *is defined as the set of all links occurring in the proof object* pr_ψ *of* ψ. $\Psi^*(pr_\psi)$ *denotes the transitive closure of* $\Psi(pr_\psi)$ *and is defined by* $\Psi^*(pr_\psi) = \Psi(pr_\psi) \cup \bigcup_{\psi' \in \Psi(pr_\psi)} \Psi^*(pr_{\psi'})$.

Lemma 3. *Let* $\mathcal{S} = \langle \mathcal{N}, \Psi_D \uplus \Psi_T \rangle$ *be a development graph. If there are locally valid proof objects* pr_ψ *with* $\psi \notin \Psi^*(pr_\psi)$ *for all* $\psi \in \Psi_T$ *then* Ψ_D *satisfies* Ψ_T.

Proof. Since $\psi \notin \Psi^*(pr_\psi)$ holds for all $\psi \in \Psi_T$ there is a partial ordering $<$ on Ψ_T with $\psi' < \psi$ iff $\psi' \in \Psi^*(pr_\psi)$. We can extend such a partial ordering to a total ordering $\ll$ on Φ_T. It is an easy inductive argument that we can construct a $\mathcal{DG}$-calculus proof in the following way: we start with the problem of proving $\Psi_D \vdash \Psi_T$ and apply the proof rule attached to the maximal element of Ψ_T wrt. $\ll$. Since the proof object is locally valid the rule is applicable and we have reduced the problem to a problem of proving $\Psi_D \vdash \Psi_T \setminus \{\psi\}$. Iterating this approach by choosing always the maximal element of the set of remaining theorem links we end up in the trivial case of proving $\Psi_D \vdash \emptyset$.

Verification in-the-large is concerned with the problem of creating and maintaining local proof objects of the types GD, GS and LS such that each of these local proof objects is locally valid and such that the proof object of a link ψ does not depend on itself, i.e. $\psi \notin \Psi^*(pr_\psi)$. The problem of maintaining LD-proof objects is discussed in section 6.3. We call a development graph verified in-the-large if and only if all GD, GS, LS-proof objects are locally valid and do not contain cycles (i.e. $\psi \notin \Psi^*(pr_\psi)$).

Starting with an empty development, which is trivially verified, the graph is manipulated by using basic operations like for instance the insertion, deletion, or change of links or axioms. After a sequence of basic operations (updating the development graph according to the change of specification made by the user) the proof objects are adapted to the needs of the actual graph. Hence, each subsequent development graph is verified reusing the old proof objects annotated in the former development graph.

To describe the update-process, assume now that we manipulated a verified development graph with the help of a sequence of basic operations. To establish the validity of the resulting development graph we perform the following steps:

Checking GD-Proof Objects: In the first phase, existing GD-proof objects are updated to be locally valid proof objects. Starting at the top-level theories (like LIST in our example), we traverse the graph according to the depth of the theories. Reasons for an invalidated GD-proof object $pr_\psi = GD(\psi_0, \langle \psi'_1, \ldots, \psi'_n \rangle, \langle \psi''_1, \ldots, \psi''_m \rangle)$ are the change of the morphism of some link or the insertion or

deletion of definition links targeting at the source of the theorem link. In the first case we replace an inappropriate link by a link with an appropriate morphism. Either such a link already exists (e.g. as a definition link) or it is created and added to Ψ_T while it inherits the (invalid) proof object of the replaced link (this proof object will be fixed in the ongoing procedure). In case of insertion or removal of definition links, both link lists $\langle \psi'_1, \ldots, \psi'_n \rangle$ and $\langle \psi''_1, \ldots, \psi''_m \rangle$ in pr_ψ are updated accordingly. This, again, might result in the creation of new theorem links, which are again added to Ψ_T, or the deletion of theorem links from Ψ_T if they have been once created using the GD-rule and are of no use anymore (i.e. they do not occur in any proof object anymore).

Checking GS- and LS-Proof Objects: In the second phase, proof objects concerned with subsumption rules are checked for validity. For each of these proof objects $pr_\psi = GS(\psi_1, \ldots, \psi_n)$ we prove whether all links ψ_i do still exist and whether the morphism of the denoted path still coincides with the morphism of the theorem link ψ. If any of these conditions fails then the proof object is removed; otherwise the proof object pr_ψ is still locally valid.

Establish New Proof Object: In the third phase local proof objects are generated for theorem links which do not possess any proof object. Either these links have been newly created or their proof objects have been removed in an earlier stage of the procedure. Given a theorem link ψ, firstly we search for an application of the GS- or LS-rule. Thus, we search for a path starting at the source of ψ and ending at the target of ψ which coincides with ψ also in its morphism. In order to obtain an acyclic proof object, each link ψ' in the path has to satisfy the property $\psi \notin \Psi^*(pr_{\psi'})$. In practice we restrict this search for a path inside a graph in the following way: First, we do not search for paths in which a node is visited twice (although in general, running through a circle may result in a different overall morphism of the path). Second, proving a theorem link $K \xrightarrow{\sigma \circ \rho} M$ which was created while verifying a theorem link $N \xrightarrow{\sigma} M$ in presence of a definition link $K \xrightarrow{\rho} N$, we do not consider paths starting with this definition link. If we would find such a path then we could strip off the definition link to obtain a path for $N \xrightarrow{\sigma} M$ (but this was already checked during the verification of this link!). If we cannot find a suitable path to establish a GS- or LS-proof object, a GD-proof object for ψ is generated. This may cause the generation of new theorem links to be added to Ψ_T if no suitable links are already available in the graph.

To illustrate our approach, consider our example in Fig. 3. As we have started with the empty development graph there are no GD, GS or LS-proof objects to be updated and we continue with phase three:

Descending the graph according to the depth of the theories, we first establish a new proof object for the global theorem link from List to NatList. The GS-rule is not applicable since there is no corresponding global path from List to NatList. Hence, the GD-rule is applied which results in a proof object $GD(\psi_0, \langle \psi_{elem} \rangle, \langle\rangle)$. ψ_0 is the local definition link from List to NatList while ψ_{elem} is a newly generated theorem link from Elem to NatList (corresponding to the import of Elem in List). Similarly, we obtain a local proof

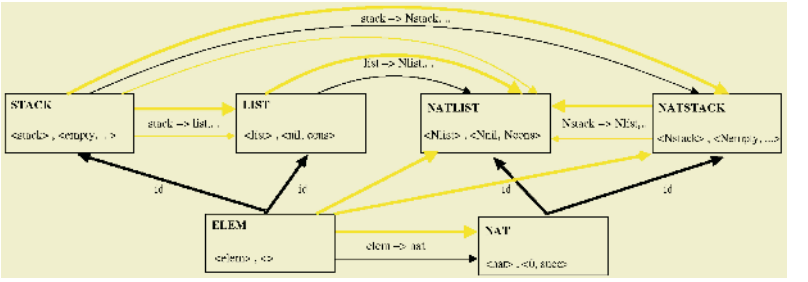


Fig. 3. Management of change for NatList.

object $GD(\psi'_0, \langle\psi'_{nat}\rangle, \langle\psi'_{stack}\rangle)$ for the global theorem link ψ' from NatStack to NatList. ψ'_0 is a newly generated local theorem link parallel to ψ', ψ'_{nat} is the global definition link from Nat to NatList. ψ'_{stack} is a newly generated local theorem link from Stack to NatList. Using the LS-rule ψ'_{stack} is proven by the path of (global) theorem links from Stack over List to NatList. Since NatStack has no local axioms, ψ'_{nat} is trivially proven using the LD-rule. Applying the GD-rule to the global theorem link from Stack to NatStack introduces a global theorem link from Elem to NatStack which is proven using the GS-rule by the path of global links from Elem over Nat to NatStack. At the end we are left with open proofs for the local theorem links from Stack to List and from Elem to Nat which are tackled by the verification in-the-small.

Suppose now, we change the graph structure by the insertion of a new theory Rel introducing a new symbol R and imported by Elem. Therefore all morphisms of the links from Elem, List and Stack to Nat, NatList and NatStack will be changed in order to incorporate an appropriate mapping of R. In the first phase of the revision process the GD-proof objects of the corresponding global theorem links are adjusted to incorporate the mapping of R. Additionally, the proof object of the global theorem link from Elem to Nat is changed to $GD(\psi'_0, \langle\psi_{REL}\rangle, \langle\rangle)$ where ψ_{REL} denotes a global theorem link from Rel to Nat (corresponding to the new definition link from Rel to Elem). In the second phase nothing has to be done since all GS- and LS-proof objects are still valid although the mapping have changed. In the third phase the new theorem link ψ_{REL} is proven with the help of the GD-rule which introduces a local theorem link from Rel to Nat denoting the proof obligations arising from the local axioms in Rel to be proven in Nat.

6.3 Verification In-the-small

Applying the local decomposition (LD-)rule gives rise to proof obligations that each local axiom of the source node mapped by the attached morphism of the link is a theorem of the target theory. To tackle these proof obligations, the system has to compute the axiomatization of the theory (ref. lemma 2) and to apply the morphism of the theorem link to the axioms of the source theory. Since the computation of the axiomatization is expensive the system caches the computed axiomatization of the target node. The axiomatization is annotated by

the information about the origin, applied morphisms and used paths of mapped axioms. Once the axiomatization of a different node is needed to tackle another proof obligation, the path information attached to the cached axiomatization is used to incrementally compute the axiomatization of the new node by comparing the needs with the annotated information. Thus, we obtain a set of axioms to be removed from the cached axiomatization and a set of axioms to be inserted to the cached axiomatization to obtain the axiomatization of the new node.

Suppose $\psi = N \xrightarrow{\sigma} M$ is a local theorem link with an attached local proof object $LD(\sigma, (Ax_1, \Phi_1), \ldots, (Ax_k, \Phi_k))$. Each axiom Ax_i of N is related to the proof description Φ_i. Φ_i is either an atom NoProof indicating that this proof obligation has not been proven yet, or an atom ProofExists indicating that some theorem prover has proven the problem but did not return an explicit proof object or at least the set of used axiom, or the set of axioms used to prove $\sigma(Ax_i)$ inside the theory of M. In this case Φ_i breaks down the used axioms according to their origins and the morphism with the help of which they are imported to the target theory.

Changing either the axioms of N, the morphism σ or the subgraph of M may render the proof object pr_ψ invalid. In the following we discuss the repair of the proof object pr_ψ for these three cases:

Change in N: The change of source axioms results in corresponding changes of the proof obligations. Insertion of a new axiom Ax_{k+1} will result in a new entry $(Ax_{k+1}, \text{NoProof})$, where NoProof indicates that $\sigma(Ax_{k+1})$ is still to be proven by some theorem prover. Deletion of some Ax_i will result in the removal of the corresponding pair (Ax_i, Φ_i). Change of a source axiom Ax_i to Ax_i' causes an invalidation of Φ_i. If the system provides explicit proof objects (instead of the set of used axioms) the system supports the theorem prover by additionally providing Ax_i and the old proof for $\sigma(Ax_i)$ when proving $\sigma(Ax_i')$ to allow for a reuse of the old proof.

Change in Morphism σ: Changing the morphism σ attached to the theorem link to σ' may result in a change of some proof obligations depending how the change of the morphism affects the mapping of local axioms of the source theory. If $\sigma(Ax_i) = \sigma'(Ax_i)$ we can reuse the old proof otherwise the proof information is invalidated but stored for a later reuse when tackling the proof obligation $\sigma'(Ax_i)$ by some theorem prover.

Change in M: Since the theory of M depends on its subgraph, every change in this subgraph may affect the theory of M. We distinguish two different approaches depending whether Φ_i is ProofExists or description of used axioms.

1. In the latter case we know about all used axioms (and their origins). The proof is still valid if all used axioms are still part of the theory of M. Instead of computing the changes in the axiomatization of M we check for all triples (τ, K, Ω) whether $\tau(\Omega)$ is still imported to M from K via a morphism τ' with $\tau'(\Omega) = \tau(\Omega)$.
2. If there is no explicit proof object, we assume that all axioms accessible at the time of the proof have been used for the proof. Thus a proof is invalid if some axiom of a node inside the subgraph of M has been changed or

deleted, or some definition link has been changed or deleted and there is no alternative path with the same morphism. This check is restricted to objects which have existed at the time when the proof was done. Hence each object (links, nodes, axioms, etc.) contains timestamps of its creation, its deletion, or its change. For example, changing a morphism does not affect the validity of a proof if all signature entries which are affected by these changes were introduced after the computation of the proof.

Consider our running example and suppose we had already proven some axioms of STACK mapped as theorems to LIST when we inserted the theory REL. As REL only adds new axioms to the theory of LIST, all proofs of the axioms are still valid. This holds although the morphism τ of the local theorem link from STACK to LIST has changed to τ' in order to incorporate the mapping of the new relation R. In case the local proof object provides the list of used axioms we can easily check that $\tau(Ax_i) = \tau'(Ax_i)$ holds for all $1 \leq i \leq n$. Otherwise, the morphisms τ and τ' are compared which results in the fact that the only differences between both morphisms concern the mapping of the relation R which has been introduced after doing the proofs of any Ax_i. Thus, changing τ to τ' will not affect the proofs of any Ax_i done before the insertion of the theory REL.

7 Implementation

The development graph as well as the techniques for their maintenance are implemented in the MAYA system (cf. [9]). Currently the fixed logic underlying the development graph is higher-order logic. The uniform representation of structured theories in the development graph supports evolutionary formal software development with respect to arbitrary specification languages, provided there exists an adequate mapping from the specification language into development graphs. Currently MAYA integrates parsers for the specification languages CASL (cf. [1]) and VSE-SL (cf. [7]). With respect to the verification in-the-small, MAYA supports the use of arbitrary theorem provers for higher-order logic. To this end a generic interface to propagate the changes of theories to the theorem provers has been implemented. Currently, the HOL-CASL instance of Isabelle/HOL (cf. [3]) and the INKA 5.0 theorem prover (cf. [1]) are integrated into MAYA via this interface. The Lisp sources of MAYA can be obtained from the MAYA-webpage [9].

8 Related Work

The KIV system [18] incorporates a development graph similar to the one presented in this paper. However, instead of having basic structuring mechanism like our global and local links, the KIV structure mechanisms are heavy tailored to the structuring constructs of their specification languages. Although this allows for a more adequate representation of global proof obligations, it lacks the ability to easily integrate support for further specification languages. With respect to the verification in-the-large, it also supports the maintenance of established proof obligations when changing the specification, but lacks a mechanism for

redundancy checking and elimination. This is due to the absence of decomposition of proof obligations between graphs into proof obligations between the respective subgraphs. With respect to the verification in-the-small, when the specification is changed, the effects on the axioms usable by the theorem prover cannot be determined in an as granular manner as in the MAYA system. Finally, the tight integration of the KIV development graph with the built-in theorem prover hampers the use of further theorem provers.

The SPECWARE system [10] is a formal software design environment. It follows the paradigm of top-down formal software development using refinement, modularization, and parameterization. The whole design and refinement process is explicitly represented in some kind of development graph and the arising proof obligations are proven using theorem provers. However, like for the KIV system, the basic structuring mechanisms are tailored to the specification language, which hampers the use of other specification languages. Finally, it lacks the support for redundancy checking and elimination, as well as the maintenance of established proof obligations.

The *Little Theories* approach [8] provides a subset of the theory structuring mechanism of development graphs, i.e. global definition links and proven global theorem links. It is more general than development graph, because each theory (node) can have its own logic, whereas for the current implementation of development graphs presented in this paper, the whole graph is with respect to a single logic. The extension of development graphs to deal with different logics has been achieved in theory in [14]. However, little theories lack on the one hand the ability to represent intermediate states of the development, i.e. a state where there still exist yet unproven postulated global theorem links. On the other hand, there are no mechanisms that exploit the graph structure to reduce the amount of proof obligations and to deal with non-monotonic changes of the theories.

9 Conclusion

For the development of industrial-size software systems, the preservation of the structure of specifications is essential not only for the specification of the systems, but also for their verification. Indeed, the structure can be exploited in order to reduce the amount of proof obligations and to support efficiently the revision of specifications, which usually arises in practice.

We presented the implementation of a system for verification in-the-large about structured specifications. It enables to formally find and eliminate redundant proof obligations. Furthermore, it incorporates strategies to transform a proof for some former specification to some new specification, while preserving as many established conjectures as possible.

The theorem proving mechanisms for verification in-the-large are the kernel of the MAYA system [9]. Around that kernel are build on the one hand a uniform interface for parsers of arbitrary specification languages[4], and on the other hand

[4] Provided there is an adequate translation of the logic and the structuring constructs of the specification language into the development graph structure.

a uniform interface to use theorem provers for verification in-the-small. These functionalities enable MAYA to bridge the gap between parsers for specification languages and state of the art automated or interactive theorem provers, and deals with all aspects of evolutionary formal software development based on structured specifications.

Future work will consist of extending the verification in-the-large mechanisms to support development graphs with hiding [14] as well as heterogenous development graphs [13]. Further work will also be concerned with the generation of proof-objects for completed developments from MAYA's internal "in-the-large" proof representation and the annotated "in-the-small" proofs. This proof object shall be used to proof check a completed development, which formally certifies a completed formal software development.

References

1. S. Autexier, D. Hutter, H. Mantel, A. Schairer. System description: INKA 5.0 – a logic voyager. *In H. Ganzinger (Ed.): 16th International Conference on Automated Deduction*, Springer, LNAI 1632, 1999.
2. S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Springer, LNCS 1827, 2000.
3. S. Autexier, T. Mossakowski. Integrating HOL-CASL into the Development Graph Manager MAYA. In A. Armando (Ed.) *Frontiers of Combining Systems (FroCoS'02)*, Santa Margherita Ligure, Italy, Springer LNAI, April, 2002.
4. CoFI Language Design Task Group. *The common algebraic specification language (CASL) – summary*, 1998. Version 1.0 and additional Note S-9 on Semantics, available from http://www.brics.dk/Projects/CoFI.
5. M. Cerioli, J. Meseguer. May I borrow your logic? *Theoretical Computer Science*, 173(2):311-347, 1997.
6. D. Hutter. Management of change in verification systems. In *Proceedings 15th IEEE International Conference on Automated Software Engineering, ASE-2000*, pages 23–34. IEEE Computer Society, 2000.
7. D. Hutter et al.: Verification Support Environment (VSE), *Journal of High Integrity Systems*, Vol. 1, 1996.
8. W. M. Farmer. An infrastructure for intertheory reasoning, In: D. McAllester, ed., *Automated Deduction – CADE-17*, LNCS, 1831:115-131, 2000.
9. MAYA-webpage: http://www.dfki.de/~inka/maya.html.
10. J. McDonald, J. Anton. SPECWARE – Producing Software Correct by Construction. Kestrel Institute Technical Report KES.U.01.3., March 2001.
11. J. Meseguer. General logics, In *Logic Colloquium 87*, pages 275–329, North Holland, 1989.
12. T. Mossakowski: CASL: From Semantics to Tools. In S. Graf (Ed.) *TACAS 2000*, LNCS volume 1785, pages 93-108. Springer, 2000.
13. T. Mossakowski. Heterogeneous development graphs and heterogeneous borrowing. In M. Nielsen (Ed.) Proceedings of *Foundations of Software Science and Computation Structures (FOSSACS02)*, Grenoble, France, Springer LNCS, 2002.

14. T. Mossakowski, S. Autexier, and D. Hutter: Extending Development Graphs With Hiding. In H. Hußmann (Ed.), *Proceedings of Fundamental Approaches to Software Engineering (FASE 2001)*, Italy. LNCS 2029, 269–283. Springer, 2001.
15. S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager MAYA. In *Proceedings 9th International Conference on Algebraic Methodology And Software Technology, AMAST2002*. Springer-Verlag, 2002.
16. D. Hutter and A. Schairer. Proof transformations for evolutionary formal software development. In *Proceedings 9th International Conference on Algebraic Methodology And Software Technology, AMAST2002*. Springer-Verlag, 2002.
17. L. C. Paulson. *Isabelle – A Generic Theorem Prover*. LNCS 828. Springer, 1994.
18. W. Reif: The KIV-approach to Software Verification, In *KORSO: Methods, Languages, and Tools for the Construction of Correct Software – Final Report*, LNCS 1009, 339-368. Springer, 1995.

A Unification Algorithm for Analysis of Protocols with Blinded Signatures

Deepak Kapur[1,⋆], Paliath Narendran[2,⋆⋆], and Lida Wang[2,⋆⋆⋆]

[1] Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA
kapur@cs.unm.edu
[2] Department of Computer Science, SUNY at Albany, Albany, NY 12222, USA
{dran,lidawang}@cs.albany.edu

Abstract. Analysis of authentication cryptographic protocols, particularly finding flaws in them and determining a sequence of actions that an intruder can take to gain access to the information which a given protocol purports not to reveal, has recently received considerable attention. One effective way of detecting flaws is to hypothesize an insecure state and determine whether it is possible to get to that state by a legal sequence of actions permitted by the protocol from some legal initial state which captures the knowledge of the principals and the assumptions made about an intruder's behavior. Relations among encryption and decryption functions as well as properties of number theoretic functions used in encryption and decryption can be specified as rewrite rules. This, for example, is the approach used by the NRL Protocol Analyzer, which uses narrowing to reason about such properties of cryptographic and number-theoretic functions.

Following [15], a related approach is proposed here in which equation solving modulo most of these properties of cryptographic and number-theoretic functions is done by developing new unification algorithms for such theories. A new unification algorithm for an equational theory needed to reason about protocols that use the Diffie-Hellman algorithm is developed. In this theory, multiplication forms an abelian group; exponentiation function distributes over multiplication, and exponents can commute. This theory is useful for analyzing protocols which use blinded signatures. It is proved that the unification problem over this equational theory can be reduced to the unification problem modulo the theory of abelian groups with commuting homomorphisms with an additional constraint. Baader's unification algorithm for the theory of abelian groups with commuting homomorphisms, which reduces the unification problem to solving equations over the polynomial ring over the integers with the

⋆ Research supported in part by the NSF grant nos. CCR-0098114 and CDA-9503064, the ONR grant no. N00014-01-1-0429, and a grant from the Computer Science Research Institute at Sandia National Labs.

⋆⋆ Research supported in part by NSF grant no. CCR-0098095 and ONR grant no. N00014-01-1-0430.

⋆⋆⋆ Research supported in part by NSF grant no. CCR-0098095.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 433–451, 2005.

commuting homomorphisms serving as indeterminates, is generalized to give a unification algorithm over the theory of abelian groups with commuting homomorphism with a *linear constraint*.
It is also shown that the unification problem over a (simple) extension of the equational theory considered here (which is also an extension of the equational theory considered in [15]) is undecidable.

1 Introduction

Search techniques for exploring the vast state space possibly generated by a given authentication cryptographic protocol have turned out to be an effective way to determine possible flaws in protocols. A typical state is the knowledge possessed by each of the principals interested in communicating among each other in a secure fashion, time clock often needed for generating nonces and/or timestamps, and most importantly, an intruder who can read, alter and delete traffic, send messages of its own, pretend to be any of the principals and who may have help from the principals, etc. Actions taken by principals and an intruder(s) lead to state changes. At the same time, certain relations between various cryptographic functions as well as properties of number theoretic functions used for public encryption and decryption must be honored by the states. For instance in a symmetric key system, it is common to introduce a rule saying that, if a principal knows a message M encrypted with a key K, and also knows the key K, then it can learn message M. In a public key based cryptosystem, encryption and decryption can be expressed in terms of two functions symbols e and d that obey the following identities: $e(Kpu, d(Kpr, M)) \rightarrow M$ and $d(Kpr, e(Kpu, M)) \rightarrow M$ where Kpu, Kpr are respectively the public and private keys for a principal.

A technical report by Clark and Jacob [7] is a comprehensive survey of authentication protocols. That report reviews numerous protocols proposed in the literature; its annotated bibliography also discusses various attacks on some of these protocols and how they have been fixed. The report briefly mentions various approaches for establishing correctness of authentication protocols. An interested reader should consult [7] for details. Below, we discuss a state-based approach for analyzing possible attacks on an authentication protocol; see [7, 14, 13] for more details.

A typical scenario for analyzing a protocol is to hypothesize an insecure state in which the protocol is compromised possibly by an intruder knowing some information that the protocol professes not to reveal, and to work *backwards* to determine whether this state is reachable, by a sequence of actions of the principals and the intruder, from a given legal state. This, for example, is the approach used by the NRL Protocol Analyzer (NPA) [14], a software tool for cryptographic protocol analysis implemented in Prolog. NPA exploits the backtracking facility in Prolog in combination with equational unification for exploring the state space. It makes use of narrowing, a general purpose technique for equation solving, to identify states which are equivalent because of properties of encryption, decryption functions as well as properties of number theoretic functions used in authentication, encryption and decryption. For instance, most public key based

cryptosystems use multiplication, exponentiation, and modulus operations on numbers. Relations are specified by terminating rewrite rules. In certain cases, NPA is even able to rule out the reachability of such insecure states.

(Narrowing is a general purpose technique for equation solving with respect to a given rewrite system; it thus requires that every property be oriented into a terminating rewrite rule. See [9], for instance.)

Although narrowing works well for certain properties relating encryption and decryption, unfortunately there are a number of other properties of cryptographic operations which cannot be handled. Certain relations, such as associativity and commutativity properties of arithmetic operations $+$, $*$, etc, cannot be oriented into terminating rewrite rules. Furthermore, narrowing is a general purpose method, whereas special purpose unification algorithms for certain theories capturing relations among cryptographic functions could perhaps be more efficient.

An approach for integrating unification algorithms for equational theories axiomatizing properties of cryptosystems including encryption, decryption and primitive number theoretic functions implementing them, is outlined in a recent paper by Meadows and Narendran [15]. The basic idea is to use a unification algorithm for an equational theory in place of narrowing using rewrite rules in the equational theory. A unification algorithm for an equational theory found useful in analyzing group Diffie-Hellman protocols was proposed in the paper and the complexity of the unifiability check for this equational theory was shown to be NP-complete.

This paper builds on [15]. The equational theory under consideration is assumed to be weaker than the theory considered in [15] with respect to the properties of the exponentiation function on numbers. in particular, the axiom $x^{y^z}, = x^{y \cdot z}$ relating exponentiation with multiplication is dropped. However, the exponents are assumed to commute, i.e., $x^{y^z} = x^{z^y}$ holds[1]. This theory is useful in cryptographic techniques such as Chaum's blinded signature [6], popular in anonymous electronic cash schemes, which also make direct use of properties such as distributivity of modular exponentiation over modular multiplication (i.e., $(x \cdot y)^z = (x^z) \cdot (y^z)$).

It is proved in this paper that the above distributivity axiom cannot be added to the theory considered in [15] without making the unification problem on the extended theory undecidable. The undecidability proof of the unification problem over the equational theory of *Cartesian Closed Categories* (*CCC*) considered in [16] can be adapted to be applicable for this theory.

A decision procedure for the unification problem for the equational theory of abelian group for multiplication along with the distributivity axiom and the exponent commutativity axiom (and the properties of the unit 1) is also given. In this sense, the theory considered in this paper and the theory considered in [15] are two proper subsets with decidable unification problems of an equational theory with an undecidable unification problem.

[1] It is easy to see that this property follows from the axiom $x^{y^z} = x^{y \cdot z}$ because of commutativity of $\cdot$.

The remainder of this paper is organized as follows. Section 2 is based on [15], providing a brief review of the Diffie-Hellman and group Diffie-Hellman algorithms, and discussing axioms relating number-theoretic properties of multiplication and exponentiation used. It is included for the sake of completeness; for details, an interested reader may consult [15].

Section 3 shows the undecidability of the equational theory considered in [15] along with the distributivity axiom of exponentiation over multiplication. The proof is essentially based on the undecidability proof given in [16] where Hilbert's tenth problem over natural numbers is shown to be an instance of the unification problem over the equational theory of cartesian closed categories. The main difference is that in the case below, it is possible to simulate negative numbers as well. Consequently, Hilbert's tenth problem over integers is reduced to this unification problem.

Section 4 develops the necessary background to relate the unification problem over the equational theory of blinded signatures to the unification problem over the equational theory of abelian groups with n commuting homomorphisms, called $AGnHC$ by Baader [2], with an additional condition, called a *linear constraint*.

Section 5 shows how Baader's algorithm for the unification problem over $AGnHC$ can be generalized so as to satisfy a linear constraint (which is similar to a linear constant restriction discussed in [1]). A new way of defining admissible term orderings is introduced, which is used to compute a Gröbner basis of a polynomial ideal.

Section 6 concludes with some remarks on complexity of the algorithm and outlines some areas for further research.

2 Protocols Based on Diffie-Hellman Algorithm

This section is borrowed from [15] where the motivation for this approach is outlined.

The Diffie-Hellman algorithm in its most basic form allows two principals to securely exchange a secret key without having any shared secret beforehand. Consider a prime number P and a generator x of the multiplicative group of $\mathbf{Z}_P$. Principal A generates a secret value N_A, and B generates a secret value N_B. The protocol then runs as follows:

1. $A \rightarrow B \ : \ x^{N_A} \bmod P$
   B then computes $(x^{N_A})^{N_B}$.
2. $B \rightarrow A \ : \ x^{N_B} \bmod P$
   A then computes $(x^{N_B})^{N_A}$, which is the same as $(x^{N_B})^{N_A}$, implying that the shared key is the same among the principals A and B.

In order to secure the above protocol against active eavesdroppers, it is necessary to include some form of authentication, usually provided by public-key signatures. Thus an equational theory that could be used to reason about Diffie-Hellman would need to take into account the relationship between exponentiation and $\cdot$, the commutativity of $\cdot$, and possibly identities obeyed by the signature algorithms used.

Interaction between exponentiation and $\cdot$ is captured by the distributivity axiom: $(x \cdot y)^z = (x^z) \cdot (y^z)$.

2.1 Some Equational Theories for Diffie-Hellman and Group Diffie-Hellman

Equational theories under consideration consist of the axioms of abelian groups (A, C, U and Inv below), denoted by AG, along with (some) axioms for exponentiation. In contrast to the equational theory considered in [15], the axiom, $(x^y)^z = x^{y \cdot z}$, used in [15] is excluded; instead, axioms (Exp2, Exp3, Exp4) are used. The relationship between exp and $\cdot$ is captured by axiom Exp3.

Consider the following axioms for $\cdot$ and exponentiation (denoted by x^y):

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \qquad \text{(A)}$$
$$x \cdot y = y \cdot x \qquad \text{(C)}$$
$$x \cdot 1 = x \qquad \text{(U)}$$
$$x \cdot x^{-1} = 1 \qquad \text{(Inv)}$$
$$x^1 = x \qquad \text{(Exp1)}$$
$$1^x = 1 \qquad \text{(Exp2)}$$
$$(x \cdot y)^z = (x^z) \cdot (y^z) \qquad \text{(Exp3)}$$
$$x^{y^z} = x^{z^y} \qquad \text{(Exp4)}$$
$$(x^y)^z = x^{y \cdot z} \qquad \text{(Exp5)}$$

The first four axioms characterize abelian groups; this theory is denoted by AG. The remaining axioms are about exponentiation as well as interactions between exponentiation and $\cdot$ (particularly, axioms (Exp4) and (Exp5)).

As stated earlier, [15] gave a unification algorithm for the equational theory consisting of AG and axioms (Exp1, Exp2, Exp5); it was proved that the unifiability check is NP-complete. It is easy to see that (Exp4) follows from AG + (Exp5).

It is shown in the next section that the unifiability check for AG and axioms (Exp1, Exp2, Exp3) and (Exp5) is undecidable. The rest of the paper focuses on the decidability of the unifiability check for AG plus axioms {Exp1, Exp2, Exp3, Exp4}.

3 Undecidability of Unifiability Check for the Theory of AG and Exponentiation

Consider the equational theory $\mathcal{E}$ consisting of AG and axioms (Exp1, Exp2, Exp3 and Exp5). This theory has the following convergent rewrite system modulo associativity and commutativity (also called AC-convergent).

$$
\begin{aligned}
(x^{-1})^{-1} &\to x \\
1^{-1} &\to 1 \\
x \cdot 1 &\to x \\
x \cdot x^{-1} &\to 1 \\
x \cdot (x^{-1} \cdot y) &\to y \\
(x \cdot y)^{-1} &\to x^{-1} \cdot y^{-1} \\
1^z &\to 1 \\
z^1 &\to z \\
(x^{-1})^y &\to (x^y)^{-1} \\
(x \cdot y)^z &\to (x^z) \cdot (y^z) \\
(x^y)^z &\to x^{(y \cdot z)}
\end{aligned}
$$

It is easy to see that the normal form of a term using the above rewrite rules is either 1 or of the form $t_1 \cdot t_2 \cdot \ldots \cdot t_n$, where each t_i is x, x^{-1}, x^{e_i}, or $(x^{e_i})^{-1}$ for some variable or constant x, where e_i is a term in normal form wrt the above AC-convergent rewrite system that is different from 1. In what follows, a term of the form b^i where i is a positive integer, is an abbreviation of $\underbrace{b \times b \times \ldots \times b}_{i}$. If i is a negative integer then b^i is an abbreviation of $\underbrace{b^{-1} \times b^{-1} \times \ldots \times b^{-1}}_{(-i)}$. ($b^0 = 1$.)

Lemma 1. *Let $t_1, \ldots, t_n$ be terms in normal form modulo the above AC-convergent system, such that none of the t_i's has $\cdot$ as the outermost symbol. If s is the normal form of $t = t_1 \cdot \ldots \cdot t_n$, then s is either 1 or s can be written as $s_1 \cdot \ldots \cdot s_k$ where $\{s_1, \ldots, s_k\} \subseteq \{t_1, \ldots, t_n\}$ (in the multiset sense).*

Proof: If t is already in normal form, then the lemma holds. If for each $i\,(1 \leq i \leq n)$ $t_i = 1$, then s is 1. Otherwise since $t_i\,(1 \leq i \leq n)$ is in normal form and none of the t_is has $\cdot$ as the outermost symbol, the only possible reduction rules that can apply to t are:

$$
\begin{aligned}
x \cdot 1 &\to x \\
x \cdot x^{-1} &\to 1 \\
x \cdot (x^{-1} \cdot y) &\to y
\end{aligned}
$$

Each time any of these these rules is applied, some t_i will be gotten rid of. By induction on the number of reduction steps, we can complete the proof. □

Lemma 2. *For all i, the $\mathcal{E}$-unification problem*

$$x \cdot a^y =^? x^b \cdot a^{b^i}$$

is solvable iff $y \leftarrow b^m$ for some integer m.

Proof: The "if" part is straightforward. It is clear that for every $n \geq i$, $\{y \leftarrow b^n, x \leftarrow a^{b^{n-1}} * a^{b^{n-2}} * \cdots * a^{b^{i+1}} * a^{b^i}\}$ is indeed a solution. We prove the "only

if" part by contradiction. Suppose the claim is not true, i.e., there is a solution with a y that is not of the form b^j where j is an integer.

Let $S = \{\ x \mid x$ is in normal form and there are y and i such that
$a^y \cdot x =_{\mathcal{E}} x^b \cdot a^{b^i}$, i is an integer, y is not of the form b^j
where j is an integer $\}$

Let x be the smallest term in the above set. Since x is in normal form, it must be that $x^b.a^{b^i}.i(a^y)$ reduces to x. Now two cases have to be considered:

(i) a^{b^i} is a part (factor) of x, i.e., $x =_{\mathcal{E}} a^{b^i} \cdot z$ for some z. Then

$$a^y \cdot a^{b^i} \cdot z =_{\mathcal{E}} z^b \cdot a^{b^{i+1}} \cdot a^{b^i}$$

Canceling on both sides, we find that $z \in S$ and z is smaller than x.

(ii) the normal form of x^b contains $i(a^{b^i})$. Since x is in normal form, x must contain $i(a^{b^{i-1}})$, i.e., $x =_{\mathcal{E}} i(a^{b^{i-1}}).z$. Then

$$a^y \cdot i(a^{b^{i-1}}) \cdot z =_{\mathcal{E}} z^b$$

Again, z is smaller than x, which is a contradiction. □

Lemma 3. *Let b and c be free constants and j be an integer. Then the $\mathcal{E}$-unification problem*

$$x^c * a^{b^j} =^? x^b * a^u$$
$$z * a^u =^? z^c * a$$

is unifiable iff $u \leftarrow c^j$. (In other words, $x^c * a^{b^j} =^? x^b * a^{c^k}$ is unifiable if and only if $j = k$.)

Proof: By Lemma 2, the second equation is unifiable iff $u \leftarrow c^k$ where k is an integer. Suppose the equation $x^c * a^{b^j} =^? x^b * a^{c^k}$ is solvable. Replace b with c everywhere in the equation. Since b and c are free constants, it must be that $y^c * a^{c^j} =_{\mathcal{E}} y^c * a^{c^k}$ where y is x with b replaced everywhere with c. Now the cancellation properties can be applied to get the result. □

Lemma 4. *Let b and c be free constants and j and k be integers. Then $\mathcal{E}$-unification problem*

$$x^{c^k} * a^{b^j} =^? x^b * a^u$$
$$z * a^u =^? z^c * a$$

is unifiable iff $u \leftarrow c^{j*k}$.

Proof: By Lemma 2, the second equation is unifiable iff $u \leftarrow c^n$ where n is an integer. Suppose the equation $x^{c^k} * a^{b^j} =^? x^b * a^{c^n}$ is solvable. Replace b with c^k everywhere in the equation. Since b and c are free constants, it must be that $y^{c^k} * a^{c^{j*k}} =_{\mathcal{E}} y^{c^k} * a^{c^n}$ where y is x with b replaced everywhere with c^k. Now the cancellation properties can be applied to get the result. □

Lemma 4 shows that multiplication of two integers can be simulated. Consider, for instance, the equation $z = x * y$. If $x = b^i$ and $y = b^j$, then we can force z to be b^{ij} in the following way:

(i) Copy x to x' changing b's to c's; i.e., $x' = c^i$. This can be done using equations as given in the statement of Lemma 4.
(ii) Multiply x' and y to get $z' = c^{ij}$.
(iii) Copy z' to z changing c's to b's.

Thus the equations we get are

$$\begin{aligned} w_1 * a^{x'} &=^? {w_1}^c * a \\ {w_2}^c * a^x &=^? {w_2}^b * a^{x'} \\ {w_3}^{x'} * a^y &=^? {w_3}^b * a^{z'} \\ w_4 * a^{z'} &=^? {w_4}^c * a \\ {w_5}^c * a^z &=^? {w_5}^b * a^{z'} \end{aligned}$$

Simulating addition is easy, since if $x = b^i$ and $y = b^j$, then $(a^x)^y = a^{b^{i+j}}$.

Now we are ready to prove the undecidability of the equational theory $\mathcal{E}$ by a reduction from Hilbert's tenth problem.

Theorem 1. *The unifiability check for the equational theory $\mathcal{E}$ is undecidable.*

Proof: Given a system of diophantine equations, construct a unification problem modulo $\mathcal{E}$ as outlined above. □

4 Unification over Equational Theory of Blinded Signatures

In this section, we consider a proper subset of the above equational theory. In particular, the axiom (Exp5) is replaced by a weaker axiom (Exp4). Let $\mathcal{E}_0$ consist of *AG* and axioms (Exp1, Exp2, Exp3, Exp4) denoted by *Exp*. The theory $\mathcal{E}_0$ − Exp4, denoted by $\mathcal{E}_0$', has the following AC-convergent rewrite system.

$$\begin{aligned} (x^{-1})^{-1} &\to x \\ 1^{-1} &\to 1 \\ x \cdot 1 &\to x \\ x \cdot x^{-1} &\to 1 \\ (x \cdot y)^{-1} &\to x^{-1} \cdot y^{-1} \\ 1^z &\to 1 \\ z^1 &\to z \\ (x^{-1})^y &\to (x^y)^{-1} \\ (x \cdot y)^z &\to (x^z) \cdot (y^z) \end{aligned}$$

Exp4 cannot be oriented into a terminating rewrite rule.

In the next sections, we show that the equational unification problem for $\mathcal{E}_0$ is equivalent to the unification problem modulo the theory of abelian groups with n *commuting* homomorphisms, denoted by *AGnHC*, but with an additional

constraint. The theory $AGnHC$ is a *monoidal* theory; further, it is shown in [17, 3] that $AGnHC$ is *unitary* with respect to unification without constants and it is also unitary with respect to unification with constants[2]. In Section 5 of [2], Baader showed that the unification problem of $AGnHC$ reduces to solving linear equations over the polynomial ring $\mathsf{Z}[h_1, \ldots, h_n]$, where $h_1, \ldots, h_n$ are the commuting homomorphisms of $AGnHC$, treated as indeterminates in the polynomial ring. We generalize Baader's algorithm by adding an additional key step to ensure that a given *linear constraint* is satisfied by the unifier, so as to apply it to the equational unification problem for $\mathcal{E}_0$. This is discussed in this section and the next section.

4.1 Unification over $\mathcal{E}_0$ as a Combination of Theories

Consider a set S_0 of equations whose unifiability needs to be checked wrt $\mathcal{E}_0$. Assuming S_0 is unifiable, given any unifier θ of S_0, θ may substitute 1 for certain variables in S_0; also many variables in S_0 may get identical substitutions. Given that there are only finitely many variables in S_0, there are only finitely many such partial unifiers for S_0 in which some of the variables in S_0 get either 1 or another variable as a substitution. After applying such a partial unifier on S_0, simplifying the result by the rewrite rules of the associated AC-convergent system, and deleting trivial equations (i.e., equations which are in the equational theory of $\mathcal{E}_0$), we get a set S_1 of equations. If S_1 is empty, then the above partial unifier is a unifier of S_0. If S_1 is not empty, then the following steps are applied. We will assume that the equations in S_1 have been normalized using the AC-convergent rewrite system for $\mathcal{E}_0'$ $(= \mathcal{E}_0 - \text{Exp4})$ discussed above. Thus, a unifier of S_1 cannot substitute for any variable x, a normalized term t properly containing x (occur-check).

Any unification problem S_1 over $\mathcal{E}_0$ can be simplified using variable abstraction (by introducing new symbols) to a *simple* $\mathcal{E}_0$-unification problem, say S_2; this is defined precisely below.

Definition 1. *An $\mathcal{E}_0$-unification problem S over Σ is called an* AG*-unification problem if each equation in S is of the form $x =^? t$, where x is a variable and t is a term over the signature of AG such that $t \neq_{AG} 1$.*

Definition 2. *An $\mathcal{E}_0$-unification problem S on Σ is called an* exponent $\mathcal{E}_0$*-unification problem if every equation in S is of the form $x =^? y^z$ where x and y are variables and z is a variable or a free constant. Also if z is a variable, z is called an* exponent variable, *otherwise it is called an* exponent constant.

Definition 3. *An $\mathcal{E}_0$-unification problem S on Σ is called a* simple $\mathcal{E}_0$*-unification problem if $S = S_1 \bigcup S_2$ where S_1 is an AG-unification problem and S_2 is an exponent $\mathcal{E}_0$-unification problem.*

Let $Var(S)$ denote the set of all variables in S.

[2] A theory is unitary if a minimal complete set of unifiers always exists and its cardinality is at most one.

It is easy to see that using abstraction, any $\mathcal{E}_0$-unification problem can be transformed into a simple $\mathcal{E}_0$-unification problem. For example, consider $S_1 = \{w =^? (x^{(y^{u\cdot v})^{-1}\cdot z^{u'\cdot v'}})^{-1}\}$. Using abstractions, the above equation in S_1 is transformed to S_2:

$$\{1.\ w =^? z_1^{-1}, \quad 2.\ z_1 =^? x^{z_2}, \quad 3.\ z_2 =^? z_3^{-1}\cdot z_4, \quad 4.\ z_3 =^? y^{z_5},$$
$$5.\ z_5 =^? u\cdot v, \quad 6.\ z_4 =^? z^{z_6}, \quad 7.\ z_6 =^? u'\cdot v'\},$$

where $z_1, z_2, z_3, z_4, z_5, z_6$ are new variables introduced to abstract alien subterms in S_1.

4.2 Relating $AG + EXP$ to $AGnHC$

Given a simple $\mathcal{E}_0$-unification problem S_2 on $\Sigma = \{\cdot, ^{-1}, 1, x^y\}$, for each equation of the form $x = y^\beta$ in S_2, we transform it into $x = h_\beta(y)$, where h_β is a homomorphism corresponding to the symbol β. Let $\mathcal{H}(S_2)$ denote the set of all homomorphisms introduced in this way. Let $\Sigma' = \{\cdot, ^{-1}, 1\} \bigcup \mathcal{H}(S_2)$, $\mathcal{E}' = AG \bigcup \{h_1(h_2(u)) = h_2(h_1(u)),\ h(u_1\cdot u_2) = h(u_1)\cdot h(u_2)\}$ for all $h, h_1, h_2 \in \mathcal{H}(S_2)$. We call the transformed $\mathcal{E}'$-unification problem on Σ' as an *h-image* of S_2.

For the above example, its *h-image* T_2 is:

$$\{1.\ w =^? z_1^{-1}, \quad 2.\ z_1 =^? h_{z_2}(x), \quad 3.\ z_2 =^? z_3^{-1}\cdot z_4, \quad 4.\ z_3 =^? h_{z_5}(y),$$
$$5.\ z_5 =^? u\cdot v, \quad 6.\ z_4 =^? h_{z_6}(z), \quad 7.\ z_6 =^? u'\cdot v'\}.$$

The requirement that a normalized unifier for S_2 wrt $\mathcal{E}_0$ satisfy the occur-check for every variable x translates to a related requirement in $\mathcal{E}'$. A normalized unifier of the h-image T_2 of S_2 wrt $AGnHC$ should satisfy (i) the occur-check for every variable x, and in addition, (ii) the substitution for x must not properly include h_x, the homomorphism introduced for x when x appears as an exponent.

In order to show the equivalence of the unifiability check over $\mathcal{E}_0$ with the unifiability check over $AGnHC$, it is necessary to place restrictions on unifiers considered for $\mathcal{E}_0$ given that we have considered a priori equivalent substitutions for variables as well as 1 as the substitution for variables. This is done by solving the unifiability problem of T_2 wrt $AGnHC$ subject to *linear* constraints (including) $x \succ h_x$ for every homomorphism $h_x \in \mathcal{H}(S_2)$, i.e., a unifier θ of T_2 should satisfy the condition that for every $x \in Var(T_2)$, $\theta(x)$ *does not contain* any occurrence of h_x.

Definition 4. *Given a simple $\mathcal{E}_0$-unification problem S and its h-image T modulo AGnHC, a* linear constraint *is a total order $\succ$ over $Var(T) \cup \mathcal{H}(S)$ such that $x \succ_C h_x$ for all exponent variables x in S.*

Definition 5. *A substitution β whose domain is $Var(T)$ is said to satisfy a linear constraint C if and only if the following holds: for every $x \in Var(T)$,* $\beta(x)$ does not contain *any of the function symbols below x in C. In other words, if $x \succ_C h_y$, then $\beta(x)$ does not contain any occurrence of h_y.*

Definition 6. *A unifier θ for a unification problem S is said to be a* discriminating *unifier if and only if the following hold for all variables in $Var(S)$:*

1. $\theta(u) \neq_{\mathcal{E}} 1$ *for all* u.
2. $\theta(v) =_{\mathcal{E}} \theta(w)$ *iff* $v = w$.

The following two theorems relate the unification problem S_2 over $\mathcal{E}_0$ to its h-image T_2 over $AGnHC$.

Theorem 2. *If a simple $\mathcal{E}_0$-unification problem S_2 has a discriminating unifier, then its h-image T_2 which is a $\mathcal{E}'$-unification problem on Σ' is unifiable. Furthermore, there is a linear constraint C that the unifier satisfies.*

Proof: Let θ be a discriminating unifier of a simple $\mathcal{E}_0$-unification problem S_2. Consider all the exponent equations in S_2:

$$\begin{aligned}\{\ & x_{u_1} =^? {x_{v_1}}^{x_{w_1}} \\ & \vdots \quad\ \vdots \\ & x_{u_i} =^? {x_{v_i}}^{x_{w_i}} \\ & \vdots \quad\ \vdots \\ & x_{u_k} =^? {x_{v_k}}^{x_{w_k}}\}\end{aligned}$$

The unifier θ includes $\{x_{u_i} \leftarrow t_{u_i}, x_{v_i} \leftarrow t_{v_i}, x_{w_i} \leftarrow t_{w_i} (1 \leq i \leq k)\}$. Thus, $t_{u_i} =_{\mathcal{E}_0} {t_{v_i}}^{t_{w_i}} (1 \leq i \leq k)$.

For each t_{w_i}, we introduce a homomorphism $h_{t_{w_i}}$. Let $\mathcal{H}''(S)$ denote the set of homomorphisms introduced for all t_{w_i}. Let $\Sigma'' = \{\cdot, ^{-1}\} \bigcup \mathcal{H}''(S)$. Let $\mathcal{E}'' = AG \bigcup \{h_{t_{w_i}}(h_{t_{w_j}}(u)) = h_{t_{w_j}}(h_{t_{w_i}}(u)),\ h_{t_{w_i}}(u_1 \cdot u_2) = h_{t_{w_i}}(u_1) \cdot h_{t_{w_i}}(u_2)\}$ for all i $(1 \leq i \leq k)$. We also define a recursive function $\boldsymbol{rep}$ as follows:

$$\begin{aligned}&\boldsymbol{rep}(a) = a \text{ where } a \text{ is a constant in } \Sigma. \\ &\boldsymbol{rep}(A \cdot B) = \boldsymbol{rep}(A) \cdot \boldsymbol{rep}(B) \text{ where } A, B \text{ are terms on } \Sigma. \\ &\boldsymbol{rep}(A^x) = h_x(\boldsymbol{rep}(A)) \text{ where } x, A \text{ are terms on } \Sigma.\end{aligned}$$

It is easy to see that the function $\boldsymbol{rep}$ removes all occurrences of the exponent operator from terms over Σ. Since θ is a discriminating unifier for S_2, for each $t_{w_i}, t_{w_j} (i \neq j)$, we have $t_{w_i} \neq_{\mathcal{E}_0} t_{w_j}$. Also it is easy to show that $s =_{\mathcal{E}_0} t$ if and only if $\boldsymbol{rep}(s) =_{\mathcal{E}'} \boldsymbol{rep}(t)$ for any terms s, t over Σ. Therefore we should have:

$$\boldsymbol{rep}(t_{u_i}) =_{\mathcal{E}''} \boldsymbol{rep}({t_{v_i}}^{t_{w_i}}) =_{\mathcal{E}''} h_{t_{w_i}}(\boldsymbol{rep}(t_{v_i}))\ (1 \leq i \leq k) \text{ ——(1).}$$

Now for each $h_{t_{w_i}} (1 \leq i \leq k)$, we introduce the homomorphism $h_{x_{w_i}}$ which is the same homomorphism we introduced for x_{w_i} in the *h-image* T_2 of S_2. We also define function $\boldsymbol{rep'}$ as:

$$\begin{aligned}&\boldsymbol{rep'}(a) = a \text{ where } a \text{ is a constant on } \Sigma'. \\ &\boldsymbol{rep'}(A \cdot B) = \boldsymbol{rep'}(A) \cdot \boldsymbol{rep'}(B) \text{ where } A, B \text{ are terms on } \Sigma'. \\ &\boldsymbol{rep'}(h_{t_{w_i}}(A)) = h_{x_{w_i}}(\boldsymbol{rep'}(A)) \text{ where } A \text{ is a term on } \Sigma.\end{aligned}$$

Obviously $\boldsymbol{rep'}$ maps all $h_{t_{w_i}} (1 \leq i \leq k)$ to the corresponding $h_{x_{w_i}}$. So from (1) and the definition of $\mathcal{E}'$,

$$rep'(rep(t_{u_i})) =_{\mathcal{E}'} h_{x_{w_i}}(rep'(rep(t_{v_i}))).$$

That means T_2 is solvable and a unifier β for T_2 can be constructed as follows: $\beta(x) \leftarrow \boldsymbol{rep'}(\boldsymbol{rep}(\theta(x)))$ for every $x \in Var(S)$.

A linear constraint that β satisfies is constructed by comparing $\theta(x_i)$ for variables in $Var(T_2)$ using a simplification AC term ordering that is total on ground terms (e.g. [12])[3]. □

Theorem 3. *Given a simple $\mathcal{E}_0$-unification problem S_2 on Σ and its h-image T_2 which is a $\mathcal{E}'$-unification problem on Σ', if T_2 has a solution which satisfies a linear constraint, then S_2 is solvable.*

Proof: Consider in S_2, all exponent equations

$$\begin{aligned}\{\ & x_{u_1} =^? x_{v_1}{}^{x_{w_1}} \\ & \vdots \quad\ \vdots \\ & x_{u_i} =^? x_{v_i}{}^{x_{w_i}} \\ & \vdots \quad\ \vdots \\ & x_{u_k} =^? x_{v_k}{}^{x_{w_k}}\ \}.\end{aligned}$$

The h-image T_2 for S_2 includes:

$$\begin{aligned}\{\ & x_{u_1} =^? h_{x_{w_1}}(x_{v_1}) \\ & \vdots \quad\ \vdots \\ & x_{u_i} =^? h_{x_{w_i}}(x_{v_i}) \\ & \vdots \quad\ \vdots \\ & x_{u_k} =^? h_{x_{w_k}}(x_{v_k})\}.\end{aligned}$$

Let β be a ground unifier of T_2 which satisfies a linear constraint C. From C, we can get a subconstraint C' on variables in $Var(T_2)$. Assume without loss of generality that $C' = x_n \succ \cdots \succ x_i \succ \cdots \succ x_1$. Now we will use induction on C' to form θ.

Let us first consider the first variable x_n in C'. Since x_n is the first variable, and $\beta(x_n)$ should not contain any item below x_n in C, it must be that $\beta(x_n)$ is composed of constants, and we define $\theta(x_n) = \beta(x_n)$.

Assume that we have already constructed all $\theta(x_{j'})$ $(j \leq j' \leq n)$. For variable x_{j-1}, the following cases arise:

[3] Given a total AC-simplification ordering on ground terms $>$, add a new constant, say $\perp$, smaller than every other symbol. Now order the terms in the set

$$\{\theta(x_1), ..., \theta(x_n), \perp^{\theta(x_1)}, ..., \perp^{\theta(x_n)}\}$$

using $>$. All these terms will be distinct because we are considering a discriminating unifier. Note also that any term that contains $\theta(x_i)$ as an exponent is $> \perp^{\theta(x_i)}$. Replacing the $\theta(x_i)$'s by the (corresponding) x_i and replacing the $\perp^{\theta(x_i)}$'s by the corresponding h_{x_i}, we get a linear chain. Reversing the order gives C.

1. $\beta(x_{j-1})$ is composed of constants. In this case, we define $\theta(x_{j-1}) = \beta(x_{j-1})$.
2. $\beta(x_{j-1})$ is composed of constants and some $h_{x_{w_i}}$ $(1 \leq i \leq k)$ where each $h_{x_{w_i}} \succ x_{j-1}$. Since $x_{w_i} \succ h_{x_{w_i}}$, we have $x_{w_i} \succ h_{x_{w_i}} \succ x_{j-1}$. By the induction hypothesis, we already know all these $\theta(x_{w_i})$. Therefore, we can define $\theta(x_{j-1}) = \boldsymbol{repp}(\beta(x_{j-1}))$ where the function $\boldsymbol{repp}$ is defined as:

$$\begin{aligned}
&\boldsymbol{repp}(a) = a \text{ where } a \text{ is a constant on } \Sigma'.\\
&\boldsymbol{repp}(A \cdot B) = \boldsymbol{repp}(A) \cdot \boldsymbol{repp}(B) \text{ where } A, B \text{ are terms on } \Sigma'.\\
&\boldsymbol{repp}(h_{x_{w_i}}(A)) = \boldsymbol{repp}(A)^{\theta(x_{w_i})} \text{ where } A \text{ is a term on } \Sigma'.
\end{aligned}$$

It can be shown that θ is a solution for S_2. Consider each exponent equation in T_2: $x_{u_i} =^? h_{x_{w_i}}(x_{v_i})$. Since $\beta(x_{u_i}) =_{\mathcal{E}'} h_{x_{w_i}}(\beta(x_{v_i}))$, we have $\theta(x_{u_i}) =_{\mathcal{E}_0} (\theta(x_{v_i}))^{\theta(x_{w_i})}$ by our definition of θ. □

In the next section, we show how Baader's algorithm for unifiability check for $AGnHC$ can be generalized to work with a linear constraint. This generalization is then used to solve the unifiability check over $\mathcal{E}$', and hence $\mathcal{E}_0$.

5 Unification over $AGnHC$ with a Linear Constraint

In [2], Baader showed that the unifiers of a unification problem wrt $AGnHC$, where $h_1, \ldots, h_k$ are the commuting homomorphisms, correspond to the solutions of (nonhomogeneous) linear equations over the polynomial ring $Z[h_1, ..., h_k]$ with $h_1, \ldots, h_k$ as indeterminates in the polynomial ring. Let $NHE =$

$$\begin{aligned}
\{p_{11}X_1 + \cdots + p_{1n}X_n &= p_1,\\
\vdots &\quad\ \vdots\\
p_{m1}X_1 + \cdots + p_{mn}X_n &= p_m\}
\end{aligned}$$

be a set of linear equations where $p_{11}, \cdots, p_{1n}, \cdots, p_{m1}, \cdots, p_{mn}, p_1, \cdots, p_m$ are in $Z[h_1, ..., h_k]$.

Baader [2] gave an algorithm for solving such nonhomogeneous linear equations by first computing a *syzygy basis* for homogeneous linear equations over $Z[h_1, ..., h_k]$ using an algorithm for computing a *weak Gröbner basis* of a polynomial ideal and then computing a particular solution[4].

Let SB denote a syzygy basis

$$\{(q_{11}, \cdots, q_{1n}), \ldots, (q_{w1}, \cdots, q_{wn})\}$$

for the set HE of the homogeneous equations

$$\begin{aligned}
\{p_{11}X_1 + \cdots + p_{1n}X_n &= 0,\\
\vdots &\quad\ \vdots\\
p_{m1}X_1 + \cdots + p_{mn}X_n &= 0\}.
\end{aligned}$$

[4] See Baader [2] for a definition of a weak Gröbner basis as well as syzygy basis.

Let $\pi = (q_1, \ldots, q_n)$ be a particular solution for the above set of nonhomogeneous equations obtained, for instance, using Baader's algorithm. From the particular solution, a most general unifier for the unification problem wrt $AGnHC$ is computed (as stated above, $AGnHC$ is unitary for unification without as well as with constants [17, 3]). The algorithm is nontrivial; we will not discuss the details here because of space limitations, but suggest the reader to refer to [2] for details.

Proposition 1. *$\pi' = (q'_1, \ldots, q'_n)$ is equivalent to π with respect to SB and hence, is also a particular solution iff there exist multipliers $b_1, \cdots, b_w$ such that $q_i - q'_i = b_1 q_{1i} + \cdots + b_w q_{wi}$ for each $1 \leq i \leq n$.*

Definition 7. *A linear constraint C on an extended alphabet $\Sigma' = \{Y_0, \ldots, Y_l, a_1, \ldots, a_l\}$, which includes $X_1, \ldots, X_n, h_1 \ldots, h_k$, is written as:*

$$Y_l \succ_C a_l \succ_C \ldots \succ_C a_2 \succ_C Y_1 \succ_C a_1 \succ_C Y_0,$$

where $\{X_1, \ldots, X_n\} \subseteq \{Y_0, \ldots, Y_l\}$ and $\{h_1, \ldots, h_k\} \subseteq \{a_1, \ldots, a_l\}$.

In the above, upper case symbols are used for variables, and lower case symbols are used for constants. Extra symbols are introduced for technical reasons so that between every two variables, there is a constant in the ordering.

Definition 8. *A solution β for the above set of nonhomogeneous linear equations satisfies a linear constraint C if and only if for every $Y \in \{Y_0, \ldots, Y_l\}$, $\beta(Y)$* does not contain *any of the symbols below Y in C. In other words, if $Y \succ_C a_j$ then $\beta(Y)$ does not contain any occurrence of a_j.*

Note that among variables, Y_l is the most constrained, since it cannot contain any of $a_1, \ldots, a_l$. On the other hand, from the point of view of constants, a_1 is the most constrained since it cannot appear in any variable other than Y_0.

5.1 Solutions Satisfying a Linear Constraint

The following claim follows from the fact that any solution of NHE can be obtained from the particular solution π and a linear combination of the syzygy basis.

Claim: If there is a solution β of the set NHE of nonhomogeneous linear equations satisfying a linear constraint C, then there is a particular solution equivalent to π as constructed above satisfying C.

The goal, thus, is to find among all particular solutions of the above set NHE of nonhomogeneous equations, a solution that satisfies the linear constraint C. As stated in the above proposition, from a solution $\pi' = (q'_1, \ldots, q'_n)$ of NHE, it is possible to obtain another solution using the syzygy basis SB of the set HE of homogeneous equations, since SB defines an equivalence relation on solutions. In order to search for a particular solution that is equivalent to π and also satisfies C, an admissible ordering $\succ_t$ on terms (and polynomials) induced by C is defined in such a way that solutions satisfying C are minimal in this ordering.

Let $\tau_1, \ldots, \tau_n$ be new indeterminates. Consider the following set MSB of polynomials in $Z[h_1, \ldots, h_k, \tau_1, \ldots, \tau_n]$, constructed from SB:

$$\{q_{11}\tau_1 + \ldots + q_{1n}\tau_n,\ \ldots\ ,\ q_{w1}\tau_1 + \ldots + q_{wn}\tau_n\}.$$

In addition, we include in MSB, additional polynomials $\{\tau_i\tau_j \mid 1 \leq i, j \leq n\}$ so that after simplification using these rules, every polynomial under consideration is *linear* in the τ_i's. Thus we only have to consider terms of the form $h_1^{d_1} \ldots h_k^{d_k}\tau_j$, where $d_1, \ldots d_k$ are nonnegative integers. Below we define an admissible ordering $\succ_t$ on such linear terms (in τ_i's) in $Z[h_1, \ldots, h_k, \tau_1, \ldots, \tau_n]$ induced by C. This term ordering $\succ_t$ is then extended to the simplified polynomials in $Z[h_1, \ldots, h_k, \tau_1, \ldots, \tau_n]$ which are linear in the τ_i's, in the usual way [5, 10].

The ordering $\succ_t$ is used to to construct a *strong* Gröbner basis $GMSB$ for the set MSB of polynomials [10]. The polynomial $\pi_p = q_1\tau_1 + \ldots q_n\tau_n$ corresponding to the particular solution π is then normalized using the Gröbner basis $GMSB$. Since the equivalence relation induced by MSB preserves solutions of NHE, the canonical (normal) form of π_p wrt $GMSB$ also corresponds to a particular solution. If this particular solution satisfies C (i.e., all terms are *good* in the sense defined below), then we get from the canonical form of π_p, a unifier for the unification problem wrt $AGnHC$ satisfying C. If the canonical form of π_p does not satisfy C, then the unification problem wrt $AGnHC$ does not have a solution satisfying C, since no polynomial in the equivalence class of π_p satisfies C (as every polynomial in the equivalence class of π_p is bigger than or equal to the normal form of π_p wrt $\succ_t$ whereas a polynomial corresponding to a solution satisfying C must be smaller wrt $\succ_t$).

In the following subsection, such an admissible ordering $\succ_t$ induced by a linear constraint C is defined on terms in $Z[h_1, \ldots, h_k, \tau_1, \ldots, \tau_n]$ which are linear in the τ_i's (i.e., whose degree in $\{\tau_1, \ldots, \tau_n\}$ is 1).

5.2 A New Way of Defining an Admissible Ordering on Terms

It is well-known that to construct a Gröbner basis of a polynomial ideal, a total admissible term ordering is needed. An admissible ordering must satisfy two properties:

1. For any term $t \neq 1$, $t \succ_t 1$, and
2. for any terms s, t, u, if $s \succ_t t$, then $u\ s \succ_t u\ t$.

Two commonly used admissible orderings in the Gröbner basis literature are the total degree ordering and the pure lexicographic ordering induced by a total ordering on indeterminates. Below we define an admissible ordering in a radically different way.

Consider any two terms s, t in $Z[h_1, \ldots, h_k, \tau_1, \ldots, \tau_n]$ which are linear in $\{\tau_1, \ldots, \tau_n\}$. Define $s \succ_t t$ iff $nf(s) >' nf(t)$, where the function nf stands for the normal form with respect to the reduction rules defined below in order to capture the linear constraint C. After defining nf and $>'$, we show that $\succ_t$ is admissible.

The term $nf(s)$ of a term s is over the extended alphabet $\Sigma_1 = \{a_1, \ldots, a_l, v_1 \ldots, v_l, a'_1, \ldots, a'_l, t_0, t_1, \ldots, t_l\}$, where a'_i is a copy of a_i distinguishing it from a_i, v_j's are introduced to represent badness in a term (there is one for every a_j), and t_j's are introduced to stand for Y_j's. Recall that $\{h_1, \ldots, h_k\} \subseteq \{a_1, \ldots, a_l\}$ and $\{X_1, \ldots, X_n\} \subseteq \{Y_0, \ldots, Y_l\}$; thus, corresponding to every X_i, there is a $Y_j = X_i$; similarly, corresponding to each τ_i, there is a $t_j = \tau_i$, i.e., $\{\tau_1, \ldots, \tau_n\} \subseteq \{t_0, \ldots, t_l\}$.

Below, *legal* term, *good* term, and *bad* term are defined on Σ_1 based on whether the term satisfies the linear constraint C.

Definition 9. *A term $s = {a_1}^{d_1}{a_2}^{d_2}...{a_l}^{d_l}\tau_i$ is called a* legal *term (only such terms appear in the polynomials in the basis MSB and in the computation of a Gröbner basis from MSB because of rules $\tau_i\tau_j \to 0$).*

A legal term $s = {a_1}^{d_1}{a_2}^{d_2}...{a_l}^{d_l}\tau_i$ is called a good *term if for each $1 \leq j \leq l$, $a_j \succ_C X_i$ in C, i.e., s satisfies the linear constraint C with respect to X_i.*

A legal term $s = {a_1}^{d_1}{a_2}^{d_2}...{a_l}^{d_l}\tau_i$ is called a bad *term if there exists a $1 \leq j \leq l$ such that it is not the case that $a_j \succ_C X_i$ in C, i.e., s does not satisfy the linear constraint C with respect to X_i.*

(A legal term that is not good, is bad.)

To capture the restrictions imposed by the linear constraint C on terms, we define the reduction rules on legal terms as:

$$\begin{array}{ll} a_i\, t_j \to {a_i}'\, t_j & \text{if } a_i \succ_C Y_j. \\ a_i\, t_j \to {a_i}'\, v_i t_j & \text{if } a_i \not\succ_C Y_j. \end{array}$$

So the normal form of a legal term ${a_1}^{d_1}{a_2}^{d_2}...{a_l}^{d_l}\tau_i$ with respect to the above rules is either

(i) $({a_1}')^{d_1}({a_2}')^{d_2}...({a_l}')^{d_l}\tau_i$, or
(ii) $({a_1}')^{d_1}({a_2}')^{d_2}...({a_l}')^{d_l}{v_{j_1}}^{d_{j_1}}...{v_{j_u}}^{d_{j_u}}...{v_{j_k}}^{d_{j_k}}\tau_i$,

where for each $1 \leq u \leq k$, it is not the case that $a_{j_u} \succ_C Y_i$.

Let $nf(t)$ denote the normal form of a term t by the above rules.

To compare the normal forms of s and t using the above reduction rules, we define the following lexicographic ordering $>'$ on symbols in Σ_1:

$$\begin{array}{l} a_1 >' \ldots >' a_l >' v_1 >' \ldots >' v_l >' t_l >' \ldots >' t_1 \\ \quad >' t_0 >' a'_1 >' \ldots >' a'_l \end{array}$$

This ordering is extended in a natural way to terms over Σ_1.

By the above reduction rules, the normal form of a bad term is greater than the normal form of a good term because only the normal form of a bad term has some v_j's which are greater than all a'_i's and t_i's.

Below, we sketch a proof that the ordering $\succ_t$ on legal terms in $Z[h_1, \cdots, h_l, t_1, \ldots, t_n]$, defined as

$$s \succ_t t \text{ iff } nf(s) >' nf(t)$$

is admissible.

For any $s \neq 1$, it is easy to see that $s \succ_t 1$.

The following lemma ensures that if $s \succ_t t$, then for any u, $u\,s \succ_t u\,t$, by proving that $nf(s) >' nf(t)$ iff $nf(u\,s) >' nf(u\,t)$.

Lemma 5. *Let Σ_1, $>'$, and nf be as defined above. Then $nf(s) >' nf(t)$ iff $nf(us) >' nf(ut)$ for all legal terms s, t, us and ut.*

Proof-Sketch: (i) If $nf(s) >' nf(t)$ then $nf(u\ s) >' nf(u\ t)$: To prove this, it is enough to prove that for any symbol a_p in Σ_1, $nf(a_p\ s) >' nf(a_p\ t)$ if $nf(s) >' nf(t)$.

The key idea is this: since $nf(s) >' nf(t)$, multiplying by a_p on both sides could contribute either a'_p or $a'_p\ v_p$ to the normal forms. The only hard case is when $nf(s)$ contains t_i and $nf(t)$ contains t_j such that $Y_j \succ_C Y_i$ (i.e., Y_j is 'more constrained' than Y_i) and in addition, $a_p \succ_C Y_i$ and $a_p \not\succ_C Y_j$. Multiplying both sides by a_p will contribute a'_p to $nf(a_p\ s)$ and $a'_p\ v_p$ to $nf(a_p\ t)$. But since $nf(s) >' nf(t)$ there must be some a_q in s such that $a_q \not\succ_C Y_i$ and the power of a_q in s is larger than the power of a_q in t. Thus $nf(s)$ includes v_q whose power in s is larger than its power in $nf(t)$. But since $a_p \succ_C Y_i$, $v_q >' v_p$. Thus $nf(a_p\ s) >' nf(a_p\ t)$ (since the presence of v_q will "lexicographically nullify" the effect of including v_p).

(ii) If $nf(u\ s) >' nf(u\ t)$, then $nf(s) >' nf(t)$: This part is easier and similar. (Observe that $>'$ is a total ordering on terms.) □

For a detailed proof, please refer to [11].

6 Conclusion

We have presented a unification algorithm for analyzing cryptographic protocols using modular exponentiation and multiplication. This algorithm along with a related algorithm in [15] addresses theories, similar to theories arising in many cryptographic protocols including the RSA cryptosystem [19], one of the most popular public-key cryptosystems.

While the check for unifiability over the theory discussed in [15] is NP-complete, the check for unifiability for the theory discussed in this paper is likely to be worse; it can be easily shown to be EXPSPACE-hard. Unification algorithms for these theories are exponential in complexity (the algorithm in this paper is double-exponential given that the Gröbner basis algorithm used for solving syzygies is doubly-exponential). An interesting challenge is thus to identify special cases arising in cryptographic protocol analysis for which these unification algorithms are more efficient. As observed in [15], Pereira and Quisquater's algorithm [18] for analyzing the cliques protocol is not precisely a unification algorithm but it appears to be closely related, and it is possible that their approach could be applied to developing an efficient unification algorithm. Pereira and Quisquater were able to discover several security problems by solving a set of linear equations.

Also, it still remains to be seen in practice whether the integration of these unification algorithms into a software tool such as NRL Protocol Analyzer works more effectively than an approach based on narrowing implemented in it. (In the

presence of associativity and commutativity (AC) properties of certain operations, it is unclear how simple narrowing is helpful unless an AC-unification algorithm is integrated into narrowing.)

As observed in [15], it will be necessary not only to develop algorithms for particular theories relevant to cryptographic protocol analysis, but to be able to combine them at will. Most protocols make use of several different forms of encryption, depending on the needs of the application.

References

1. F. Baader and K.U. Schultz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. Proc. *11th Conference on Automated Deduction (CADE-11),* Saratoga Springs, NY, Springer LNAI 607, 1992, 50–65.
2. F. Baader. Unification in Commutative Theories, Hilbert's Basis Theorem, and Gröbner Bases. *J. ACM,* 40 (3), 1993, 477–503.
3. F. Baader and W. Nutt. Adding Homomorphisms to Commutative/Monoidal Theories, or: How Algebra Can Help in Equational Unification. Proc. *4th International Conference on Rewriting Techniques and Applications, RTA 91,* Springer LNCS 488, 1991, 124–135.
4. F. Baader and W. Snyder. *Unification Theory.* In: J.A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning. Elsevier Science Publishers, 2001.
5. B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In *Multidimensional Systems Theory* (N.K. Bose, ed.), Reichel, Dordrecht, 1985, 184–229.
6. D. Chaum. Security without Identification: Transaction Systems to Make Big Brother Obsolete. CACM 28 (10), 1985, 1030–1044.
7. J. Clark and J. Jacob. *A Survey of Authentication Protocol Literature: Version 1.0.* Unpublished Technical Report, Department of Computer Science, University of York, UK, Nov. 1997. Available at the URL: `www-users.cs.york.ac.uk/~jac/papers/drareviewps.ps`
8. M. Davis. *Computability and Unsolvability.* Dover Publications, 1982.
9. J.-M. Hullot. Canonical forms and unification. In *Proc. of the 5th conference on Automated Deduction (CADE-5),* Lecture Notes in Computer Science 87, 318–334.
10. A. Kandri-Rody and D. Kapur. Computing the Gröbner Basis of a Polynomial Ideal over Integers. Proc. *Third MACSYMA Users' Conference,* Schenectady, NY, July 1984, 436–451.
11. D. Kapur, P. Narendran, and L. Wang. *A Unification Algorithm for Analysis of Protocols with Blinded Signatures.* Technical Report, Department of Computer Science, University at Albany–SUNY, Albany, NY. Also: Technical Report, Department of Computer Science, University of New Mexico, Albuquerque, NM. July 2002.
12. D. Kapur and G. Sivakumar. A Total, Ground Path Ordering for Proving Termination of AC-Rewrite Systems. Proc. *Rewriting Techniques and Applications, 8th International Conference, RTA-97,* Sitges, Spain (ed. Comon, H.), Springer LNCS 1231, June 1997, 142-156.
13. C. Meadows. Formal Verification of Cryptographic Protcols: A Survey. Proc. *AsiaCrypt 96,* 1996.
14. C. Meadows. The NRL Protocol Analyzer: An Overview. *J. Logic Programming,* 26(2), 1996, 113–131.

15. C. Meadows and P. Narendran. A Unification Algorithm for the Group Diffie-Hellman Protocol. Presented at the *Workshop on Issues in the Theory of Security (WITS 2002),* Portland, Oregon, Jan 2002.
16. P. Narendran, F. Pfenning, and R. Statman. On the Unification Problem for Cartesian Closed Categories. *Journal of Symbolic Logic,* 62 (2), June 97, 636–647.
17. W. Nutt. Unification in Monoidal Theories. Proc. *10th International Conference on Automated Deduction (CADE-10),* Kaiserslautern, West Germany, Springer LNCS 449, July 1990.
18. O. Pereira and J.-J. Quisquater. A Security Analysis of the Cliques Protocols Suites. Proc. *14th IEEE Computer Security Foundations Workshop,* June 2001, 73–81.
19. R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. CACM, 21 (2), 1978, 120–126.

Exploiting Generic Aspects of Security Models in Formal Developments*

Heiko Mantel and Axel Schairer

German Research Center for Artificial Intelligence (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{mantel,schairer}@dfki.de

Abstract. The construction of security models from scratch is a difficult, time consuming, and expensive task. In this article, we demonstrate how to exploit generic concepts, in particular the concept of secure information flow, to simplify the construction of security models. Requirements concerned with confidentiality or integrity can often be expressed nicely as restrictions on the allowed flow of information. For a verification of these restrictions, it is necessary to explicate formally what information flow means. Various information flow properties have been suggested for this purpose and we employ *MAKS*, the "Modular Assembly Kit for Security" [Man00a], for a unified perspective on these properties. How to exploit the generic security models based on secure information flow in practice is described in the context of the VSE system [AHL+00].

1 Introduction

The security of information systems has become a vital issue. To date, applications in, e.g., electronic commerce or electronic communication demand that IT-systems are trustworthy. Future applications, in particular in the area of electronic government, including e-voting, let us expect that security will become an even more important topic in the future. The main difficulty in the construction of secure systems is that these systems must operate correctly even if they are used in hostile environments. However, difficulties arise not only from possible malicious attacks but also from the complexity of these systems. For example, it must be taken into account that the widespread interconnection of information systems, in particular, by the Internet, allows various forms of communication. Due to this complexity, the application of formal methods in the construction of these systems appears to be most appropriate in order to ensure security. Using formal methods, the desired security policy for a given system can be specified in precise mathematical terms by a formal security model. Benefits of constructing a formal security model include that ambiguities in the description of security requirements are avoided, that the consistency of these requirements can be verified, and that it can be proved that a system satisfies its security requirements.

* This work has been partly supported by the German Research Foundation (DFG) and the German Federal Ministry of Education and Research (BMBF).

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 452–475, 2005.

Moreover, formal security model are mandatory for evaluations according to criteria like ITSEC or CC [ITS91,CC99] (level E4/EAL5 or higher).

Since formal security models closely depend on the particular application it might appear natural to construct these models from scratch for every application. However, constructing formal security models from scratch is a difficult, time consuming, and expensive task. Fortunately, certain aspects of security models are independent of the particular application. This makes it possible to construct *generic security models*, i.e. schemas for security models in which some application-specific parts are left unspecified. Usually, a security model consists of three components: a system specification that specifies how the system behaves, a specification of security properties that capture the security requirements, and a proof that establishes a satisfaction relation between these two specifications. Generic aspects can be exploited in each of these components.

Historically, early generic security models [Lam71,BL76,Bib77] have evolved from corresponding concepts for security in the pre-IT world of documents and safes. However, these models were found to have certain deficiencies when used in the context of IT-systems [Lam73,McL85]. These observations have led to the development of alternative approaches that are preferable for the specification of security requirements, in particular, the noninterference approach [GM82]. In this approach, confidentiality or integrity requirements are specified as restrictions on the allowed flow of information. The noninterference approach has subsequently been elaborated in order to better deal with nondeterminism, intransitive information flow, and compositional system development. This has led to a large collection of security properties based on the idea of secure information flow (e.g. [Sut86,McC87,JT88,GN88,O'H90,McL94a,FG95,Ros95,ZL97]). This is the class of security properties on which we focus in this article. For a unified perspective on these properties, we employ *MAKS*, the "Modular Assembly Kit for Security" [Man00a]. *MAKS* is a framework for the uniform representation of information flow properties that already has been employed for the comparison of information flow properties, for handling intransitive information flow [Man01a], and for the derivation of verification techniques [Man00b] as well as of techniques for stepwise system development [Man01b,Man02]. However, in this article we employ *MAKS* for a different purpose, i.e. to simplify the construction of generic security models in practice. For this purpose, we integrate the basic concepts of *MAKS* into an existing general-purpose tool for formal methods, the "Verification Support Environment 2" (abbreviated by VSE) [AHL$^+$00]. Conceptually, the result of our integration can be viewed as a library of VSE specifications that simplifies the construction of formal security models. We also describe procedures that can be used to translate the application-specific parts of a security model from well known general specification formalisms into the VSE specification language. Hence, our integration into VSE goes beyond a mere library in the sense that we also set up an infrastructure for a simple use of this library.

This article is structured as follows. In Sect. 2, the historical development of security models for IT-systems is reviewed from early concepts to the concept of secure information flow. The basic concepts of *MAKS* are recalled in

Sect. 3. In Sect. 4, it is described how to automatically translate system specifications based on pre- and postconditions into event-based specifications in the VSE specification language. How the modular representation of information flow properties in *MAKS* can be supported in the VSE system is described in Sect. 5. This modular representation also allows one to construct large parts of proofs of information flow properties in a generic way such that they can be used for different applications as described in Sect. 6. We conclude in Sect. 7.

2 Generic Security Models

2.1 Security Concepts from the World of Documents and Safes

Early approaches to specify security requirements have been based on concepts that were originally developed for security in the pre-IT world of documents and safes. Organizational and military security requirements usually involved the protection of valuable objects like, e.g., classified documents, from unauthorized accesses or modifications by human beings. Abstractly, security requirements of this kind can be formalized as relations between subjects, objects, and access rights where a subject s (e.g. a person) has a particular access right a (e.g. the authorization for modifications) for a given object o (e.g. a document) if the access relation holds for the triple (s, a, o). The access relation can be viewed as an access matrix, i.e. a matrix in which rows and columns are, respectively, associated with subjects and objects and the entries consist of sets of access rights. A subject is authorized for a particular access to an object if and only if the respective access right is listed in the corresponding entry of the matrix.

Since the basic concepts of access control have obvious counterparts in the world of documents and safes, access control models are quite easy to understand. However, transferring concepts from the world of documents and safes into the world of IT-systems is only possible to a limited extent. Access control has some serious shortcomings when being used to model security requirements for IT-systems. Let us briefly review some of these well-known shortcomings.

Determining all relevant subjects and objects in an IT-system is not as simple as it might seem at first sight. Experience has shown that this is an error prone task. E.g., the program counter is an object that is easily forgotten [McL94b]. However, objects that are not considered by the access control might lead to so called *covert channels* [Lam73], i.e. undesired communication channels that can be exploited by malicious programs like computer viruses or Trojan horses. Moreover, concrete accesses have to be mapped correctly to abstract access rights. However, in an IT-system, subjects access objects in many different and, sometimes, quite subtle ways. Therefore, determining a correct mapping is sometimes nontrivial [McL90]. Furthermore, it is common practice to permit certain so called *trusted processes* that need not obey the rules of the access control. Hence, these processes must be trusted. However, verifying the trustworthiness of these processes is outside the scope of access control. Finally, access rights are not static. Rather, in many cases, access rights change dynamically over time. However, a security-preserving management of these changes is error prone. A

famous example for this kind of problem is McLean's system Z [McL85,McL87] that pointed to a shortcoming of the well known Bell/LaPadula model [BL76].

These (and other) shortcomings of access control models have been the motivation for the development of security models that are based on the idea of secure information flow rather than on access control.

2.2 The Idea of Noninterference

Modeling security requirements as abstract restrictions on the allowed flow of information is a very elegant approach. In order to verify that a system complies with restrictions of this kind, it is necessary to formally define what information flow means. Various information flow properties have been proposed for this purpose. The information flow properties on which we focus in this article can all be regarded as descendants of Goguen and Meseguer's *noninterference* [GM82]. Intuitively, a group of processes is *noninterfering* with another group if the actions of the first group have no effect on the observations of the second group. Information flow properties can be used to model *confidentiality requirements* according to the following observation:

> *If a group of processes is noninterfering with another group then the first group cannot reveal any secrets to the second group.*

Although information flow properties are more popular for modeling confidentiality requirements, they can also be used to model *integrity requirements* because:

> *If a group of processes is noninterfering with another group then the integrity of the second group cannot be corrupted by the first.*

The intuitive idea of noninterference is quite appealing. However, the original formal definition of noninterference had two shortcomings: first, the system model underlying this definition is inadequate for nondeterministic systems and, second, noninterference is too restrictive wrt. intransitive information flow. To cope with nondeterminism is necessary when, e.g., investigating distributed systems or systems with random events and to cope with intransitive information flow is necessary for common features of secure systems like, e.g., information filters or explicit downgrading. Therefore, the two shortcomings of the original definition were quite severe and much research has aimed at overcoming them without dropping the appealing intuitive idea that underlies noninterference.

2.3 State of the Art in Research on Information Flow Properties

Starting with Sutherland's nondeducibility [Sut86], numerous information flow properties have been proposed that can be regarded as descendants of noninterference but that are more suitable for nondeterministic systems than noninterference (e.g. [McC87,JT88,GN88,WJ90,O'H90,McL94a,FG95,Ros95,ZL97]). In order to simplify their application, most of these information flow properties are based on possibilistic system models that describe systems in terms of their *possible* executions without considering the probabilities of these executions (in contrast to more complex probabilistic models). Variants of noninterference that are

suitable for dealing with intransitive information flow have been suggested in [HY87,Rus92,Pin95,RG99] for deterministic systems and in [Man01a] for nondeterministic systems[1]. The verification of information flow properties also has received considerable attention. Usually, a technique called *unwinding* is employed in the verification of these properties. Unwinding of noninterference was first suggested in [GM84] and subsequently has been refined as well as extended to other information flow properties [HY87,Rya91,Rus92,Mil94,Pin95,Zak96,Man00b]. Another important research topic has been the preservation of information flow properties under stepwise development, i.e. under composition and refinement. The known compositionality results constitute one of the main strengths of this approach [McC87,JT88,Rya91,McL94a,FG95,RW95,ZL98,Jür00,Sch01,Man02]. For many information flow properties it is known whether they, in general, are preserved under compositions or not. For some of the properties that are not preserved in general, it is known how composition can be restricted in order to preserve these properties. In contrast to this, most information flow properties that are adequate for nondeterministic systems are not preserved under trace refinement [Jac89,Rya91,McL92]. This observation is often referred to as the *refinement paradox*. However, technically, it is an immediate consequence of the fact that information flow properties are properties of sets of traces [McL94b]. A solution to circumvent the refinement paradox is to admit only restricted forms of trace refinement by refinement operators [Man01b]. Information flow properties are generic concepts for modeling confidentiality and integrity requirements. However, to date, the application of information flow properties has mostly focused on specifying and verifying security requirements of operating systems [SRS+00] and on protocol verification [FGG97,FGM00].

Several comparisons of possibilistic information flow properties have been performed in order to better understand these properties and their relation to each other [McL94a,FG95,ZL97,RS99,Man00a]. Some of these comparisons have been simplified by the use of frameworks for the uniform representation of information flow properties.

3 Modular Assembly Kit for Secure Information Flow

In this article, we employ one of these frameworks, the "Modular Assembly Kit for Security" (abbreviated by *MAKS*), that has been previously proposed by one of the authors in [Man00a] and subsequently extended by verification techniques [Man00b], concepts for intransitive information flow [Man01a], and foundations for the stepwise development of secure systems [Man01b,Man02].

In *MAKS*, an *information flow property* is represented by two elements: a flow policy and a security predicate. The purpose of the *flow policy* is to define the application-specific restrictions on the allowed flow of information. Thereby, confidentiality and integrity requirements can be expressed. The *security predicate* gives these restrictions a precise formal semantics. A specialty of *MAKS* is that security predicates are represented in a modular way, i.e. as the conjunction

[1] The approach in [RG99] can cope with some but only very limited nondeterminism.

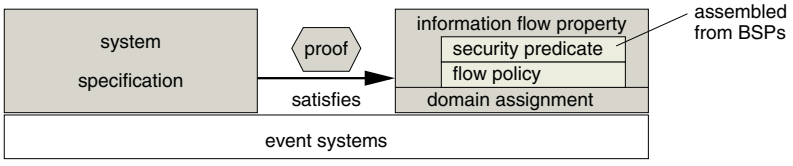


Fig. 1. Structure of a security model based on *MAKS*.

of *basic security predicates* (abbreviated by *BSPs*). An information flow property can be connected to a given system specification by means of a *domain assignment*. For the specification of systems, *MAKS* assumes a trace-based system model, i.e. event systems [JT88], that can be regarded as a descendant of Hoare's trace-semantics for the process algebra CSP [Hoa85]. The resulting structure of a security model based on *MAKS* is summarized in Fig. 1.

3.1 System Specifications

The behavior of a system can often be adequately specified by the set of its possible execution sequences. We follow this trace-based approach throughout this article. A *trace* is a sequence of events that models one possible execution. An *event* is an atomic action like, e.g., sending or receiving a message. For a given system, we distinguish between input, output, and internal events. The underlying intuition is that input events are controlled by the environment while output and internal events are controlled by the system. When a system is capable to prevent occurrences of input events, then this can be regarded as a signal to the environment. To avoid this kind of communication, input totality is often assumed, i.e. that a system cannot prevent occurrences of input events. Since input totality is quite restrictive, we refrain from making this restriction a general assumption in this article.

The system model that we employ is that of event systems [JT88]. An *event system ES* is a quadruple (E, I, O, Tr) where E is a *set of events*, $I, O \subseteq E$, respectively, are the *sets of input and output events*, and $Tr \subseteq E^*$ is the *set of possible traces*, i.e. a set of finite sequences over E. Each trace $\tau \in Tr$ models a possible behavior of *ES*. The set Tr must be closed under prefixes, i.e. any prefix of a trace in Tr must also be contained in Tr. Event systems allow for the specification of nondeterministic systems where nondeterminism is reflected by the choice between different events. Event systems constitute a possibilistic system model that abstracts from probabilities.

3.2 Security Properties

Security requirements can be expressed as restrictions on the allowed flow of information within a system. To express confidentiality or integrity by such restrictions is the key idea of information flow control. In *MAKS*, the specification of an *information flow property* consists of a flow policy and a security predicate.

A *flow policy FP* is a tuple $(\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ that specifies restrictions on the information flow within a system where $\mathcal{D}$ defines a set of security domains,

the relations $\rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow \subseteq \mathcal{D}\times\mathcal{D}$ must form a disjoint partition of $\mathcal{D}\times\mathcal{D}$, and $\rightsquigarrow_V$ must be reflexive. Typical security domains denote, e.g., groups of users, collections of files, or memory sections. The *noninterference relation* $\not\rightsquigarrow$ specifies where information flow between domains is forbidden. E.g., $D_1 \not\rightsquigarrow D_2$ expresses that *information must not flow* from D_1 to D_2. The *interference relation* $\rightsquigarrow_V$ specifies that activities of certain domains are directly visible for others. E.g., $D_1 \rightsquigarrow_V D_2$ expresses that activities of D_1 *are visible* for D_2. Finally, the relation $\rightsquigarrow_N$ specifies between which domains information flow is *not* restricted. E.g., $D_1 \rightsquigarrow_N D_2$ expresses that *we do not care* if information about D_1 is deducible for D_2. However, activities of D_1 must not be visible for D_2 (in contrast to $\rightsquigarrow_V$). Note that for any two domains either $D_1 \not\rightsquigarrow D_2$, $D_1 \rightsquigarrow_V D_2$, or $D_1 \rightsquigarrow_N D_2$ holds because $\rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow \subseteq \mathcal{D}\times\mathcal{D}$ is a disjoint partition of $\mathcal{D}\times\mathcal{D}$. If $\rightsquigarrow_V$ is a transitive relation then *FP* is called *transitive* and, otherwise, *intransitive*. In this article, we will only consider transitive flow policies.

A *domain assignment* is a function $dom : E \to \mathcal{D}$ that assigns domains to events. Thereby, a domain assignment links an information flow property to a system specification. We often leave *dom* implicit and denote the set of all events that are associated with a given domain D also by D, the name of the security domain. We also use that name in lower case, possibly with indices or primes, e.g., $d, d_1, \ldots$, to denote events with that domain.

Flow policies can be depicted as graphs where each node corresponds to a security domain. The relations $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$ are, respectively, depicted as solid, dashed, and crossed arrows. For the sake of readability, the reflexive subrelation of $\rightsquigarrow_V$ is usually omitted. This graphical representation is illustrated in Fig. 2 for the flow policies FP_{LH} and FP_{LHI}. FP_{LH} consists of two domains H (high-level events) and L (low-level events). According to FP_{LH}, occurrences of low-level events may be visible for the high-level domain ($L \rightsquigarrow_V H$). Occurrences of high-level events must not be visible to L and, moreover, no information about such occurrences must be deducible for L ($H \not\rightsquigarrow L$). The flow policy FP_{LHI} results from FP_{LH} by splitting the high-level domain into two domains HI (high-level input events) and $H \backslash HI$ (high-level internal and output events). The main difference to FP_{LH} is the dashed arrow from $H \backslash HI$ to L. While occurrences of high-level inputs must not be deducible for the low-level ($HI \not\rightsquigarrow L$), we do not care if information about occurrences of other high-level events is deducible for the low-level ($H \backslash HI \rightsquigarrow_N L$).

The *view* $\mathcal{V}_D = (V, N, C)$ *for a domain* $D \in \mathcal{D}$ in *FP* under *dom* is a disjoint partition of E that is defined by $V = \{e \in E \mid dom(e) \rightsquigarrow_V D\}$, $N = \{e \in E \mid dom(e) \rightsquigarrow_N D\}$, and $C = \{e \in E \mid dom(e) \not\rightsquigarrow D\}$. Consequently, V contains all events that are *visible* for D, C contains all events *confidential* for D, and N contains all events *neither* visible nor confidential for D. The views of all domains in FP_{LH} and FP_{LHI} are depicted in Fig. 2. Among these, only the views of the low-level domain L are interesting because they give rise to nontrivial proof obligations. We abbreviate the view of L in FP_{LH} and FP_{LHI} by, respectively, $\mathcal{V}_L^{LH}$ (L visible, H confidential) and $\mathcal{V}_L^{LHI}$ (L visible, $H \cap I$ confidential).

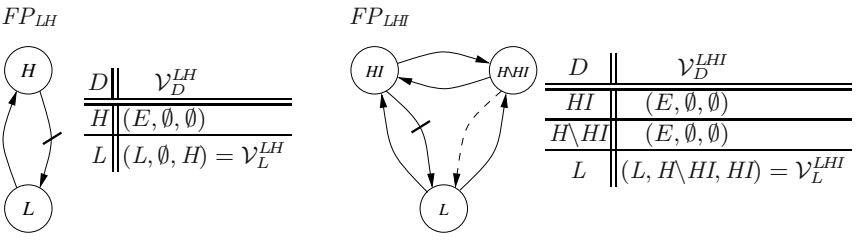


Fig. 2. The flow policies FP_{LH} and FP_{LHI} and the views of all domains.

3.3 Basic Security Predicates

Recall that an information flow property is defined by a flow policy FP and a security predicate SP. We say that an event system *satisfies an information flow property* (FP, SP) wrt. a domain assignment *dom* if SP holds for the view $\mathcal{V}$ (denoted by $SP_{\mathcal{V}}(Tr)$) of every domain in FP under *dom*. In *MAKS*, security predicates are composed by conjunction from one or more *BSPs*, i.e. $SP_{\mathcal{V}}(Tr)$ holds if $BSP_{\mathcal{V}}(Tr)$ holds for every *BSP* from which SP is composed.

BSPs are closure properties on sets of possible traces. Intuitively, a *BSP* expresses that there are *sufficiently many possible traces* such that an adversary cannot deduce confidential information of a particular kind (depending on the respective *BSP*). Various *BSPs* have been developed, however, for the purposes of this article, we focus only on three of these *BSPs*, i.e. *BSD*, *BSI*, and *BSIA* (cf. Fig. 3) and refer the interested reader to [Man00a,Man01a,Man02] for a larger collection of *BSPs*.

For example, $BSD_{\mathcal{V}}(Tr)$ (abbreviates Backwards Strict Deletion) demands that for every possible trace $\beta.\langle c\rangle.\alpha$ with $c \in C$ and $\alpha|_C = \langle\rangle$ there is another possible trace $\beta.\alpha'$ where α' may differ from α only in N-events[2]. Note that $\beta.\alpha'$ results from $\beta.\langle c\rangle.\alpha$ by *deleting* the last confidential event c (hence the term "deletion" in the name of the *BSP*) and possibly adapting N-events in α (but not in β – hence the term "backwards strict"). The requirement $BSD_{\mathcal{V}}(Tr)$ can be read as: the occurrence of a confidential event must *not add* possible V-observations. Thus, the security guarantee provided by *BSD* is: adversaries cannot deduce from any V-observation that a confidential event c *has* occurred.

$BSI_{\mathcal{V}}(Tr)$ (Backwards Strict Insertion) demands that for every confidential event $c \in C$ and every possible trace $\beta.\alpha$ with $\alpha|_C = \langle\rangle$ there is another possible trace $\beta.\langle c\rangle.\alpha'$ where α' may differ from α only in N-events. Note that $\beta.\langle c\rangle.\alpha'$ results from $\beta.\alpha$ by *inserting* c (hence the term "insertion" in the name of the *BSP*) and possibly adapting N-events in α. The requirement $BSI_{\mathcal{V}}(Tr)$ can be read as: the occurrence of a confidential event must *not remove* possible V-observations. Thus, the security guarantee provided by *BSI* is: adversaries cannot deduce from any V-observation that a confidential event c *has not* occurred.

The definition of $BSIA^{\rho}_{\mathcal{V}}(Tr)$ (Backwards Strict Insertion of Admissible events) differs from the definition of $BSI_{\mathcal{V}}(Tr)$ only in that the additional assumption

[2] The projection $\alpha|_X$ of α to $X \subseteq E$ results from α by deleting all events not in X.

$$
\begin{aligned}
BSD_{\mathcal{V}}(Tr) \equiv\ & \forall \alpha, \beta \in E^*. \forall c \in C. \\
& ((\beta.\langle c \rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
& \quad \Longrightarrow \exists \alpha' \in E^*. (\beta.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle))
\end{aligned}
$$

$$
\begin{aligned}
BSI_{\mathcal{V}}(Tr) \equiv\ & \forall \alpha, \beta \in E^*. \forall c \in C. \\
& ((\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
& \quad \Longrightarrow \exists \alpha' \in E^*. (\beta.\langle c \rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle))
\end{aligned}
$$

$$
\begin{aligned}
BSIA^{\rho}_{\mathcal{V}}(Tr) \equiv\ & \forall \alpha, \beta \in E^*. \forall c \in C. \\
& ((\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)) \\
& \quad \Longrightarrow \exists \alpha' \in E^*. (\beta.\langle c \rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle))
\end{aligned}
$$

Fig. 3. Formal definitions of three basic security predicates.

$Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)$ is made (cf. Fig. 3) that is defined by $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c) := \exists \gamma \in E^*.(\gamma.\langle c \rangle \in Tr \wedge \gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})})$ where ρ is a function from views in E to subsets of E. I.e. $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)$ expresses that there is some trace γ that has the same projection for the events in $\rho(\mathcal{V})$ as β and after that c is enabled. The motivation behind assuming ρ-admissibility in the definition of *BSIA* is to avoid a restriction to systems that behave chaotically in the set of confidential events: if $BSI_{\mathcal{V}}(Tr)$ holds then all confidential events must be enabled at any point of time. Hence, *BSI* can only be fulfilled by a system that behaves chaotically in the set C of confidential events, obviously a quite restrictive requirement. The security guarantee provided by $BSIA^{\rho}$ is: adversaries cannot deduce from any V-observation that a ρ-admissible confidential event c *has not* occurred. Consequently, the guarantee provided by $BSIA^{\rho}$ is slightly weaker than the one provided by *BSI*.

3.4 Representing Known Security Properties

MAKS is expressive enough to represent the most important (possibilistic) information flow properties that have been suggested for trace-based system models over the last 20 years. The information flow properties that have been represented in *MAKS* include *generalized noninterference* [McC87], *forward correctability* [JT88], *nondeducibility on outputs* [GN88], *noninference* [O'H90,McL94a], *generalized noninference* [McL94a], *separability* [McL94a], and the *perfect security property* [ZL97]. For a detailed description on how to represent these properties in *MAKS*, we refer the interested reader to [Man00a,Man02]. Here, we illustrate the representation of information flow properties in *MAKS* with two examples:

For example, *generalized noninterference* [McC87] can be specified in *MAKS* by choosing FP_{LHI} as flow policy (cf. Fig. 2) and the conjunction of *BSD* and *BSI* as security predicate. *Separability* [McL94a,ZL97] can be represented in *MAKS* by choosing FP_{LH} as flow policy and the conjunction of *BSD* and $BSIA^{\rho_C}$ as security predicate (where $\rho_C(\mathcal{V}) = C$ for any view $\mathcal{V} = (V, N, C)$).

The modular representation of known information flow properties in *MAKS* reduces reasoning about complex information flow properties to reasoning about simpler *BSPs* and also provides a uniform perspective on the represented proper-

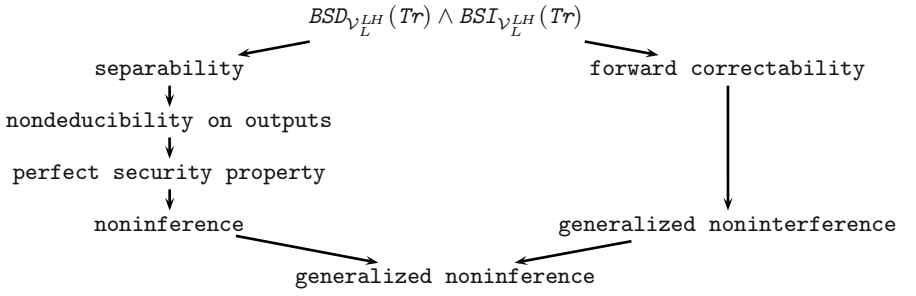


Fig. 4. Ordering of Existing Security Properties by Implication.

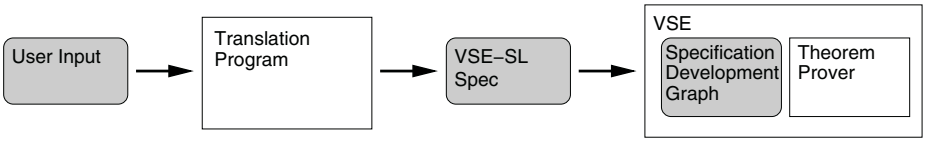


Fig. 5. Integration into VSE (architecture).

ties. Since the various information flow properties differ only in subtle technical details that are not immediately obvious from the original definitions of these properties, the representation in *MAKS* has simplified the comparison of these properties. Among other insights, this has led to a taxonomy of known information flow properties and to new information flow properties. Figure 4 illustrates the ordering of information flow properties by implication that was derived based on *MAKS* [Man02]. The top node is a novel information flow property.

4 Event-Based Specifications in VSE

In order to exploit generic security models in practice, tool support for formal modeling and verification is needed. We use the general specification and verification tool *Verification Support Environment* (VSE) [AHL+00]. Since VSE is a general-purpose tool, it does not directly support specification and verification of event systems and information flow properties as they are assumed by *MAKS*. We mitigate this by providing a library of generic specifications and by translating application-specific system and property descriptions into a form that can be used directly in VSE, cf. Fig. 5. This way, we are able to depend on all the support for formal modeling and verification that VSE offers while retaining the concepts of *MAKS*.

VSE's specification language, VSE-SL, supports axiomatic specifications of abstract datatypes, refinement of abstract datatypes using an imperative, Pascal-like programming language, and specification of concurrent reactive systems using temporal logic. In order to handle large specifications, specifications can be structured, their structure being represented in a development graph. Nodes in

a development graph represent parts of the specification, e.g. theories. Links between nodes in the graph represent either definitorial relationships, e.g. extensions of one theory by another one, or postulated relationships, e.g. one theory satisfying another one. For postulated relationships, proof obligations are generated mechanically. Tool support for verifying these proof obligations formally is provided by VSE, i.e. the verification component includes a theorem prover. The relationship between specification, proof obligations, and proofs is kept book of by VSE's correctness management, which takes care that, e.g., lemmata are verified before a proof using the lemma is considered complete.

We formalize systems and information flow properties in VSE using VSE's abstract datatypes. Rather than specifying the set of possible traces of an event system directly in the form described in Sect. 3, we use an indirect approach: we specify a state machine, i.e. a state-event system, that generates the possible traces.

A *state-event system SES* is a tuple (S, s_0, E, I, O, T) where S is a set of states, $s_0 \in S$ is the initial state, E is a set of events, $I, O \subseteq E$ are the sets of input and output events, respectively, and $T \subseteq S \times E \times S$ is a transition relation. For each $s \in S$ and $e \in E$ there must be at most one $s' \in S$ with $(s, e, s') \in T$, i.e. T is the graph of a (partial) function of type $S \times E \hookrightarrow S$. For multi-event transitions we use the notation $s \stackrel{\tau}{\Longrightarrow}_T s'$ where $\tau \in E^*$ is a finite sequence of events. The relation $\stackrel{\tau}{\Longrightarrow}_T$ is defined by $s \stackrel{\langle\rangle}{\Longrightarrow}_T s'$ iff $s = s'$ and $s \stackrel{\langle e\rangle.\tau}{\Longrightarrow}_T s'$ iff there is an $s'' \in S$ such that $(s, e, s'') \in T$ and $s'' \stackrel{\tau}{\Longrightarrow}_T s'$. If the transition relation T is obvious from the context then we omit it as index and just write $s \stackrel{\tau}{\Longrightarrow} s'$. We say that a state $s \in S$ is *reachable* for the state-event system *SES* iff $s_0 \stackrel{\tau}{\Longrightarrow} s$ for some $\tau \in E^*$, denoted by *reachable*(s). A sequence of events $\tau \in E^*$ is *enabled* in a state $s \in S$, denoted by *enabled*(s, τ), iff there exists a state $s' \in S$ such that $s \stackrel{\tau}{\Longrightarrow} s'$ holds.

A sequence $\tau \in E^*$ of events is a *possible trace* of a state-event system *SES* if it is enabled in the initial state, i.e. if there is a state $s \in S$ for which $s_0 \stackrel{\tau}{\Longrightarrow} s$ holds. The set of all traces that are possible for *SES* is denoted by Tr_{SES}. We omit the index and simply write Tr if the state-event system is obvious from the context. Consequently, every state-event system $SES = (S, s_0, E, I, O, T)$ induces an event system $ES_{SES} = (E, I, O, Tr_{SES})$.

State-event systems are specified as abstract datatypes in VSE. Specifying a given state-event system in VSE-SL, VSE's specification language, involves the specification of several concepts. In particular these include the concrete set of states S, set of events E, sets of input and output events I and O, respectively, initial state s_0, and transition relation T. Other concepts that need to be specified include event sequences, $\stackrel{\tau}{\Longrightarrow}$, *reachable*, *enabled*, and Tr_{SES}. In principle, all these could be specified by the user in VSE-SL directly. However, in order to make life easier for the user, we support a more convenient syntax for state-event systems. A translation program is provided to translate from this syntax into VSE-SL, cf. Fig. 5. The benefit of this approach is that the user can concentrate on the domain-specific aspects, e.g. the transition relation, without having to bother with our encoding of state-event systems into VSE-SL.

The behavior of a given system is modeled by the transition relation T of the state-event system. In the next section we introduce a convenient notation for specifying a transition relation. This syntax is supported by the translation procedure.

4.1 Pre/Postcondition-Statements

Pre/postcondition-statements (abbreviated by *PP-statements*) can be used to specify the transition relation of a state-event system. PP-statements presume a specific notion of state, i.e. that *a state is a mapping from state variables to values*. Given a set E of events and a set $VAR = \{var_1, \ldots, var_n\}$ of n distinct variables, a PP-statement `Stat` restricts the possible transitions for some event $e_i \in E$ based on the notation

$$\begin{array}{l} e_i \texttt{ affects } var_{j_1}, \ldots, var_{j_k} \\ \texttt{Pre } : P(var_1, \ldots, var_n) \\ \texttt{Post} : Q(var_1, \ldots, var_n, var'_1, \ldots, var'_n) \end{array} \tag{1}$$

where $1 \leq j_l \leq n$ $(1 \leq l \leq k)$. The `affects` slot specifies that an occurrence of e_i may only affect the value of state variables in $var_{j_1}, \ldots, var_{j_k}$. The values of all other state variables remain unchanged when e_i occurs. The precondition slot of the PP-statement specifies that e_i is only *enabled* in states for which the condition P is satisfied. P must be a first-order logic formula that contains no primed state variables and the postcondition Q must be a first-order logic formula that may contain primed as well as unprimed state variables. Using primed variables other than $var_{j_1}, \ldots, var_{j_k}$ in Q is not ruled out but should be avoided for reasons that will be explained later.

The semantics of PP-statements is given by a translation into higher-order formulas. The above PP-statement `Stat` translates into the following formula where the free variables s, e, s' are implicitly universally quantified:

$$\begin{array}{rl} (s, e, s') \in T_{\texttt{Stat}} \iff [e = e_i \Rightarrow (\forall var \notin \{var_{j_1}, \ldots, var_{j_k}\}.\, s(var) = s'(var) & \\ \qquad \wedge\, P(s(var_1), \ldots, s(var_n)) & \\ \qquad \wedge\, Q(s(var_1), \ldots, s(var_n), s'(var_1), \ldots, s'(var_n))] & \end{array}$$

Hence, each PP-statement `Stat` in a specification specifies a respective transition relation $T_{\texttt{Stat}} \subseteq S \times E \times S$. According to the above formula, a transition (s, e_i, s') complies with $T_{\texttt{Stat}}$ if and only if all frame axioms hold for s, s' (values of variables not in $var_{j_1}, \ldots, var_{j_k}$ remain unchanged), the precondition P holds for s, and the postcondition Q holds for s, s'. For all events besides e_i, $T_{\texttt{Stat}}$ permits arbitrary transitions, i.e. $\forall e \in E. \forall s, s' \in S. (e \neq e_i \Longrightarrow (s, e, s') \in T_{\texttt{Stat}})$. Note that using primed state variables other than $var_{j_1}, \ldots, var_{j_k}$ in the postcondition Q may lead to a contradiction with the frame axioms. This is the reason why all primed variables that occur in Q should be listed in the affects slot.

The *pre-/postcondition-specification* (abbreviated by *PP-specification* in the following) of a transition relation consists of a set `Spec` of PP-statements. In practice, a PP-specification usually contains exactly one PP-statement for every

event in E.[3] The semantics of a PP-specification `Spec` is given by the following translation into a higher-order formula:

$$(s, e, s') \in T_{\texttt{Spec}} \iff \bigwedge_{\texttt{Stat} \in \texttt{Spec}} (s, e, s') \in T_{\texttt{Stat}}$$

I.e. a transition (s, e, s') complies with the transition relation $T_{\texttt{Spec}}$ specified by a PP-specification `Spec` if and only if it complies with each transition relation $T_{\texttt{Stat}}$ specified by some PP-statement `Stat` in `Spec`.

Remark 1. For notational convenience, we permit the use of place holders in PP-statements. A parametric PP-statement for a list $x_1, \ldots, x_m$ of (typed) place holders has the following form:

$$\begin{array}{l} e_i(x_1, \ldots, x_m)\ \texttt{affects}\ var_{j_1}, \ldots, var_{j_k} \\ \texttt{Pre}\ \ : P_{x_1,\ldots,x_m}(var_1, \ldots, var_n) \\ \texttt{Post} : Q_{x_1,\ldots,x_m}(var_1, \ldots, var_n, var'_1, \ldots, var'_n)\ . \end{array}$$

As usual, parametric PP-statements denote the set of all grounded PP-statements that are (type correct) instantiations of these parametric statements.

Example 1. In Fig. 6, a syntactic specification with parametric PP-statements is illustrated. The state-event system denoted by this specification models a random generator that can output any sequence of natural numbers (events $\texttt{out}(n)$) before it terminates (event `term`). Possible traces of this system have the form $\langle \texttt{out}(n_1) \ldots \texttt{out}(n_m) \rangle . \langle \texttt{term} \rangle$ or $\langle \texttt{out}(n_1) \ldots \texttt{out}(n_m) \rangle$ (respectively models that the sequence $n_1 \ldots . n_m$ has been output and that the system has or has not terminated). The dash in the `affects` slot of the PP-statement for `out` indicates the empty list of state variables, i.e. occurrences of `out` do not affect any state variables. ◇

4.2 Translation into VSE-SL

Our approach provides a translation of user input from a convenient, special purpose language into VSE's general specification language. The user input to the translation program includes the system specification, a description of the system's postulated security property, and auxiliary information for the verification. This is visualized in Fig. 7, where the high-level structure of the resulting specification is shown as a development graph, i.e. VSE's representation of structured specifications. In this section, we are only concerned with the translation of the system specification, i.e. the subgraph labeled "system" in the figure. Other aspects will be described later: Section 5 deals with the postulated security property, while the verification is described in Sect. 6.

[3] If there are multiple PP-statement for some event $e \in E$ then these PP-statements can be merged by (1) conjoining the preconditions by conjunction, (2) conjoining the postconditions by conjunction, and (3) intersecting the lists in the affects-slots. In order to avoid chaotic behavior, a PP-specification should contain at least one PP-statement for each event $e \in E$ because, otherwise, this event would be always enabled and could lead to arbitrary successor states.

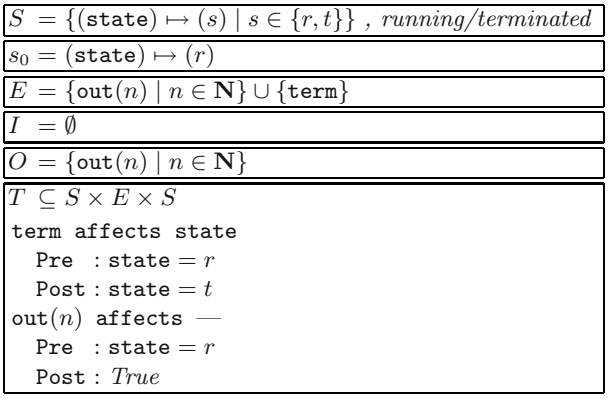


Fig. 6. Specification of the random generator by pre- and postconditions.

The essential concepts of a state-event system are specified in a structured way, cf. the right hand side of Fig. 8. The state space S is specified in theory `State`, the set of events E in `Event`, initial state s_0 and transition relation T in `TransInit`, and the set of input and output events I and O in theory `InOut`. The theory `StateEventSystem` collects together the other specifications and represents the state-event system as a whole.

The state space S is modeled in VSE-SL as a freely generated datatype `state` in theory `State`, i.e.

`TYPE state = FREELY GENERATED BY mk_state(`$var_1 : T_1, \ldots, var_m : T_m$`)`

where $var_j : T_j$ is a selector of type T_j. For a given state `s: state`, var_j(`s`) denotes the value of the j-th variable in `s`. Since we have to quantify over states and VSE does not support higher-order quantification, we use this notation rather than the functional notation $s(var_j)$ that we used in Sect. 4.1. However, the two approaches are equivalent.

The definition of `state` depends on the user input. The sequence of typed state variables $var_1 : T_1, \ldots, var_m : T_m$ is given by the user. This presupposes that the types T_i $(1 \leq i \leq n)$, which are application specific, have been specified by the user. Their definitions are imported by the theory `State`. To this effect, a set of application-specific theory names is given in the translation input, and these theories are imported by the relevant theories generated by the translation, cf. the lower right part of Fig 8.

The set of events E is also modeled as a freely generated datatype `event` in `Event`. Its definition includes a constructor for each event, i.e.

`TYPE event = FREELY GENERATED BY` $\cdots \mid e_i(sel_1 : T'_1, \ldots, sel_m : T'_m) \mid \cdots$

for a parametric event $e_i(x_1 : T'_1, \ldots, x_m : T'_m)$, where the names sel_j of the selectors are constructed from the name e_i of the event and the names of the place holders, i.e. x_j, to avoid name conflicts when the same place holder is used in several events. The set of events is extracted from the PP-statements in the user input.

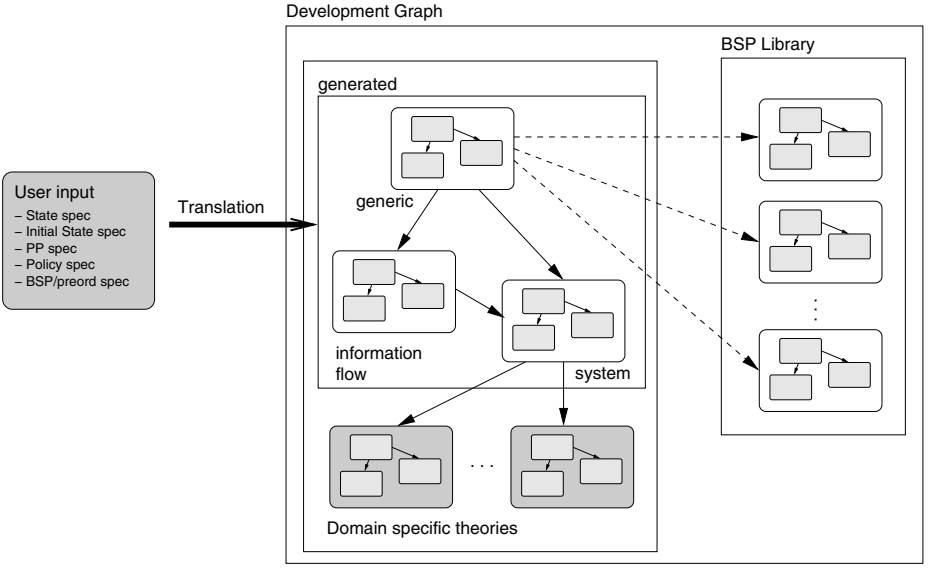


Fig. 7. Generation of the development graph for the overall specification.

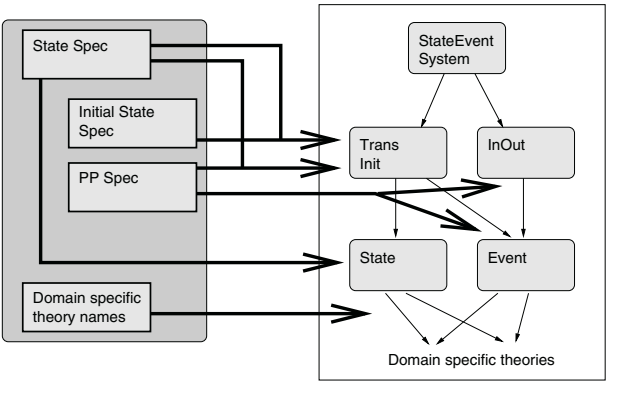


Fig. 8. Application-specific input and output for generating the system specification.

The transition relation T is modeled as a relation `T` in `TransInit`. `T` is defined in terms of the relations `pre` and `post` by the following equivalence:

$$\texttt{ALL } s_1\texttt{, e, } s_2\texttt{: T(}s_1\texttt{, e, } s_2\texttt{) <-> (pre(}s_1\texttt{, e) AND post(}s_1\texttt{, e, } s_2\texttt{))}$$

where the first-order formula $\texttt{T(}s_1\texttt{,e,}s_2\texttt{)}$ corresponds to $(s_1,e,s_2)\in T$ in Sect. 4.1. For each PP-statement

$$\begin{aligned} &e_i(x_1 : T'_1, \ldots, x_m : T'_m)\ \texttt{affects}\ var_{j_1}, \ldots, var_{j_k} \\ &\texttt{Pre}\ \ : P_{x_1,\ldots,x_n}(var_1, \ldots, var_n) \\ &\texttt{Post} : Q_{x_1,\ldots,x_n}(var_1, \ldots, var_n, var'_1, \ldots, var'_n)\ . \end{aligned}$$

in the user input, one axiom for each of `pre` and `post`, respectively, is added to `TransInit` as follows:

$$\texttt{pre(s}_1, e_i(x_1, \ldots, x_m)\texttt{) <-> } \tilde{P}$$
$$\texttt{post(s}_1, e_i(x_1, \ldots, x_m), \texttt{s}_2\texttt{) <-> } \tilde{Q} \texttt{ AND } U$$

$\tilde{P}$ and $\tilde{Q}$ are generated from $P_{x_1,\ldots,x_n}$ and $Q_{x_1,\ldots,x_n}$ by substituting $var_i(\texttt{s}_1)$ for var_i and $var_i(\texttt{s}_2)$ for var_i'. U is a conjunction of frame axioms $var_l(\texttt{s}_1) = var_l(\texttt{s}_2)$ for each $l \notin \{j_1, \ldots, j_k\}$, i.e. for each state variable that is not listed in the `affects` slot of the PP-statement. The translation checks that there is exactly one PP-statement for each parametric event $e_i(x_1, \ldots, x_m)$.

The initial state s_0 is modeled as a predicate `initialstate(s) <->` $\tilde{R}$, where $\tilde{R}$ is generated from a formula R that is part of the user input by substituting $var_i(\texttt{s})$ for var_i. Note that our definition of state-event systems in Sect. 3.1 requires there to be at most one initial state, and at most one successor state for any given state and event. These properties are not checked here but are verified as part of the verification process, cf. Sect. 6.2.

The sets of input events I and output events O are formalized by predicates `input` and `output`, respectively, in `InOut`. For each input event $e_i(x_1, \ldots, x_m)$, axioms `input(`$e_i(x_1, \ldots, x_m)$`)` and `NOT output(`$e_i(x_1, \ldots, x_m)$`)` are added to `InOut`, and similar for output and internal events. Whether an event e_i is an input, output, or internal event is specified in the user input as an annotation to the PP-statement for e_i.

Remark 2. The translation of PP-statements into first-order axioms preserves the semantics of the transition relation given in Sect. 4.1. Recall that there is a one-to-one correspondence between states as assumed in Sect. 4.1 and states as defined in this section. Assuming that s_1 and s_2 correspond to $\texttt{s}_1$ and $\texttt{s}_2$, respectively, $(s_1, e, s_2) \in T$ holds iff $\texttt{T(s}_1, \texttt{e}, \texttt{s}_2\texttt{)}$ holds. The proof of this fact is outside the scope of this paper, however. It depends on the fact that there is exactly one parametric PP-statement per parametric event.

This concludes the description of the details of the translation of system descriptions into VSE-SL. The translation from a compact user input reduces the effort needed to formulate the system specification. It also makes changing the system specification easier: changing, e.g., an event description necessitates consistent changes to `event`, `pre`, `post`, `input`, and `output`, which is taken care of by the translation procedure (rather than by the user).

5 Assembling Security Properties in VSE

We will now describe how security properties are formalized in VSE and how they are related to the system model. Like the system specification, the specification of the security property is generated from user input by the translation procedure.

In *MAKS*, a security property consists of a flow policy and a security predicate and is linked to the system by a domain assignment *dom*. In VSE, a flow policy $(\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ is formalized as a datatype `domain` that represents $\mathcal{D}$ and predicates `interferesV`, `interferesN`, and `noninterferes` that model $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$, respectively. The definition for `domain` is constructed from the set of domains given by the user input. The relations `interferesV`, `interferesN`,

and `noninterferes` are specified explicitly as part of the user input to the translation and the specification is integrated into the generated VSE development graph[4]. Restrictions on the relations forming the flow policy are verified as part of verifying the information flow property, cf. Sect. 6.2.

The domain assignment function *dom* is modeled by a function `dom` that maps events to domains. In a system description, each parametric PP-statement for events $e(x_1, \ldots, x_m)$ is annotated with the domain D to which e-events are mapped. From this annotation an axiom `dom(`$e(x_1, \ldots, x_m)$`) = `D is generated.

A security predicate in *MAKS* is a conjunction of *BSPs*, i.e. a system needs to satisfy each *BSP* in order to satisfy the overall security predicate. Specific *BSPs* from *MAKS* are predefined in VSE, cf. Fig. 7. The conjunction of *BSPs* of a postulated security predicate for a given system is represented in VSE by a set of postulated links to the effect that the state-event system representing the system satisfies all *BSPs* in its security predicate.

As an example for the formalization of a *BSP* in VSE, we consider the specification of *BSD* as defined in Sect. 3.3.

```
VARS D: domain
AXIOMS all D: BSDof(D)
```

I.e. *BSD* has to hold for the view of every domain `D`. For the view $\mathcal{V}_{\mathtt{D}} = (V, N, C)$ of domain `D`, `BSDof` is specified as[5]

```
BSDof(D) <->
ALL alpha, beta, e:
           Cof(D, e)
       AND Tr(append(extend(beta, e), alpha))
       AND projCof(D, alpha) = empty
    -> EX alphaprime:
              Tr(append(beta, alphaprime))
          AND projVof(D, alphaprime) = projVof(D, alpha)
          AND projCof(D, alphaprime) = empty
```

The specification of *BSD*, and of *BSPs* in general, is parametric in the event system and the flow policy. I.e. the specification of *BSD* above is a generic specification and, e.g., `Tr` and `interferesV` (used in the specification of `projVof`) are parameters. In this way, generic specifications of *BSPs* can be collected and can be instantiated for particular applications.

The benefit is that the specification of the flow property is generated mechanically from the user input. *BSPs* are predefined and need not be specified by users. Users only have to specify the flow policy and which *BSPs* they want to use as conjuncts in the security predicate.

[4] For the special case of a finite enumeration of domains, the translation program accepts a very compact description of `noninterferes` and a subset of `interferesV`. It then extends `interferesV` to the reflexive, transitive closure of the given subset, and defines `interferesN` such that the three relations form a disjoint union of `domain` $\times$ `domain`.

[5] `Cof`(D, e) holds if $e \in C$, `projCof`(D, α) formalizes $\alpha|_C$, and `projVof`(D, α) formalizes $\alpha|_V$.

6 Verifying Secure Information Flow

Defining information flow properties in terms of whole traces improves the understandability of these properties. However, for proving them it would be better to have a more local formulation in terms of single events. This is the idea of unwinding. Information flow properties are reformulated by local conditions, so called *unwinding conditions*, and an *unwinding theorem* ensures that a proof of the unwinding conditions implies that the global definition is valid.

In this section, we present unwinding conditions that are appropriate for verifying *BSPs* and explain how to integrate unwinding into VSE.

6.1 Modular Unwinding with *MAKS*

Recall that an information flow property (FP, SP) is satisfied for a given state-event system $SES = (S, s_0, E, I, O, T)$ if $SP_{\mathcal{V}}(Tr_{SES})$ holds for the view $\mathcal{V}$ of every domain in FP under the given domain assignment *dom*. Security predicates are assembled from more primitive *BSPs* by conjunction. Consequently, $SP_{\mathcal{V}}(Tr_{SES})$ holds if and only if $BSP_{\mathcal{V}}(Tr_{SES})$ holds for each *BSP* from which *SP* is assembled. I.e. based on the modular representation in *MAKS*, the task to prove a complex information flow property can be reduced to proving simpler *BSPs*.

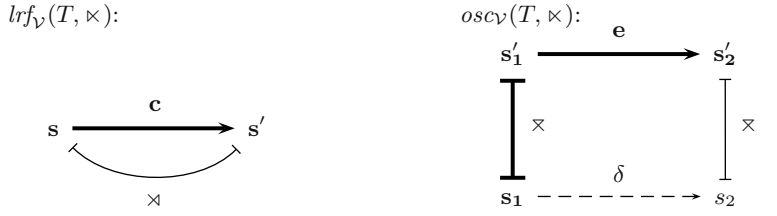
Verifying *BSPs* by unwinding involves two subtasks. Firstly, for each view an *unwinding relation* must be constructed, i.e. a binary relation $\ltimes : S \times S$ between states that must have certain properties[6]. Secondly, unwinding conditions that involve the chosen unwinding relation must be verified. Which unwinding conditions have to be verified depends on the particular *BSP*. For example, the unwinding conditions for *BSD* are $lrf_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$. For a view $\mathcal{V} = (V, N, C)$, these conditions are defined as follows (also cf. Fig. 9):

$$lrf_{\mathcal{V}}(T, \ltimes) : \forall s, s' \in S.\, \forall c \in C.\\ ((reachable(s) \wedge (s, c, s') \in T) \Longrightarrow s' \ltimes s)$$

$$osc_{\mathcal{V}}(T, \ltimes) : \forall s_1, s_1', s_2' \in S.\, \forall e \in V \cup N.\\ [(reachable(s_1) \wedge reachable(s_1') \wedge s_1' \ltimes s_1 \wedge (s_1', e, s_2') \in T)\\ \Rightarrow \exists s_2 \in S.\, \exists \delta \in E^*.\, (\delta|_C = \langle\rangle \wedge \delta|_V = \langle e \rangle|_V \wedge s_1 \stackrel{\delta}{\Longrightarrow}_T s_2 \wedge s_2' \ltimes s_2)]$$

Roughly, the unwinding condition *osc* demands that if $s_1' \ltimes s_1$ holds, then every event $e \in V \cup N$ that is enabled in s_1' must also be enabled in s_1. Moreover, the resulting states s_2' and s_2 must also be related by $\ltimes$. These properties can be extended to sequences of events by an inductive argument. Consequently, $s_1' \ltimes s_1$ can be read as: every V-observation that is possible in s_1' is also possible in s_1. The unwinding condition *osc* is not specific to *BSD*. Rather, the purpose of *osc* is to ensure that a sensible unwinding relation has been chosen. The other

[6] Traditionally, only equivalence relations have been employed for unwinding. However, a restriction to equivalence relations is not only unnecessary but is quite restrictive. Rather, pre-orders, i.e. relations that are reflexive and transitive but not necessarily symmetric, can be used as unwinding relations [Man00b].



where $c \in C$, $e \in V \cup N$, *and* $\delta \in (N \cup V)$ *with* $\delta|_V = \langle e \rangle|_V$

Fig. 9. Unwinding conditions for *BSD*.

unwinding condition, i.e. *lrf*, however, is specific to *BSD*. *lrf* demands that the state s' after the occurrence of some confidential event c must be related to the state before this event has occurred, i.e. $s' \ltimes s$. From our above understanding of $\ltimes$, we obtain that all observations possible in s' (after c has occurred) must also be possible in s (before c has occurred). This is precisely what the definition of *BSD* demands. For a formal justification of this statement, i.e. the corresponding unwinding theorem, as well as for unwinding results for other *BSPs*, we refer the interested reader to [Man00b].

6.2 Verification in VSE

In VSE, the concepts of modular unwinding are formalized as follows. The unwinding relations $\ltimes_D$ for all views $\mathcal{V}_D$ ($D \in \mathcal{D}$) are modeled together as one predicate `preord` such that `preord`(D, s_1, s_2) formalizes $s_1 \ltimes_D s_2$. Unwinding conditions for a *BSP* are formalized by axioms corresponding to the unwinding conditions.

As an example, the specification of the unwinding conditions for *BSD* read[7]

```
/* lrf */
   Cof(D, c) and reachable(s) and T(s, c, sprime)
-> preord(D, sprime, s)

/* osc */
       NOT(Cof(D, e))
   AND reachable(s1)
   AND reachable(s1prime)
   AND preord(D, s1prime, s1)
   AND T(s1prime, e, s2prime)
-> EX s2, delta:
          projCof(D, delta) = empty)
      AND projVof(D, delta) = projVof(D, extend(empty, e))
      AND Tstar(s1, delta, s2)
      AND preord(D, s2prime, s2)
```

[7] `Tstar`(s_1, γ, s_2) formalizes $s_1 \stackrel{\gamma}{\Longrightarrow} s_2$.

The specification of unwinding conditions is parametric in the event system, flow policy, and unwinding relation. Therefore, parameterized unwinding conditions can be predefined and instantiated for specific applications. A parameterized unwinding condition is associated with a parameterized *BSP* such that an instantiation of the unwinding conditions for a given system (subject to side conditions, see below) implies that the system satisfies the *BSP*. The association is due to the respective unwinding theorem from [Man00b]. This has the benefit that, for a given system, users only need to prove that the system satisfies the local unwinding relations and side conditions, rather than the global *BSP*s.

In order to verify that a system model satisfies a *BSP*, the local conditions to be proven consist of the following non-trivial proof obligations:

1. The unwinding conditions are satisfied.
2. The following side conditions are satisfied:
   (a) There is exactly one initial state, i.e. exactly one state s for which `initialstate`(s) holds.
   (b) The transition relation `trans` is the graph of a partial function.
   (c) `interferesV`, `interferesN`, and `noninterferes` form a disjoint partition of `domain` $\times$ `domain`, and `interferesV` is reflexive and transitive.
   (d) For all $D \in$ `domain`, `preord`(D, _, _) is a pre-order.

The collection of unwinding conditions and the translation process are set up such that these proof obligations are generated and managed by VSE's general-purpose development graph facility and correctness management. The benefit of this is that the user need not be concerned with whether all necessary proof obligations have been considered but can rely on VSE's correctness management to take care of this.

When a system's postulated security predicate consists of more than one *BSP*, there is a verification task including the proof obligations given above for each of the *BSPs* used. The modular structure of the security predicate leads to a modular structure of the verification tasks. This structure is exploited to share proof effort between the verification tasks for different *BSPs*. In particular, the side conditions 2a, 2b, and 2c need only be proved once because they are identical for all verification tasks.

If the same pre-order `preord` is used for the verification of more than one *BSP*, then proof obligation 2d also needs to be proved only once. In this case a proof for *osc* can also be shared between all the *BSPs* using the same `preord`, because all *BSPs* have the unwinding condition *osc* in common. Using VSE's generic support for lemmata and its correctness management, structuring the verification problems this way and factoring out common proofs is possible without further effort.

7 Conclusions and Future Directions

In this article, we have described how generic aspects of security models can be exploited in formal developments. The class of security properties that we considered, information flow properties, evolved from noninterference [GM82].

This class of properties can be used to formalize security requirements that are concerned with confidentiality and integrity in a very elegant way. For a uniform perspective on the various information flow properties, we employed *MAKS*, the modular assembly kit for security properties [Man00a].

For our purposes, *MAKS* has turned out to be a very suitable basis. In particular, the uniform, modular representation of information flow properties in *MAKS* has simplified our work. This modular structure of information flow properties is also reflected in the generated VSE specifications. In particular, it leads to a collection of *BSP* specifications together with specifications of the corresponding unwinding conditions. Since *BSPs* and unwinding conditions are parametric, a library of them can be specified. Instantiations of these specifications can be used without any need to explicitely specify them for every application.

In order to provide tool support for using our approach in practice, we have employed an existing, general verification tool, VSE [AHL+00]. Since VSE is a general-purpose tool, it did not support the basic concepts of *MAKS* directly. For this reason, we encoded these concepts as abstract data types in VSE-SL. The benefit of this approach is that the advantages of *MAKS* are combined with the powerful features of VSE, including the structuring of large scale specifications, the management of these specifications, and the theorem proving component.

Rather than enforcing that users must specify the application-specific parts of their particular security model using our encoding in VSE-SL, we offer notations that are more convenient for this purpose. The translation of the user input into VSE-SL is performed by an automatic translation procedure that we have implemented. Using this translation, users can focus on the domain-specific aspects of their application without having to worry about technicalities of the encoding. In a case study [MSK+01], we have made very positive experiences with our approach. In particular, the size of the specification that had to be typed in explicitely was reduced by an order of magnitude (in lines of specification). Given this positive experience with our prototypical implementation of the translation procedure, the described approach deserves further attention.

One valuable direction for future work would be the integration of other features of *MAKS* into VSE like, for example, principles for compositional system design [Man02], principles for stepwise refinement [Man01b], and techniques for dealing with intransitive information flow [Man01a]. Another possibility would be the integration of *MAKS* into other tools like, for example, MAYA [AHMS02] and INKA [AHMS99]. It would also be desirable to further improve the prototypical implementation of the translation procedure.

References

[AHL+00] S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, W. Stephan, R. Vogt, and A. Wolpers. VSE: Formal Methods Meet Industrial Needs. *Special Issue on Mechanized Theorem Proving for Technology Transfer of the STTT-Springer International Journal on Software Tools for Technology Transfer*, 3(1):66–77, 2000.

[AHMS99] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System Description: INKA 5.0 – A Logic Voyager. In *Proceedings of 16th International Conference on Automated Deduction, CADE–16*, LNAI 1632, pages 207–211, 1999.

[AHMS02] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The Development Graph Manager MAYA. To appear in *Proceedings of the 9th International Conference on Algebraic Methodology And Software Technology, AMAST'2002*, LNCS, 2002.

[Bib77] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE, 1977.

[BL76] D. E. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, MITRE, March 1976.

[CC99] Common Criteria Project Sponsoring Organisations. Common Criteria for Information Technology Security Evaluation (CC) Version 2.1, 1999. Also appeared as ISO/IEC 15408: IT – Security techniques – Evaluation criteria for IT security.

[FG95] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[FGG97] R. Focardi, A. Ghelli, and R. Gorrieri. Using Non Interference for the Analysis of Security Protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[FGM00] R. Focardi, R. Gorrieri, and F. Martinelli. Non Interference for the Analysis of Cryptographic Protocols. In *27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, LNCS 1853, 2000.

[GM82] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[GM84] J. A. Goguen and J. Meseguer. Inference Control and Unwinding. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86, 1984.

[GN88] J. D. Guttman and M. E. Nadel. "What Needs Securing?". In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 34–57, 1988.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HY87] J. T. Haigh and W. D. Young. Extending the Noninterference Version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2):141–150, 1987.

[ITS91] Office for Official Publications of the European Communities. Information Technology Security Evaluation Criteria (ITSEC), 1991.

[Jac89] J. Jacob. On the Derivation of Secure Components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, 1989.

[JT88] D. M. Johnson and F. J. Thayer. Security and the Composition of Machines. In *Proceedings of the Computer Security Foundations Workshop*, pages 72–89, 1988.

[Jür00] J. Jürjens. Secure Information Flow for Concurrent Processes. In *Proceedings of the International Conference on Concurrency Theory, Concur 2000*, LNCS 1877, pages 395–409, 2000.

[Lam71] B. W. Lampson. Protection. In *Proceedings of 5th Princeton Conference on Information Sciences and Systems*, page 437, 1971.

[Lam73] B. W. Lampson. A Note on the Confinement Problem. *Communications of the ACS*, 16(10):613–615, 1973.

[Man00a] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, 2000.

[Man00b] H. Mantel. Unwinding Possibilistic Security Properties. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 238–254, 2000.

[Man01a] H. Mantel. Information Flow Control and Applications – Bridging a Gap. In *Proceedings of FME 2001: Formal Methods for Increasing Software Productivity*, LNCS 2021, pages 153–172, 2001.

[Man01b] H. Mantel. Preserving Information Flow Properties under Refinement. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 78–91, 2001.

[Man02] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–101, 2002.

[McC87] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 161–166, 1987.

[McL85] J. D. McLean. A Comment on the "Basic Security Theorem" of Bell and LaPadula. *Information Processing Letters*, 20:67–70, 1985.

[McL87] J. D. McLean. Reasoning about Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 123–131, 1987.

[McL90] J. D. McLean. The Specification and Modeling of Computer Security. *IEEE Computer*, 23(1):9–16, 1990.

[McL92] J. D. McLean. Proving Noninterference and Functional Correctness using Traces. *Journal of Computer Security*, 1(1):37–57, 1992.

[McL94a] J. D. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 79–93, 1994.

[McL94b] J. D. McLean. Security Models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc., 1994.

[Mil94] J. K. Millen. Unwinding Forward Correctability. In *Proceedings of the Computer Security Foundations Workshop*, pages 2–10, 1994.

[MSK+01] H. Mantel, A. Schairer, M. Kabatnik, M. Kreutzer, and A. Zugenmaier. Using Information Flow Control to Evaluate Access Protection of Location Information in Mobile Communication Networks. Technical Report 159, CS Department, University of Freiburg, 2001.

[O'H90] C. O'Halloran. A Calculus of Information Flow. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 147–159, 1990.

[Pin95] S. Pinsky. Absorbing Covers and Intransitive Non-Interference. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 102–113, 1995.

[RG99] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 228–238, 1999.

[Ros95] A. W. Roscoe. CSP and Determinism in Security Modelling. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 114–127, 1995.

[RS99] P. Y. A. Ryan and S. A. Schneider. Process Algebra and Non-interference. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 214–227, 1999.

[Rus92] J. Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical Report CSL-92-02, SRI International, 1992.

[RW95] A. W. Roscoe and L. Wulf. Composing and Decomposing Systems under Security Properties. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 9–15, 1995.

[Rya91] P. Y. A. Ryan. A CSP Formulation of Non-Interference and Unwinding. *Cipher*, pages 19–30, Winter 1991.

[Sch01] S. Schneider. May Testing, Non-interference, and Compositionality. Technical Report CSD-TR-00-02, Royal Holloway, University of London, 2001.

[SRS+00] G. Schellhorn, W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a Formal Security Model for Multiapplicative Smart Cards. In *European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 17–36, 2000.

[Sut86] D. Sutherland. A Model of Information. In *9th National Computer Security Conference*, 1986.

[WJ90] J. T. Wittbold and D. M. Johnson. Information Flow in Nondeterministic Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 144–161,1990.

[Zak96] A. Zakinthinos. *On the Composition of Security Properties*. PhD thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, 1996.

[ZL97] A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, 1997.

[ZL98] A. Zakinthinos and E. S. Lee. Composing Secure Systems that have Emergent Properties. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 117–122, 1998.

Verification Support Environment

Werner Stephan, Bruno Langenstein, Andreas Nonnengart, and Georg Rock

German Research Centre for Artificial Intelligence,
Stuhlsatzenhausweg 3,
66123 Saarbrücken, Germany
{stephan,langenstein,nonnenga,rock}@dfki.de

Abstract. Formal software development turns out to become one of the key issues in software engineering. Today an enormous variety of methods and tools exist that serve as an aid for the software engineer to formally specify and verify large-scaled systems. This paper reviews some of the most important general notions in formal software engineering and, in particular, gives an overview on VSE (*Verification Support Environment*), a tool that supports both hierarchical specification and formal verification.

1 Introduction

This paper is about the *Verification System Environment*, the basic ideas that guided its early development, its current status and recent activities, lessons learnt from its application in industrial case studies and projects, and, based on these, directions for its future development.

In 1990 the German Information Security Agency (BSI) issued a call for tender for an industrial strength tool to support software developments compliant with the high assurance levels of the German "Green Book" which was a precursor of the European and international IT security criteria, ITSEC and Common Criteria (CC), respectively.

The VSE consortium reacted with a bid basically proposing to combine case-technology with formal methodology and theorem proving support. Accordingly, the consortium included a case-tool manufacturer GPP[1] and academic partners, as there are the University of Karlsruhe[2] providing the KIV tool (and methodology) [9], the University of Saarland[3] providing the INKA tool [18] and the University of Ulm[4] providing experience in formal specification languages. Since VSE was intended for industrial use, already the first development phase included two large case studies supplied by Dornier. The IT division of this company was also in charge with the project management.

In 1994, when the first version of the VSE tool was completed, formal methods were still questioned by influential people while their advocates lacked a

[1] Gesellschaft für Prozessrechner-Programmierung.
[2] M. Heisel, W. Menzel, W. Reif, W. Stephan.
[3] D. Hutter, J.H. Siekmann.
[4] E. Canver, W.F. von Henke.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 476–493, 2005.

clear perspective on their use in industry with respect to security and safety. In the following three years VSE was used in a broad though somewhat arbitrarily chosen spectrum of industrial applications in a rather ad hoc way more or less outside the actual development processes.

Nevertheless, the tool showed its ability to cover (horizontally) the critical aspects of a large variety of systems and also (vertically) the various development stages from abstract solutions to efficient implementations. It turned out that the time experts needed to carry out interactive proofs showed the feasibility to integrate formal development into real development processes. And also, the detection of serious errors by failed proofs promised a pay-off in terms of costs.

In a second phase from 1996 up to 1999 the VSE methodology (see Section 3) was extended and a number of technical improvements suggested by the early applications were built into the system. This was done by a consortium consisting of IST[5], an offspring of GPP, the University of Ulm[6], and DFKI[7] as the main contractor.

Since 1999 VSE is maintained and further developed by DFKI. Starting with an information filter for the German armed forces, VSE was used on the formal parts in a number of commercial projects, most of them being developments according to ITSEC or CC. The actual tool development is accompanied by research on domain specific modelling techniques and the integration of formal into conventional software engineering.

Altogether, the significance of formal methods has drastically changed in the last decade. For hardware systems the use of formal methods is widely agreed and on the software side a market perspective for highly dependable systems developed with the help of formal methods has become visible. In particular, more than ten years after the first quality criteria were issued and the development of VSE was initiated, there is an emerging market for high assurance IT systems and – as a consequence of this – a demand for tools that are compliant with the requirements of these levels.

Although formally developed software still forms a niche market, the question *Should formal methods be used?* has shifted to *How should formal methods be used?*

2 Formal Development

In contrast to the early seventies of the last century we are today confronted with an enormous variety of formal approaches and tools. To obtain a classification of systems that allows us to compare VSE with other approaches we first take a look at the *software engineering process* in general.

[5] Innovative Software Technologie.
[6] M. Balser, W. Reif.
[7] S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, J.H. Siekmann, W. Stephan, A. Wolpers.

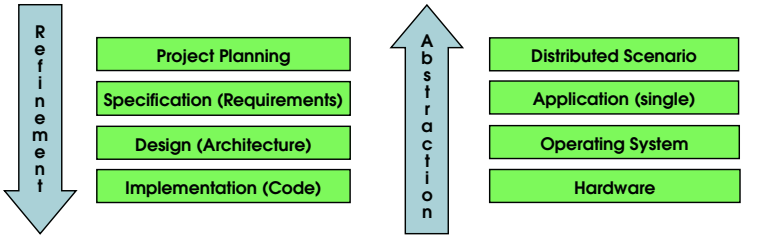


Fig. 1. Development Phases and Layers.

2.1 Software Engineering

The development of real world systems[8] is often conceived as a *deliverable oriented process*. Deliverables or artefacts include reports, design documents, reviews, and, finally, code. The result of development activities are made visible by new artefacts taking documented results of earlier stages as input.

Independently of how these activities are organised according to a particular *process model*, see for example [31] for a comparison of such models, for the purpose of this discussion we may roughly consider the *phases* shown in Figure 1.

Most software projects are not developed from scratch but utilise components that are combined and extended by new functionalities. Nevertheless, many different *layers* as depicted in Figure 1 are relevant and have to be considered in the development process. Even when the architecture is not explicitly structured into several layers services provided by a system at the top level will depend on lower level functionalities.

Application specific *aspects* determine the relevant views on a system. Each such view may require specific engineering techniques for describing and analysing technical solutions and their (desired) properties. Typically, aspects, some of which are collected in Figure 2, relate to certain layers in the general architecture and are considered in certain phases of the development process.

2.2 Formal Development Techniques

To control the engineering process artefacts have to be subject to review procedures that accompany the development. For executable artefacts quality assurance is usually obtained by testing. For the earlier phases conventional software engineering uses audits and inspections. Maybe it is due to the intangible nature of these procedures that conventional engineering techniques often do not distinguish between two basic kinds of artefacts or relations, namely those that are definitional and those that are postulated. As opposed to being definitional a postulated artefact or relation may or may not hold (or be satisfied) although for informal description techniques there is very often no way to obtain an answer that is commonly agreed upon. The question whether certain postulated relations or requirements actually hold is different from checking syntactic properties of artefacts as it is done in ordinary case tools.

[8] As opposed to the programming exercises that prevail in academic education.

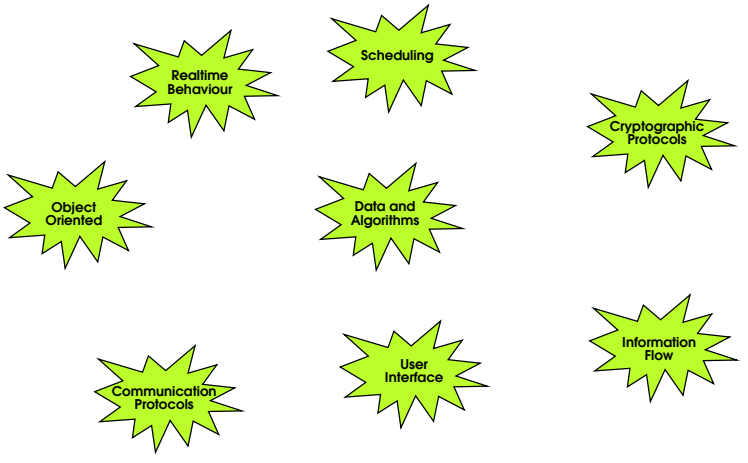


Fig. 2. Aspects.

A *formal methodology* provides description techniques for (certain) artefacts occurring in software engineering processes. Since these are underpinned by a mathematical semantics, the validity of postulated properties (requirements) and relationships can – at least in principle – be established rationally. In that sense formal methods add scientific rigour to the development process of software (or hardware).

2.3 Types of Systems

The first and perhaps most important question to be asked about tools for formal software development is concerned with the methodology they support: What kinds of artefacts and relations between these can be treated formally?

The current situation is characterised by the dichotonomy between

- highly specialised systems that are tailored for particular aspects (related to certain development phases and system layers), and
- (more) comprehensive systems that cover large parts of the overall development in a coherent way.

Typically the "small" systems are strong in the sense that they provide a fully automatic analysis of properties[9]. They are set up on problem specific representations that often are close(r) to conventional engineering practise. If doubts remain that formal software development will be dominated by loose collections of these "lightweight" domain specific tools, this is due to two main shortcomings, *instability* and *fragmentation*. This technology is often used in a way that a single critical part of a development is (manually) selected, isolated, and adapted. Even small changes in representation or size may lead to problems that are outside the scope of the tool or no longer tractable. Moreover, a rigorous

[9] This is often called a push-botton technology.

formal development of complex software systems requires an integrated analysis of various critical aspects and the coherent coverage of several architectural layers and subsequent development phases. Examples of domain specific tools are HyTech [12], UPPAAL [7], SMV [24] or KRONOS [33].

"Large" systems that support this kind of comprehensive formal development are usually based on a fairly general *specification language* and use interactive theorem proving to establish desired properties of formal models and verify correspondence relations between different design stages. Representations used in this context cannot be entirely free of some "mathematical" or "logical" overhead and interactive proof generation is and will remain (more) time consuming. Examples for methods and tools of this kind are Z [32], B [3], VDM [19], and VSE [16].

At the other end of the spectrum we find purely logical formalisms and generic problem solving techniques with no (or very few) features that are specific for software engineering. Systems of this kind range from SAT-provers, like [23], to interactive proof environments for Higher Order Logic, like [25] and [26].

Expressive logical systems like the ones mentioned above – while hardly considered as per se supporting a methodology – have been used in two ways:

- to "embed" formal approaches into software engineering as particular logical theories and
- to formalise individual systems in an ad hoc way.

As opposed to approaches where modelling techniques might change with each application, there exist systems (like VSE) that support a *fixed methodology* which is uniformly used for all developments. Such a fixed methodology gives rise to a specific *tool support* and, although somewhat restrictive, yields comparable results.

2.4 Tool Support

Methodological concepts are often *discussed* using highly abstracted and simplified examples. To *use* them for real world developments tool support is needed.

While it has been advocated in the past that (formal) modelling alone gives enough benefit, today there is a general agreement that the main advantage of formal techniques lies in the possibility to actually turn postulates into guaranteed properties of software systems. *Verification techniques*, therefore, have become the key issue in tool support.

General systems for interactive proof generation, model checking, and automated theorem proving have to be augmented by techniques that are specific for the underlying method or even for particular application areas. Improving efficiency, in particular the degree of automation in interactive theorem proving, the ease of use, and the integration of verification techniques into *case technology* for formal developments will turn academic developments into a new engineering discipline.

Case technology starts with *front ends* for editing, visualising developments. Textual representations, although still sometimes preferred by professional developers, are complemented by graphical representations of various kinds. Typically

the well formedness of specifications is guaranteed by parsing and subsequent type checking.

In many cases the specification language cannot be used directly as an input to verification mechanisms. After having parsed and typechecked the input internal structures representing developments have to be transformed into their axiomatic counterparts or representations of models.

Experimental systems are often used (and evaluated) following the cycle:

1. develop solution on paper,
2. type it into the system (and translate if necessary to prover),
3. try to prove some theorems (possibly using prefabricated paper proofs),
4. in case of problems start again.

Production systems for real world applications have to manage

- incremental extensions of partial developments,
- verification, including interactive proof discovery, and
- local changes (corrections, revisions) to already existing parts

all being done inside an integrated system by possibly several people.

To that end developments and their internal representations have to be structured into units that can be worked on locally and that are connected by well defined relationships to rule the flow of information and to be exploited by a *management of change*.

3 Methodology

A formal methodology first of all manifests itself by basic description techniques for certain elements of software engineering, like for example

- data structures,
- sequential algorithms (programs),
- states and state transitions, and
- concurrent processes

and augments them by operations for structuring specifications. Typically structure building operations include composition and refinement.

The VSE method provides an integrated specification language for *abstract data types* and *concurrent state transition systems*. In both cases VSE provides operations for *combining* specifications giving rise to the horizontal structure of developments. Modelling techniques offered by VSE are general enough to support various styles of specifications and have been used to model a broad spectrum of (types of) systems. They consider different levels of *abstraction* where notions of *refinement* that preserve established properties allow the user to transform abstract, possibly non-constructive (requirements) specifications into executable code.

3.1 Abstract Data Types

Many design steps are concerned with *data structures* and *operations* to transform them. In the *abstract data types* approach [21, 5], data objects are viewed as resulting from the nested application of certain operations, a (concrete) data type being given by a collection of domains (carriers) and (total) functions on these domains. To abstract from particular realisations, one separates syntax from semantics and considers classes of *algebras* (or models) $\mathcal{A}$ which are interpretations of a fixed collection Σ of *function symbols* f as functions $f_{\mathcal{A}}$ on the carriers of $\mathcal{A}$. In VSE classes of algebras are restricted by axioms of a first-order language over Σ [17].

The approach to *functional modelling* realised in VSE basically can be used in two principal ways. To model the state space of state transition systems data type specifications are *imported* (to state-based specifications) to describe the domains of state-depended variables and to provide the basic functions for the definition of so-called actions. However, functional modelling can also be used to describe system models of their own, for example by (inductively) defining event traces for cryptographic protocols. Just recently a method for combining independent specific "views" on a system has been developed.

In VSE there are two ways of structuring data type specifications: the actualisation of *generic* specifications and the *import* of specifications. Using import, *enrichment* and (disjoint) *union* can be modelled. Generic specifications provide an additional parameter part to describe the formal parameter including axioms. Upon actualisation of a generic theory proof obligations are generated for these axioms. Figure 3 gives an example for a parameterised specification. The parameter theory `Elems` contains the definition for a type `element`. So `list_data` can be instantiated to form lists of elements of any kind. Furthermore, `list_data` is used in the theory `list` where additional functions (`append`) and predicates (`ELEM`) are defined.

VSE implements an elaborated theory of data type refinements [28]. Operations of an (abstract) export algebra are implemented by abstract imperative programs that use operations from some (more concrete) imported algebra. The axioms of the export specification give rise to proof obligations that are assertions about the implementing programs. Properties of the import specifications are used in the course of verifying the assertions in a variant of Dynamic Logic [11].

In general, two models $\mathcal{A}_1$ and $\mathcal{A}_2$ of Ax will not necessarily be isomorphic, that is, they differ not only in the concrete representation of data objects. This allows for a really abstract style of specification where one describes *what* a function does but not *how* this is realized. A well known example is encoding and decoding. On the abstract level it might suffice to know that $dec(enc(v)) = v$ leaving open a wide range of implementations for later refinements.

Consistency of this type of specifications can be shown by refinements that end up with data types where the meaning of symbols is essentially fixed (definitions). Classes of algebras are restricted by requiring that certain carriers are *generated* by constructors from some $\Sigma' \subset \Sigma$ which means that for each element a of the corresponding carrier A, there is a term τ over Σ' that denotes a. In par-

ticular, we consider *freely* generated structures where each element $a \in A$ has a *unique* representation in Σ'. Generated clauses bring about *induction principles* that are used if inductive theorems or lemmata have to be proven. The axiomatic counterpart of these clauses is generated by the system, when the deduction unit belonging to the specification is generated.

The constructive definition of functions and relations that follows certain schemes supported by VSE (in syntax and proof theory) preserves the consistency of the corresponding basic types (and their extensions). This approach to data types is close to so-called *model based approaches* like VDM [19] and Z [32], the only difference being that the actual (mathematical) representation of data objects is left open by considering isomorphic algebras.

Figure 3 shows the parametrised specification of lists as a basic data type and the enrichment of these lists by additional functions and predicates with their axiomatisation as a theory.

```
THEORY elems
 PURPOSE "type element with default constant"
 TYPES element
 FUNCTIONS default : element
THEORYEND

BASIC list_data
 PARAMS elems
 USING NATURAL
 list = nil WITH nilp |
        cons(first : element, rest : list) WITH consp
 VARS x, y, z : list
 SIZE FUNCTION length : list -> NAT
BASICEND

THEORY list
 PURPOSE "concatenation of lists and element-predicate"
 PARAMS elems
 USING list_data[element, default]
 FUNCTIONS  append   : list, list -> list
 PREDICATES _ ELEM _ : element, list
 VARS x, y, z : list;
      a       : element
 AXIOMS
 FOR append :
   append(nil, x) = x;
   append(cons(a, x), y) = cons(a, append(x, y))
 FOR ELEM :
   a ELEM x <-> (EX y, z : x = append(y, cons(a, z)))
THEORYEND
```

Fig. 3. Freely Generated Data Types.

3.2 State Based Systems

In many applications it is most appropriate to describe a system in terms of

- a *state space*,
- a *transition relation*, and
- communications with the outside world (environment, other components).

Temporal Logic, as for example described in [22], is a formalism to reason about finite and infinite sequences of (global) states, sometimes called behaviours. The concurrent execution of components is modelled by *interleaving*. The global state space is divided into the local state spaces of components and partitions shared between certain components for communication. Temporal formulae are used to specify systems φ_{sys} as well as their properties φ_{prop}. To establish behavioural properties one has to show $\varphi_{sys} \rightarrow \varphi_{prop}$ while $\varphi_{sys_1} \rightarrow \varphi_{sys_2}$ expresses the fact that sys_1 is a refinement of sys_2.

The basic state-based specification technique used in VSE is based on Lamport's Temporal Logic of Actions [1, 20, 2], which restricts the form of temporal specifications to an abstract form of automata. The VSE realisation of Temporal Logic emphasises *modularity* with respect to

- the logical representation of complex systems,
- interactive proof generation, and
- the management of developments.

Based on the formalism of Temporal Logic, VSE supports a general notion *state transition systems* that allows the user to model various special instances of this general paradigm.

Elementary Specifications. A state is given by the content of the so-called *flexible* (state dependent) variables. An elementary specification names a finite set of flexible variables that are relevant for the local behaviour of a component (or a family of similar components). The possible values these variables can take are specified by abstract data types using the techniques described above. A subset of these variables is used for communication with the component's environment. In addition to the theory of compositional development presented in [2], which covers the composition of systems using input and output variables, *shared variables* are supported by VSE.

State transitions caused by the component's environment, which might not yet be formally specified, are taken into account by steps (actions of the form $(x_1 = x_1' \wedge \ldots \wedge x_n = x_n')$) that allow for arbitrary changes except that certain variables, given by the so-called *stuttering index*, are left untouched. Leaving open certain steps is also exploited for the refinement of systems.

The basic form of elementary specifications, also discussed in [2], then is $\exists x_1, \ldots, x_n.(\mathit{Init} \wedge \Box\, [\text{SYS-STEPS}]_{\bar{v}} \wedge \mathit{FAIR})$, where

- SYS-STEPS are the *actions* (steps) made by the system,
- $\bar{v}$ is the stuttering index, which contains flexible variables of the system,
- *Init* is a predicate which holds initially,

– $x_1, \ldots, x_n$ are the internal variables (hiding), and
– $FAIR$ stands for the fairness requirements of the system.

In addition to the normal form, the user can specify an elementary specification using a pseudo programming language which will be translated in subsequent steps into the normal form given above.

Structuring of Specifications. VSE provides two operators to structure state-based specifications: **combine** and **include**. We focus on the **combine**-operator which models the concurrent execution of two components given by the specifications S_1 and S_2.

Communication between components is by input-output variables or by shared variables. Concurrency is modelled by considering all possible *interleavings* of actions of the components involved. Thus, a behaviour $\bar{s}$ is a behaviour of the combined system if and only if it is an interleaved behaviour of both S_1 and S_2. Since in the local specifications we have already taken into account transitions caused by the outside world, specifications of the form $S_1 \wedge S_2$ are basically what we need. However, in the presence of shared variables interleaving can no longer be defined uniformly, i.e. without considering the particular actions of the system. In VSE, see [30, 16], it is made explicit for each step which component is *active*. To this end, a special predicate $active_{spec}$ is available. By using $active_{spec}$ it becomes possible to distinguish steps of a component from steps of the environment. Let $A_1, \ldots, A_n$ be the elementary actions of a component given by S. The action formula SYS-STEPS is of the form

$$((A_1 \vee \ldots \vee A_n) \wedge active_S) \vee (\neg active_S \wedge \bar{x} = \bar{x}' \wedge \bar{o} = \bar{o}') \ ,$$

where $\bar{x}$ and $\bar{o}$ are the internal and output variables of S. The second disjunct claims that environment steps do not affect the variables "owned" by S. For VSE-SL specifications in canonical form, the system builds this formula from the list of actions automatically.

Given two elementary systems S_1 and S_2 as above the combined system is given by a formula of the form

$$S_1 \wedge S_2 \wedge \Box\, (\neg active_{S_1} \vee \neg active_{S_2}) \ .$$

In addition, the specification of the combined system contains a (formal) mapping of the interfaces of the components.

Assumption Guarantee Specifications. A complex system needs not to be specified as a single monolytic block, but as a collection of independent components. The verification process supports the modular specification style as it allows the user to prove properties local to components using assumptions about the environment of the component.

To this end specifications of concurrent systems are not flattened. Instead, the component specifications and the specification of the combination are kept as separate units referring to the logical representations discussed above.

To prove theorems local to (deduction) units associated with single components requires in almost all cases *assumptions* about the behaviour of the environment. In VSE there are two ways to handle assumptions. They can be *marked* and *pruned* upon completion of the combined system following rules that are enforced by the management system discussed below. On a logical basis one can express by a formula of the form

(*System Guarantee*) **unless** ¬ (*Assumption about Environment*)

that a certain *guarantee* holds "as long as" some assumption is satisfied by the environment. Results of this kind can be combined with a general (derived) rule of VSE's temporal logic to yield a desired property of the overall system. This rule is located in the "combined node", the unit corresponding to the combined system. It has to be used to prune mutual depended assumptions. The "circularity rule"[10] is of the form:

$$(((active_{S_1} \rightarrow P_1) \textbf{ unless } \neg(\neg active_{S_1} \rightarrow \tilde{P}_2)) \ \wedge \tag{1}$$

$$((active_{S_2} \rightarrow S_2) \textbf{ unless } \neg(\neg active_{S_2} \rightarrow \tilde{P}_1)) \ \wedge \tag{2}$$

$$\Box\,((active_{S_1} \rightarrow P_1) \rightarrow (\neg active_{S_2} \rightarrow \tilde{P}_1)) \ \wedge \tag{3}$$

$$\Box\,((active_{S_2} \rightarrow P_2) \rightarrow (\neg active_{S_1} \rightarrow \tilde{P}_2))) \tag{4}$$

$$\rightarrow$$

$$\Box\,((\neg active_{S_1} \rightarrow \tilde{P}_2) \wedge \Box\,((\neg active_{S_2} \rightarrow \tilde{P}_1)). \tag{5}$$

P_1 and P_2 represent the guarantees of the given systems S_1 and S_2, and $\tilde{P}_1$ and $\tilde{P}_2$ denote the assumptions S_1 and S_2 make about the environment.

With the assumptions (that are discharged in the combined system) and with (1) and (2) we immediately obtain $\Box\,(active_{S_1} \rightarrow P_1)$ and $\Box\,(active_{S_2} \rightarrow P_2)$ local to the lemma bases of S_1 and S_2, respectively.

4 Deductive Support

An interactive theorem prover serves as the deductive unit that has been implemented in VSE-II. VSE-II specifications written in VSE-SL (VSE Specification Language) are automatically translated into logic formalism and proof obligations are automatically generated.

There are two kinds of proof obligations that have to be checked, namely those that arise from the properties the system under consideration has to fulfill and those that represent the claim that a system is in fact a refinement of another given system. In order to tackle these proof obligations there is a need for a structured deduction that decomposes large proof obligations into simpler tasks and synthesises an overall proof from the derived partial solutions. Such a structured deduction not only simplifies the speculation of lemmata but also enables the user to utilise methods that are somewhat polished to solve the specific problems that arise.

[10] This is the special case for two components.

Moreover, for specialised logics as, for instance, Dynamic Logic [10] for algorithmic program constructs or TLA [20] for reactive and concurrent systems, there exist appropriate proof strategies especially tailored for formal specifications and proof problems arising thereof.

In VSE-II the knowledge how specific proof situations are to be tackled is encoded in a bundle of individual tactics. The accumulation of various such tactics imposes an emerging functionality which is able to prove complex theorems in a goal directed way. All these tactics operate on a common representation of the actual proof state that is reflected in a proof tree annotated by additional tactical knowledge. This proof tree is visible to the user and thus allows him to give strategic advice following the *direct manipulation paradigm*. Tactics may prune or refine this proof tree and represent the *algorithmic* part of the proof search. In VSE-II proof decisions can be withdrawn by chronological backtracking as well as by pruning arbitrary branches of the proof tree. The approach combines a high degree of automation with an elaborate interactive proof engineering environment.

In addition to tactics that provide a more general frame to tackle certain proof situations, the VSE-II system also knows about certain heuristics. Usually such heuristics are rather global and handle specification independent proof situations. The need for specification/theory dependent heuristics did arise during the VALIKRYPT (Verification and Validation of Cryptographic Protocols) project while applying the Paulson approach [27] for the verification of protocols. These theory dependent heuristics extend the global heuristics by the possibility to speak directly about the elements of user defined theories, as for example functions or predicates. This makes possible a very high degree of automation in specialised areas as, for example, in the protocol verification field.

Often the attempt to prove a lemma or a proof obligation fails and errors thus detected in the specification or implementation have to be corrected. In VSE-II an elaborate correctness management supervises the application of corrections and invalidates only those proofs which really are affected by the modifications.

4.1 Architecture

The general picture of the methodology VSE-II is based on can be found in Figure 4. As a tool for formal software development VSE-II consists of

- a basic system for editing and type checking specifications and implementations written in the specification language VSE-SL.
- a facility to display the development structure,
- a theorem prover for treating the proof obligations arising from development steps,
- a central database to store all aspects of the development, including proofs, and
- an automatic management of dependencies between development steps.

Compared to VSE-I [14, 15], which was based on a simple, non-compositional approach for state based systems, VSE-II [16, 17] is extended by comprehensive

methods that are especially tailored for *distributed* and *concurrent systems* [29]. Also, it has an even more efficient and uniform proof support which utilises the implicit structure of the arising proof obligations. The basic logic formalism used in VSE-II is close to TLA (Temporal Logic of Actions) [20], where a refined *correctness management* allows for an evolutionary software development.

VSE-II is based on a methodology to use the structure of a given specification (e.g. parameterisation, actualisation, enrichment, or modules) to distribute the deductive reasoning into local theories [30]. Each theory is considered as an encapsulated unit, with its own local signature and axioms. Relations between different theories, as they are given by the model-theoretic structure of the specification, are represented by different links between theories. Each theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulae included from linked theories.

This method of a structured specification *and* verification is reflected in the central data structure of a *development graph* (see Figure 5), the nodes of which correspond to the units mentioned above. It also provides a graphical interface for the system under development. Different types of specifications are displayed as different types of nodes, e.g. abstract data types as hexagons, while the relations between the nodes are displayed as links in the development graph.

The development graph shown in Figure 5 represents parts of the consistency proof of a trusted digital signature device.

4.2 Structured Deduction

Structuring specifications as described above supports readability and makes it easier to edit specifications by allowing the user to use local notions. However,

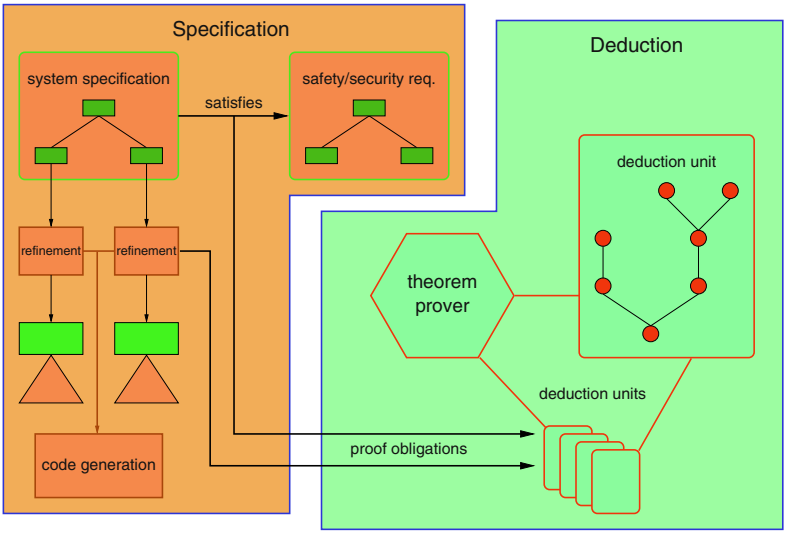


Fig. 4. VSE-II Methodology.

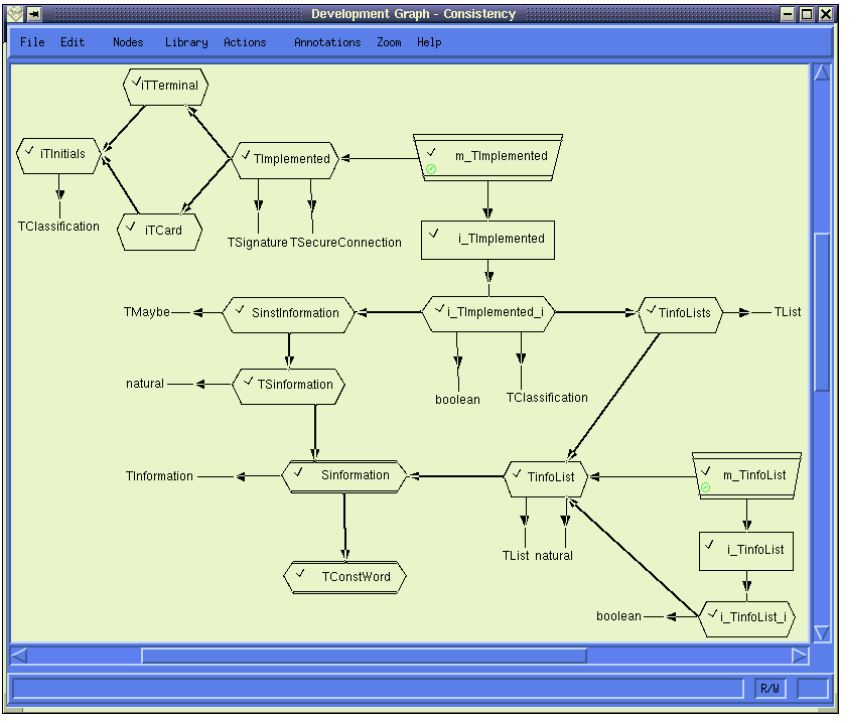


Fig. 5. Development Graph of Consistency Proof of a Digital Signature Device.

the system exploits structure beyond this purely syntactical level. Components of a combined system can be viewed as systems in their own right where certain parts can be observed from outside while most of the inner structure, including the flow of control and local program variables are hidden.

In particular, this allows the user to prove properties of a combined system in a modular way. This means that local *lemma bases* can be attached to components where local proofs are conducted and stored. Exchange of information between lemma bases is on demand. This approach has two main advantages: First, the given structure of the specification is used to reduce the search space as large parts of the overall system are not visible and, second, the *revision process* is supported by storing the proofs local to certain lemma bases, thus making the export and import of information (between lemma bases) explicit.

Management of Assumptions and Lemmata. As mentioned above, lemma bases are associated with each specification (elementary or combined). A lemma base consists of a set of formulas together with (optionally) proofs, where each such formula is marked as *axiom*, *assumption*, *proof obligation*, or *auxiliary*. While working on a proof, all lemmata of sub-specifications may be used.

The lemma bases are organised into levels. The local specification of a component forms the lowest level (marked as *axiom*). In a proof on level i, all lemmas from levels $< i$ can be used, and lemmas from level i can be used if no circular

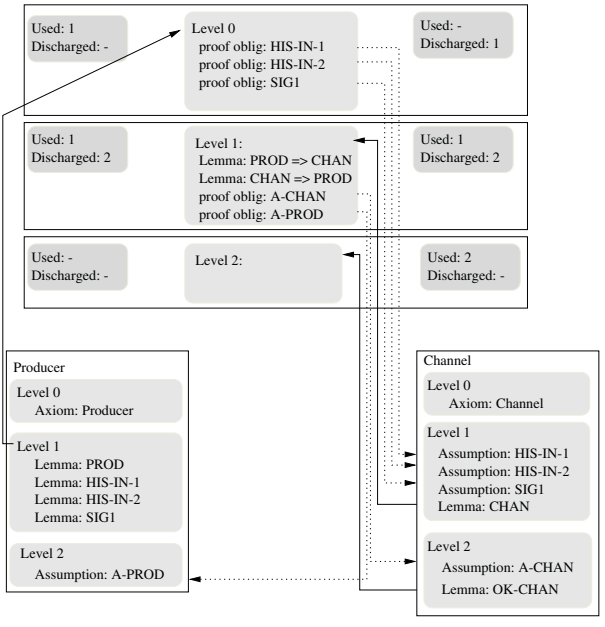


Fig. 6. Example lemma base.

dependency is caused on level i. This is sufficient to organise the reasoning process for a single specification. To discharge assumptions of a combined system, without the need for a circularity check involving all combined systems, an additional mechanism is necessary.

For each level of the lemma base of a **combine** node and for each component, the highest level from which lemmas were imported is recorded. Likewise, the smallest level on which assumptions are discharged is recorded, too. If for each level the first number is smaller than the second, the use of lemmas and assumptions does not lead to circular reasoning. When an assumption is discharged, it is added to the **combine** node as a proof obligation.

The lemma bases for the safety part of the proofs from the previous section are shown in Figure 6 (the consumer is omitted).

Note that "exporting" CHAN-OK to level 0 in the **combine**-node would lead to a cycle, so we need a new level 1 on which CHAN-OK can be used without jeopardising non-circularity.

5 The Future of VSE

Many of the systems supporting the formal development of software operate with interactive theorem provers as for example VSE[14, 13, 16] or the B-Tool [3, 6, 8]. Experiences show that most of the problems arising in the verification of a system require induction. This fact indicates that theorems or properties required for industrial sized system specifications cannot be proved fully automatically in general.

Nevertheless, automatic methods have shown their applicability in many case studies. These technologies try to compute all the states the specified system can reach from a given initial state with the help of some fixpoint computation. Whenever this succeeds – i. e. this computation terminates – they can answer questions concerning single or all of the possibly reachable states. This technology is called *model checking*. Usually, model checking strategies are limited to finite state problems and in this case are guaranteed to terminate. In Hybrid Systems, which are in general not finite state, the fix-point computation is no longer guaranteed to terminate. Nevertheless, it is always worth a try. For problems where the procedure does not succeed, techniques like abstraction, as for example presented in [4], have to be applied. This, however, requires an interactive proof system to show the correctness of the abstraction.

Although most of today's tools supporting the verification of large scaled systems favour either theorem proving or model checking, the two methodologies do not exclude each other at all. It thus seems reasonable to integrate them in a single formal specification and verification tool, and that in a way such that results of the interactive side can be used on the automatic side and vice versa.

Such an integration (or cooperation) is planned to become one of the next extensions of VSE-II. However, rather than inventing new logics, calculi or methodologies it is intended to utilise tools that are commonly available. A first promising result in this direction has already been found in the integration of Hybrid Automata[11] model checking and VSE-II. Verification tools for Hybrid Automata are usually based on infinite-state model checking. The integration into VSE-II is such that suitable sub-problems that are to be proved with VSE-II get transformed into Hybrid System verification problems that can be model checked. If successful, the resulting solution can be fed back into VSE-II. Hence, lengthy and somewhat boring parts of the proof can be delegated to an automatic procedure.

The integration of various decision systems is also planned for the near future. As an example consider linear arithmetics. Currently, (natural) numbers belong to the basic data types within VSE-II. However, they are represented by their constructors 0 and *succ*. Although sufficient in principle, there are some obvious caveats when dealing with larger numbers or even simple arithmetic calculations. Therefore, additional modules are to be integrated that allow the user to solve satisfiability problems of linear arithmetic constraints automatically, e. g. with the help of Fourier's elimination of number variables.

But this is just a starting point for integrating even further decision procedures. Today, a large amount of decision procedures is known for various abstract data types, most of them based on Nelson/Oppen's or Shostak's work. The integration of these more general methods will further reduce the software engineer's effort in verifying large scaled systems. And so, the software engineer is rather concerned with the creative work of the overall verification process and gets relieved of comparatively uninteresting technical details.

[11] Hybrid Automata are especially useful for the specification and verification of real-time systems.

References

1. Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
2. Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.
3. J-R. Abrial. The B tool. In G. Goos and J. Hartmanis, editors, *VDM – The Way Ahead. Proc. 2nd VDM-Europe Symposium*, volume 328 of *Lecture Notes in Computer Science*, pages 86–87. VDM-Europe, Springer-Verlag, 1988.
4. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
5. Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors. *Algebraic foundations of systems specification*. IFIP state-of-the-art reports. Springer, Berlin, 1999.
6. B Core UK Ltd. *B-Tool manual*, 1994.
7. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of uppaal. In *International Workshop on Software Tools for Technology Transfer, Aalborg, Denmark*, 1998.
8. D. Bert. B'98: Recent advances in the development and use of the b method, April 1998.
9. Rainer Drexler, Wolfgang Reif, Gerhard Schellhorn, Kurt Stenzel, Werner Stephan, and Andreas Wolpers. The KIV system: A tool for formal program development. volume 665 of *Lecture notes in computer science*, Berlin, 1993. Springer.
10. R. Goldblatt. Axiomising the logic of computer programming. volume 130 of *LNCS*. Springer Verlag, Berlin, 1982.
11. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.
12. T. A. Henzinger and P.-H. Ho. HyTech: The cornell hybrid technology tool. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, pages 265–293. Springer Verlag, Lecture Notes in Computer Science, vol. 999, 1995.
13. D. Hutter, B. Langenstein, C. Sengler, J. Siekmann, W. Stephan, , and A. Wolpers. Verification support environment (vse). In *Journal of High Integrity Systems*, 1995.
14. Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Deduction in the Verification Support Environment (VSE). In Marie-Claude Gaudel and James Woodcock, editors, *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER, 1996.
15. Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Verification support environment (vse). *High Integrity Systems*, 1(6):523–530, 1996.
16. Dieter Hutter, Heiko Mantel, Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. VSE: Controlling the Complexity in Formal Software Development. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, pages 351–358, Boppard, Germany, 1999. Springer-Verlag, LNCS 1641.
17. Dieter Hutter, Georg Rock, Jörg H. Siekmann, Werner Stephan, and Roland Vogt. Formal Software Development in the Verification Support Environment (VSE). In Bill Manaris Jim Etheredge, editor, *FLAIRS-2000: Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*, pages 367–376. AAAI-Press, 2000.

18. Dieter Hutter and Claus Sengler. INKA: The Next Generation. In *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Computer Science*, New Brunswick, N.Y., 1996. Springer.
19. C.B. Jones. *Systematic Software Development Using VDM.* Prentice Hall, 2nd edition, 1990.
20. Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), 1994.
21. Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of Abstract Data Types.* Teubner, Chichester;New York;Brisbane, 1996.
22. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer Verlag, New York, 1992.
23. W. McCune. Mace 2.0 reference manual and guide. Technical Memo ANL/MCS-TM-249, Argonne National Laboratory, June 2001.
24. K. McMillan. The smv model checker. http://www-cad.eecs.berkeley.edu/∼kenmcmil/smv/.
25. Sam Owre, John M. Rushby, and Natarajan Shankar. Pvs: a prototype verification system. In Deepak Kapur, editor, *Proceedings of the CADE-11*, volume 607 of *LNAI*, pages 748–752. Springer, 1992.
26. Lawrence C. Paulson. *ISABELLE, A Generic Theorem Prover*, volume 828 of *LLNCS.* Springer Verlag, 1994.
27. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
28. W. Reif. Correctness of generic modules. In Nerode and Taitslin, editors, *Symposium on Logical Foundations of Computer Science*, volume 620 of *LNCS.* Springer, 1992.
29. Georg Rock, Werner Stephan, and Andreas Wolpers. Tool support for the compositional development of distributed systems. In *Tagungsband 7. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, number 315 in GMD Studien. GMD, 1997.
30. Georg Rock, Werner Stephan, and Andreas Wolpers. Modular Reasoning about Structured TLA Specifications. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, pages 217–229. Springer, WienNewYork, 1999.
31. Ian Sommerville. *Software Engineering.* International Computer Sciences Series. Addison-Wesley, Harlow, UK, 5th edition, 1995.
32. J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, 2nd edition, 1992.
33. S. Yovine. Kronos: A verification tool for real-time systems. In *Software Tools for Technology Transfer*, volume 1, pages 123–133, 1997.

SAT-Based Cooperative Planning: A Proposal

Marco Benedetti and Luigia Carlucci Aiello

Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza", Italy
{benedetti,aiello}@dis.uniroma1.it

Abstract. We present a work-in-progress on distributed planning, which relies on the "planning as satisfiability" paradigm. It allows for multi-agent cooperative planning by joining SAT-based planning and a particular approach to distributed propositional satisfiability. Each agent is thus enabled to plan on its own and communicate with other agents during the planning process, in such a way that synchronized and possibly cooperative plans come out as a result. We discuss in some details both piers of our construction: SAT-based planning techniques and distributed approaches to satisfiability. Then, we propose how to join them by presenting a working example.

1 Introduction

Planning is a research area in artificial intelligence aiming at the construction of algorithms – called planners – that enable an agent (a robot or a "softbot") to synthesize a course of actions that will achieve its goals. Planning has been studied since the early days of AI; recently the interest has been renewed and results abound, with proposals of systems capable to deal with real life applications. We cannot go into many details here, neither illustrate the vast literature in the field; a comprehensive presentation of the state of the art can be found – for instance – in the survey paper by Weld [23].

Much research has recently been done in multi-agent systems and in robot teams, i.e., scenarios where several software or embodied systems cooperate at the solution of a given goal. In this case, planning becomes "cooperative planning".

Cooperative planning has various meanings in the current AI literature. All of them are geared around the idea that each agent – participating in the cooperative achievement of a common goal – has to take into account, in its planning activity, that other agents are present. Both the presence and the actions of the various agents affect the scenario. In other words, things not only change because of the effects of the actions of an individual agent, but also because of the effects of the actions of its companions. In some cases, planners take also into account the possibility of exogenous actions; of course, in a multi-agent scenario, actions of the companion agents are not to be considered exogenous. This view is exemplified – for instance – in [12].

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 494–513, 2005.

A distinction between "cooperative distributed planning" (CDP) and "negotiated distributed planning" (NDP) is made by desJardins et al. in [11]. In their words:

> Because it places the problem of forming a competent (sometimes even optimal) plan as the ultimate objective, CDP is typically carried out by agents that have been endowed with shared obectives and representations by a single designer or team of designers. Although in some cases the purpose of the agents is to form a central plan, more generally the purpose is that the distributed parts of the developing plan will jointly execute in a coherent and effective manner. Thus, in CDP, agents typically exchange information about their plans, which they iteratively refine and revise until they fit together well.

In NDP instead, each agent engaged in the planning activity negotiates with the others, with the goal not to form a good collective plan, but to form an individual plan that, when executed in the global context, meets the agent's own local objectives.

Although CDP involves coomunication and coooperation during the planning activity, the solutions proposed in the literature share a common feature: each agent in the team acts, in some sense, in isolation. It receives (from some central authority) a (sub)goal to tackle, and formulates its own plan to solve its own (sub)goal. Communication and coordination with the other agents in the team has the only purpose of avoiding conflicts at execution time.

As a prototipical architecture that exemplifies the above view about cooperative planning, we refer to the three level architecture proposed by Botelho and Alami in [6]. They propose three distinct levels for: [1] mission decomposition[1], [2] task allocation, and [3] task achievement. Level 1 is not distributed. The distribution happens at level 2, where the task allocation is done (by some centralized intelligence) to the various agents in the team. At level 3, each agent carries on its part of the share, to achieve the common goal.

Botelho and Alami in [6] say:

> Mission decomposition is a purely deliberative task. It is at this level that there are less needs of context dependent information. It can be done in a central way. It is essentially a one thread process.
> Of course it can benefit from several CPUs but this is a distribution of computing load, which is different in nature from problems calling for cooperative decision-making based on independent goals, on various robot capabilities and contexts.

Hence they resort to a central high level planner, and exploit coordination and cooperation only at level 3, where cooperation essentially consists in opportunistic action reallocation, and detection and suppression of redundancy.

Conversely, our idea is that, in order to fully exploit the presence of several agents in the scenario, hence to reach a true cooperation, agents should cooperate

[1] In the robotic literature high level goals are more often called "missions," and "mission planning" consists in the decomposition of a mission into tasks.

also in the plan formation activity, i.e., at level 1 of the above hierarchy. If cooperation means working together to reach a common goal, cooperation should start at the planning level, i.e., working together to *jointly build* a common plan for a common goal. We believe that there are several advantages in this, the increase of efficiency achieved by distributing the computing load among several CPUs being only one of them, certainly not the most prominent one.

A remarkable advantage comes from the fact that agents have generally to plan with an incomplete knowledge of the world and of others' goals. This may result in an advantage, rather than in an additional problem as it is generally believed to be. Different agents in fact may have different viewpoints on the world, which they may share, thus enriching the overall knowledge, while staying able to only reason on what they care about. We may call this phenomenon "collective knowledge for collective planning". In addition, we may also face situations where each single agent – if left alone – has not enough information to solve a problem, but, by sharing individual viewpoints and/or conclusions, the team of agents can come to a solution. This phenomenon is very well exemplified by the three wisemen problem (proposed long ago by John McCarthy), which for many years has been a sort of benchmark for knowledge representation formalisms (see, e.g. the solution proposed in [7, 8]). So, inter agent communication during the plan formation phase has many advantages, as the variuos agents can synchronize their planning efforts. In addition, they can communicate both the failures and successes of their attempts to other agents, hence saving them from going through the same reasoning. This aspect is very interesting: not only an agent can learn by its own experience, but it can also learn "by being told". In the plausible case of agents aware of their capabilities, a further advantage stems from the fact that each agent may bid for working on the part of the plan relative to the parts of the world he knows better, or where he can work better.

After several efforts to develop specialized planning languages, in the assumption that logic was not a suitable language for representing and solving planning problems, mainly for complexity reasons, the recent years have seen several proposals where logic is used to represent planning problems. Then, either deduction or model checking are used as the tool for plan generation. A noteworthy proposal, made by Kautz and Selman in [16], views plan generation as a SATisfiability problem in proposiational logic. Many proposals have followed the original one, with variations on the theme, that have brought the efficiency of SAT based planners to face industrial applications.

The work we present here aims at making it feasible a real distributed planning process, which can in principle exploit some of the advantages of distributed planning we just pointed out. Our proposal is to distribute plan generation by distributing the underlying SAT-solving procedure. To this end however, we cannot rely on a distribution strategy for SAT whose only aim is to increase efficiency. Conversely, we need a distribution strategy that on the one side reflects the goal/subgoal distribution that occurs in the initial phase, when each agent selects which part of the plan to work on, and on the other side allows for an effective communication of partial results, during plan generation.

1.1 Organization of the Paper

We first illustrate the SAT problem, and shortly introduce the major techniques to attack it (see Section 2). We then briefly present motivations and techniques to distribute and/or parallelize the task of solving a SAT instance (see Section 3). In Section 4 we move to a short description of the planning problem in AI, mainly to introduce the relevant nomenclature. Section 5 is then devoted to illustrate planning as a satisfiability problem. Section 6 contains a description of how to distribute SAT in a way suitable for planning applications. Cooperative distributed planning is proposed in Section 7 with the help of a working example. Section 8 is devoted to a many-sided discussion concerning what is still to be worked out in our proposal and which directions we are about to follow. Few concluding remarks close the paper (see Section 9).

2 SAT

The SAT problem (SAT, for short) consists of deciding whether a propositional formula has (at least) one satisfying assignment (equivalently, it is the problem of deciding whether a model for a propositional theory exists). In case models exist, the formula is said to be *satisfiable*, while it is *unsatisfiable* (or *contradictory* or *inconsistent*) in the opposite case. So, SAT is a *decision* problem, i.e.: a function from the set of all possible propositional formulas to the set $\{sat, unsat\}$. For satisfiable formulas, it is often useful to obtain a particular model (truth assignment to the variables in the formula) and not only to know that one exists. In this case a *search* problem is to be solved.

SAT is one of the most prominent problems in logic, computer science and artificial intelligence, and it is of both theoretical and practical interest. From a theoretical point of view, the time and space complexity of SAT have been investigated in details for the general case and for particular sub-classes of the problem. From the practical point of view, new or refined algorithms to face SAT are continuously proposed in the literature.

SAT belongs to the class of NP-complete problems (any other NP-complete problem is reducible to SAT in polynomial time [9]) whose algorithmic solutions are currently believed to have exponential worst case [21]. Actually, it was the prototypical NP-complete problem and it was also the first problem whose NP-completeness was proved [9, 20]. So, there is no polynomial time algorithm which solves SAT, provided $P \neq NP$: for every satisfiability algorithm there exist classes of formulas which require superpolynomial run time. A strictly related problem is the one of deciding unsatisfiability, called UNSAT (UNSAT is in the class coNP). What researchers do within the boundaries of these theoretical limitations is to look for algorithms with good *average* performance (possibly only good on some specific sub-classes of propositional formulas).

Despite being a very old problem in logic and artificial intelligence, SAT has attracted an increasing interest in the last decade. This is witnessed by the growing number of SAT-related papers recently published in journals and conference proceedings, as well as by the many proposals of new algorithms for SAT or refined variants of known ones. Web sites entirely devoted to SAT also

appeared recently [14, 3]. They collect algorithms, instances, papers and even propose online comparisons among different algorithms on several benchmarks.

This new spread of algorithms and proposals for SAT is mainly due to a practical interest in solving real world problems with *SAT-encoding* techniques.

Two main classes of algorithms for SAT exist: *complete algorithms* (also called *deterministic* or *systematic*), are those claiming that a formula is (un)satisfiable if and only if it is (un)satisfiable; *incomplete algorithms* (also called *local* or *randomized* or *stochastic*), are those performing incomplete search: if a model is not found after some specific resource limits have been exceeded, these algorithms terminate failing to prove both satisfiability and unsatisfiability. Even though incomplete algorithms have been sometimes employed in SAT based planning, we here devote our attention to complete algorithms only.

Complete algorithms perform an exhaustive search in some state space. Two subclasses of these algorithms are to be considered, depending on the aim of the search.

Direct model search algorithms are those explicitly searching the space of possible partial truth assignment, so they aim at finding a model. The DPLL algorithm [10] (and subsequent variants) is a well known example falling in this category.

Refutation procedures are those aiming at proving unsatisfiability. In case a complete refutation procedure fails to prove inconsistency, the formula is guaranteed to be satisfiable, and in some cases a model can be extracted as well. These methods are based on *resolution*.

Direct model search algorithms gained the widest popularity in the SAT community, even though some features of refutation-based approaches still deserve great attention. For certain classes of applications – not by chance the one we propose below is among these – algorithms that work by resolution are best suited.

3 Distributing SAT

Distributing satisfiability means to consider more than one *process* (or *agent*, within the agent-oriented perspective we adopt) and give each one a share of a SAT problem. The question is why and how that distribution should be done.

Propositional satisfiability is a particular case of automated deduction, a field where parallel theorem proving strategies have been investigated in the literature. An excellent survey on this subject is [5], and we here refer to the principles and taxonomy reported there.

A first important concept is the one of *parallelism at the search level*, i.e.: the use of multiple deductive processes that work at the same problem in parallel. This strategy is adopted – for example – when the search space for the problem of finding a satisfying assignment is *decomposed* (see for example [4]). As an example, given a formula $\mathcal{F}$, one arbitrarily chooses a variable x occurring in $\mathcal{F}$, then solves separately and in parallel the SAT problem on $\mathcal{F}$ with x assigned to

true, and the one on $\mathcal{F}$ with x assigned to *false*. The search space is thus partitioned into two not overlapping regions and no communication is needed among processes, other than the eventual "*my portion does (not) contain a solution*".

The set of agents involved in one such distributed algorithm, can contain either *homogeneous* or *heterogeneous* entities, depending on whether or not each agent is provided with exactly the same inference system. In addition to this, a *peer agents* scheme can be adopted as opposite to a *master-slave* implementation: in the former case, each agent plays the same role as the others, and has no special task with respect to the SAT problem under consideration; in the latter case, a hierarchy among agents does exist, and some of them have special roles with respect to coordination, distribution and decision.

Distributed search generally aims at speeding-up the process of finding a solution. However, not only could one desire to speed up search, but he also could aim at producing a particular application, as in our case.

Many approaches to distributed satisfiability aiming at speeding up the search exist in the literature. Just to cite a few: `PSATO` [24], which is a distributed-search master-slave implementation of the Davis-Putnam algorithm; `Parallel MODOC` [22], a parallel version of a model-elimination based algorithm which exploits autarkies; `Parallel Satz` [15], relying on a master/slave model for communications and on a dynamic preemptive workload balancing by work stealing.

4 Planning

We here limit ourselves to present the definition of a planning problem and to give enough background on the approach we take.

A *planning problem* is a representation, in some formal language, of the following three aspects:

1. a description of the initial state of the world;
2. a description of the goal state the agent has to reach;
3. a description of the possible actions that can be performed by the agent. This is often called *domain theory* or *action theory*.

The solution of the problem (if one exists) is a *plan*, i.e., a sequence of actions that, when executed in any world satisfying the initial state description, will achieve the goal. A more general solution to a planning problem is a set of *partially ordered* actions. When this representation is used, not all couples of actions are ordered (as it happens in a sequence of actions), but only those actually requiring to be serialized in some particular order to maintain the correctness of the plan. Consequently, a direct acyclic graph of actions is a well suited representation for the solution of a planning problem.

In addition to the description of a planning problem, a planner has to be provided with a *background theory*, i.e., a formal description of general known properties of the world.

The planning problem naturally extends to a multi-agent framework. In this case, the basic formalization steps given above are still valid, but several choices are possible on whether or not agents explicitly cooperate, on the way of

representing plans of other agents, on whether or not the planning phase itself is distributed, on the time instants and the reasons why agents communicate to synchronize their courses of action, and so on. Whichever the case, a multi-agent plan is still a partially ordered set of actions, but additional mechanisms to have the agents cooperating in producing and executing it are needed.

To conclude this brief introduction, it is worth saying that in classical planning problems, a number of simplifying assumptions are made (our proposal is a classical one, as it maintains many of these limitations). Some of them are absolutely not trivial to be removed: atomic time (or, equivalently, instantaneous duration of action execution), no exogenous events, deterministic action effects, etc., and are presently topics of active research. Last but not least, the problem of distributing the planning activity in multi-agent domains, despite absent in classical planning, is very relevant to applications and theoretically quite interesting: it is exactly the kind of problem we address in what follows, by opening the perspective of a SAT-based distributed mechanism.

5 Planning in SAT

A SAT-encoding technique is a well-known procedure in three steps: (1) a way to encode a problem from a domain of interest into a propositional formula is defined; if the original problem is a decision problem, the encoding is made in such a way that the corresponding SAT decision problem answers the original question; in general, for search problems that require to extract a particular solution (and not just a SAT/UNSAT answer), the encoding is made in such a way that a simple procedure exists which generates a solution to the original problem, provided a model for the encoded formula is given; (2) the formula is solved (and a model is possibly extracted for satisfiable formulas); (3) the answer to the encoded SAT problem is used to answer the original problem, possibly by means of a procedure which constructs a solution for the original problem given a model for the formula.

As an example of encoding technique, we consider the *planning* problem [17, 18, 13, 19] (see Figure 1). In this case a *compiler* takes as input a planning problem (which consists of the action descriptions and the initial and goal state) guesses a solution length (number of actions in the solution plan) and produces

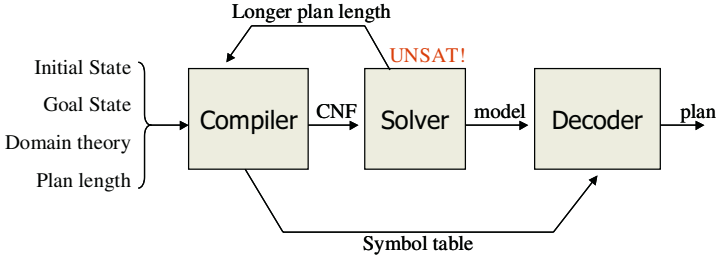


Fig. 1. Planning as satisfiability.

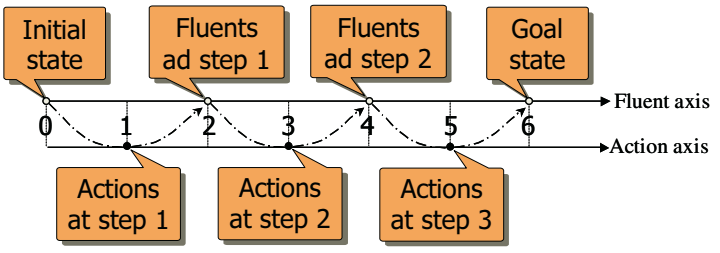


Fig. 2. Time encoding.

a propositional formula and a *symbol table* as output. The propositional formula encodes all possible plans of the considered length in such a way that a solution of that length exists if and only if the formula is satisfiable. The symbol table is used to record correspondences between propositional variables and the planning instance. If a model for the formula is found, a *decoder* translates that model into a plan by using the symbol table. If the formula is found to be unsatisfiable, the compiler generates a new encoding reflecting a longer plan length.

Several different ways for compiling planning instances into SAT instances exist. We here briefly present a simple version of the compiler in Figure 1, with the aim of just giving the necessary tools to further proceed in the exposition.

A first question is how to represent time. This is usually done by means of non-negative integer values. Values for *state fluents* are given only in even time instant, while *actions* only occur at odd time instants. According to this representation, the truth value of the fluent P after i steps is denoted by $P_{2i}(x_1, x_2, ..., x_n)$, while an action A executed at step i is denoted by $A_{2i+1}(x_1, x_2, \ldots, x_n)$. In Figure 2 this kind of time encoding is represented for a 3-steps plan.

An important point is that terms such as $P_{2i}(x_1, x_2, \ldots, x_n)$ and $A_{2i+1}(x_1, x_2, \ldots, x_n)$ are encoded as single propositional variables. For example, $P_{2i}(x_1, x_2, \ldots, x_n)$ might become variable x_{123}. The correspondence between x_{123} and $P_{2i}(x_1, x_2, \ldots, x_n)$ is exactly what is recorded in the symbol table of Figure 1. When a model for the encoded formula is extracted, the truth value of the fluent P at time $2i$ on arguments $\langle x_1, x_2, ..., x_n \rangle$ is known to be – via the symbol table – the truth value of the variable x_{123}.

In the same way, if action $A_{2i+1}(x_1, x_2, ..., x_n)$ has been encoded – say – into variable x_{98} and if (and only if) the satisfying assignment assigns *true* to x_{98}, the action A has to be executed at step i with arguments $\langle x_1, x_2, ..., x_n \rangle$, according to the resulting plan.

The way initial and goal states are represented immediately follows from the choice for time encoding.

Initial State. It is written as a set of unit clauses $\{P_0^j(\mathbf{x})\}$ asserting that some facts *are true* at time 0. These facts represent the initial situation of the world.

Goal State. It is written as a set of unit clauses $\{P_{2n+2}^j(\mathbf{x})\}$ representing facts which *have to be true* in the final state of a n-step plan.

Apart from initial and goal states, the core of the SAT encoded instance is represented by the domain theory. There are several ways to perform this encoding. Here we present a very simple technique.

Action Definition. It is encoded by means of a set of clauses joining action occurrences at time $2i+1$ with their effects at time $2i+2$, and their preconditions at time $2i$. In particular, for each odd time instant, for each action A in the domain, and for each ground version of A (a ground version being an instance of the action with fully instantiated arguments), a set of binary clauses is introduced, meaning that action $A_{2i+1}(\mathbf{x})$ performed at step i needs preconditions $\{P^j_{2i}(\mathbf{y}_j)\}$ and has effects $\{E^j_{2i+2}(\mathbf{z}_j)\}$:

$$A_{2i+1}(\mathbf{x}) \rightarrow P^1_{2i}(\mathbf{y}_1), P^2_{2i}(\mathbf{y}_2), ..., P^n_{2i}(\mathbf{y}_n), E^1_{2i+2}(\mathbf{z}_1), E^2_{2i+2}(\mathbf{z}_2), ..., E^m_{2i+2}(\mathbf{z}_m)$$

Frame Axioms. They are used to specify which fluents are not affected by moving from an even time instant to the following one. In particular, for each fluent P^j and for each even time instant $2i$, a n-ary clause

$$P^j_{2i}(\mathbf{y}) \wedge \neg P^j_{2i+2}(\mathbf{y}) \rightarrow A^{1j}_{2i+1}(\mathbf{x}) \vee A^{2j}_{2i+1}(\mathbf{x}) \vee ... \vee A^{mj}_{2i+1}(\mathbf{x})$$

encodes (all the) reasons for P to possibly become false. Contrapositively, each such clause represents the set $\{A^{kj}\}$ of actions whose non-occurrence at time $2i+1$ guarantees that $P^j_{2i}(\mathbf{y}) \rightarrow P^j_{2i+2}(\mathbf{y})$.

Conflict Exclusion Clauses. According to the adopted representation, more than an action can simultaneously occur in each odd time instant. This feature allows for flexible encoding of plans, but must be carefully managed. In fact, it is necessary to explicitly ensure that all the linearized versions of the resulting plan are executable. Linearizability is threatened when a couple of actions exists, in the same time instant, such that one's preconditions are inconsistent with the other's effects. Action definition and frame axioms alone do not generally suffice to avoid such conflicts, but they can be excluded by adding a binary clause $\neg P^j_{2i}(\mathbf{x}) \vee \neg P^k_{2i}(\mathbf{x})$ for each couple of conflicting clauses in each odd time instant.

We exemplify the above mechanism on the classic *blocks world* with fluents $on_{2i}(x, y)$ (block x is on block y at step i), and $free_{2i}(x)$ (that no block is situated upon block x at step i). The action schemata we adopt are $move_{2i+1}(x, y)$ to represent the action of moving block x onto block y at step i and $moveOnTable_{2i+1}(x)$ with similar meaning when block x is moved onto the table. Constants used to represent objects in the domain are *table* and a distinct lowercase roman letter for each block.

Let us consider the planning instance in Figure 3. To encode the domain theory into a propositional representation of all the plans of length n on this instance, the following clauses have to be included:

– Action definition clauses; for all $x, y \in \{a, b, c, d\}$, $x \neq y$, $z \in \{a, b, c, d, table\}$, and $i \in \{0, 1, ..., n-1\}$, six clauses are written: $move_{2i+1}(x, y) \rightarrow free_{2i}(x)$, $free_{2i}(y), on_{2i}(x,z), \neg free_{2i+2}(y), free_{2i+2}(z), on_{2i+2}(x,y)$. In a similar way,

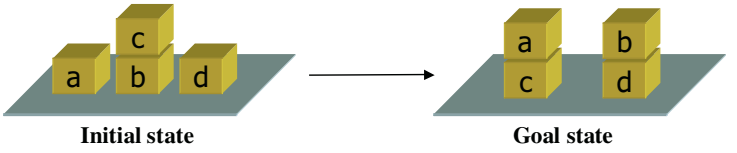


Fig. 3. A sample planning instance in the blocks world.

the four clauses $moveOnTable_{2i+1}(x) \rightarrow free_{2i}(x), on_{2i}(x,z), free_{2i+2}(z), on_{2i+2}(x, table)$ are introduced with respect to each fully instantiated action of type *moveOnTable* in each odd time instant.

- Frame axioms (one for each fluent); in particular, for all $x, y, z \in \{a, b, c, d\}$ and for all $i \in \{0, 1, ..., n-1\}$, the following clauses are written:

$$on_{2i}(x,y) \wedge \neg on_{2i+2}(x,y) \rightarrow move_{2i+1}(x,z) \vee moveOnTable_{2i+1}(x)$$
$$free_{2i}(x) \wedge \neg free_{2i+2}(x) \rightarrow move_{2i+1}(a,x) \vee ... \vee move_{2i+1}(d,x)$$

- Conflict exclusion clauses are written to avoid that the same block is moved twice at one single time instant. Action definition clauses do not suffice to ensure that the same block is not moved to two different locations at one time, as nowhere does that axiomatization specify that the position of a block is a function, not a general relation. Thus, we have to enforce that each block can have at most one underlying block at every time step, and this is done by avoiding at each action step two actions moving the same block. In particular: $\neg move_{2i+1}(x,y) \vee \neg move_{2i+1}(x,z)$ for all $x, y, z \in \{a, b, c, d\}$ and for all $i \in \{0, 1, ..., n-1\}$. Analogous clauses have to be written for avoiding the same block to be on the table and not on the table at the same time.

The resulting propositional formula encodes both the domain theory and the planning instance in such a way that a plan of length n exists if and only if the formula is satisfiable. In particular, the mechanism presented in Figure 1 first constructs the encoding with $n = 1$ and checks that it is unsatisfiable (no plan of length 1 achieves the goals), then it generates the encoding with $n = 2$ and this comes out to be satisfiable. Figure 4 shows all (and only) the variable assigned to true in this models. They are organized in a graphical representation according to their meaning as extracted from the symbol table. This way, a plan that reaches the original goals has been implicitly synthesized.

6 Distributing SAT for Cooperative Planning

We now introduce a new way of distributing the satisfiability problem (see also [1] and [2]). The *distributed search* principle is widely applied in our approach, and the way the search space is divided is quite unusual. In fact, we mix distributed search with a particular kind of *multi-search*, i.e. we let each process tackle a simplification of the overall SAT problem, with the constraint that by joining the solution of every simplified sub-problem we get a solution for the original SAT problem (either a model for the entire formula or the assurance that such a model does not exist).

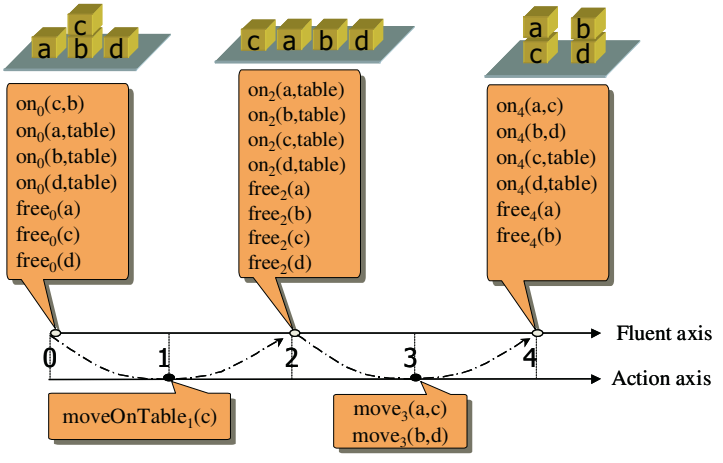


Fig. 4. A SAT-based solution to the sample planning instance.

Let us consider in what follows (with no loss of generality) that only *two* agents exist and work together. The key idea is to consider a CNF formula $\mathcal{F}$ as the conjunct of two formulas $\mathcal{F}_1$ and $\mathcal{F}_2$ in such a way that $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The agent A_1 is assigned the SAT problem on $\mathcal{F}_1$ and $\mathcal{F}_2$ is assigned to A_2. These two sub-problems are simpler than the original one, but without communication between A_1 and A_2 we are not guaranteed that the composition of the sub-solutions is consistent. We here present a mechanism to handle and solve this problem.

We consider *homogeneous* agents, in the sense that each agent is provided with exactly the same inference system. Moreover, we use a *peer agents* scheme (every agent plays the same role as the others, and has no special task with respect to the SAT problem under consideration).

The distribution of the instance is based on a partition of propositional variables between *shared* variables and *private* variables. There are two opposite perspectives to introduce such an approach. From one perspective, we can imagine that a parent CNF formula $\mathcal{F}$ is decomposed by distributing clauses into several CNF sub-formulas $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n$, in such a way that $\mathcal{F} = \cup_{i=1,..,n}\mathcal{F}_i$ (this decomposition may happen to be a *partition* of the parent formula, even though this is not supposed to be true in what follows). The set $V^S_{i,j} = Var(\mathcal{F}_i) \cap Var(\mathcal{F}_j)$ is the set of variables *shared* by $\mathcal{F}_i$ and $\mathcal{F}_j$ (the set of variables appearing in both $\mathcal{F}_i$ and $\mathcal{F}_j$). Thus, the set of *shared variables* of $\mathcal{F}_i$ is $V^S_i = \cup_{j \neq i} V^S_{i,j}$ and the set of its *private variables* is $var(\mathcal{F}_i) \setminus V^S_i$.

From another perspective, we may start from a set of formulas $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n$ (each one with its own distinct variables: $Var(\mathcal{F}_i) \cap Var(\mathcal{F}_j) = \emptyset$ for all $i \neq j$) and then define an equivalence relation $\overset{var}{=}$ defined on the set $\cup_{i=1,...,n} Var(\mathcal{F}_i)$, in such a way that $\mathcal{F}_i \ni v \overset{var}{=} w \in \mathcal{F}_j$ implies $i \neq j$. In this case, the set of *shared variables* of $\mathcal{F}_i$ is $V^S_i = \{v | \exists w. v \overset{var}{=} w\}$ and the set of its *private variables* is still $var(\mathcal{F}_i) \setminus V^S_i$.

Whichever the way of distinguishing shared variables from private ones, the meaning of the distinction is the same, and is related to the satisfiability of $\cup_{i=1,..,n}\mathcal{F}_i$: a shared variable must have the same truth value in all sub-formulas sharing it.

As an example, let us consider the basic case of two formulas $\mathcal{F}_1$ and $\mathcal{F}_2$ in CNF which *share* some propositional variables (according to the former notation of the two introduced above). We denote the set of shared variables with $V^S = Var(\mathcal{F}_1) \cap Var(\mathcal{F}_2)$. Variables in $Var(\mathcal{F}_1) \setminus V^S$ are *private* to $\mathcal{F}_1$, those in $Var(\mathcal{F}_2) \setminus V^S$ are *private* to $\mathcal{F}_2$.

The satisfiability of $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ depends on the satisfiability of $\mathcal{F}_1$ and $\mathcal{F}_2$ in the following way: whenever $\mathcal{F}$ is satisfiable, $\mathcal{F}_1$ and $\mathcal{F}_2$ are satisfiable as well. Conversely, the satisfiablity of $\mathcal{F}_1$ and $\mathcal{F}_2$ on its own does not ensure that a model for $\mathcal{F}$ exists.

To ensure the satisfiability of $\mathcal{F}$ (provided $\mathcal{F}_1$ and $\mathcal{F}_2$ are satisfiable) it is necessary that the projection on the shared variables of the set of models of $\mathcal{F}_1$ has at least one element in common with the projection on the shared variables of the set of models of $\mathcal{F}_2$.

As an example, let us consider the following two formulas:

$$\mathcal{F}_1 = \{\{A, \neg c\}, \{\neg B, \neg c, d\}, \{c, \neg d\}, \{B, c, d\}, \{\neg A, b\}\}$$
$$\mathcal{F}_2 = \{\{A, B, e\}, \{\neg B, e\}, \{\neg e, A, f\}, \{\neg f, A\}\}$$

Both of them are satisfiable, and it is:

$$\mathcal{M}(\mathcal{F}_1) = \{\{A, \neg B, c, d\}, \{A, \neg B, c, \neg d\}, \{\neg A, B, \neg c, \neg d\}\}$$
$$\mathcal{M}(\mathcal{F}_2) = \{\{A, B, e, f\}, \{A, B, e, \neg f\}, \{A, \neg B, e, f\},$$
$$\{A, \neg B, e, \neg f\}, \{A, \neg B, \neg e, f\}, \{A, \neg B, \neg e, \neg f\}\}$$

By projecting these models onto the set of shared variables $V^S = VAR(\mathcal{F}_1) \cap VAR(\mathcal{F}_2) = \{A, B\}$ we get $\mathcal{M}(\mathcal{F}_1) \downarrow V^S = \{\{A, \neg B\}, \{\neg A, B\}\}$ and $\mathcal{M}(\mathcal{F}_2) \downarrow V^S = \{\{A, B\}, \{A, \neg B\}\}$. So, the intersection of the projections is not empty, and we are ensured that $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ is satisfiable as well. In fact, it has eight different models, each of which assigns true to A, and false to B.

We can state this result more precisely.

Proposition. Given two formulas $\mathcal{F}_1$ and $\mathcal{F}_2$ such that $V^S = Var(\mathcal{F}_1) \cap Var(\mathcal{F}_2)$ is not empty, the formula $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ is satisfiable iff
$\mathcal{M}(\mathcal{F}_1) \downarrow V^S \cap \mathcal{M}(\mathcal{F}_2) \downarrow V^S \neq \emptyset$.

We just linked up the satisfiability of $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ to that of $\mathcal{F}_1$ and $\mathcal{F}_2$. A closely related subject is how to solve the SAT problem for $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ by dealing with $\mathcal{F}_1$ and $\mathcal{F}_2$ separately. In fact, if the SAT problems on $\mathcal{F}_1$ and the one on $\mathcal{F}_2$ were independently solved by two agents A_1 and A_2, we could observe two possible outcomes with respect to $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$: (1) if either A_1 or A_2 (or both of them) find their formula to be unsatisfiable, we are guaranteed that $\mathcal{F}$ is unsatisfiable as well; (2) if A_1 obtains a model $m_1 \in \mathcal{M}(\mathcal{F}_1)$ and A_2 does the same and obtains $m_2 \in \mathcal{M}(\mathcal{F}_2)$, there are three possible sub-cases: (2a) m_1 and

m_2 assign the same truth value to the variables in V^S; (2b) m_1 and m_2 do not assign the same truth value to the variables in V^S and no other couple $\langle m'_1, m'_2\rangle$ (with $m'_1 \in \mathcal{M}(\mathcal{F}_1)$ and $m'_2 \in \mathcal{M}(\mathcal{F}_2)$) does; (2c) m_1 and m_2 are not consistent on V^S, but some consistent couple $\langle m'_1, m'_2\rangle$ does exist.

We would like to avoid situations (2b) and (2c) as they are uninformative with respect to the satisfiability of $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. In fact, neither A_1 nor A_2 have enough information to distinguish between (2b) and (2c), but these situations are quite different, as $\mathcal{F}$ is unsatisfiable in (2b) and satisfiable in (2c).

We can solve this problem by allowing A_1 and A_2 to communicate during the search for a solution. The key idea is that whenever A_1 (1)*discovers* or (2)*decides* something regarding a shared variable, or a set of shared variables, it has to communicate that piece of information to A_2, and vice-versa. We first illustrate with an example what *discovering* information on shared variables means. Then, we address the topic of *shared decision*.

Given $\mathcal{F}_1 = \{\{A, \neg c\}, \{c, B, f\}, \{\neg f\}\}$, we have no clause in $\mathcal{F}_1$ which contains only shared variables. So, none of the three clauses in $\mathcal{F}_1$ makes any sense with respect to A_2. However, we can derive $\{\{A, \neg c\}, \{c, B\}\}$ and then $\{A, B\}$ by means of two steps of resolution among clauses in $\mathcal{F}_1$. Now we have a clause that contains only shared variables (we call it *shared clause*), and is derived using only $\mathcal{F}_1$. It is a sharable and meaningful information: every model of $\mathcal{F}_1$ and every model of $\mathcal{F}_2$ have to contain either A or B assigned to *true* in order for $\mathcal{F}_1 \cup \mathcal{F}_2$ to be satisfiable. This is not informative for $\mathcal{A}_1$ itself, as $\mathcal{F}_1 \rightarrow A \vee B$, but it could be informative for $\mathcal{F}_2$. So, A_1 communicates the clause $\{A, B\}$ to A_2. Equivalently, A_2 sends shared clauses he discovers to A_1.

When we look at the channel between the two agents, we see several shared clauses transferred in both ways. If we collect all those shared clauses, we obtain a growing CNF formula $\mathcal{F}^S_{(i)}$ (where i is the number of collected clauses) containing only shared variables, that is: a *shared formula*. For every i, it is both $\mathcal{M}(\mathcal{F}_1)\downarrow V^S \subseteq \mathcal{M}(\mathcal{F}^S_{(i)})$ and $\mathcal{M}(\mathcal{F}_2)\downarrow V^S \subseteq \mathcal{M}(\mathcal{F}^S_{(i)})$, so that $(\mathcal{M}(\mathcal{F}_1)\downarrow V^S) \cap (\mathcal{M}(\mathcal{F}_2)\downarrow V^S) = \mathcal{M}(\mathcal{F})\downarrow V^S \subseteq \mathcal{M}(\mathcal{F}^S_{(i)})$. The greater the number of shared clauses discovered (the i index), the fewer the models in $\mathcal{M}(\mathcal{F}^S_{(i)})$ not in $\mathcal{M}(\mathcal{F})|_{V^S}$. So, $\mathcal{M}(\mathcal{F}^S_{(i)})$ approximates $\mathcal{M}(\mathcal{F})\downarrow V^S$ as i increases (incidentally, this implies that the need for communication decreases as the search goes on, as agents accumulate knowledge about externally-generated constraints on the common variables).

The shared formula thus shrinks the set of possible models from the set containing all the possible partial assignments on the shared variables, towards the actual set of allowed models $\mathcal{M}(\mathcal{F})|_{V^S}$.

Even if the SAT problem consists of finding just *one* satisfying assignment (if it exists), each agent would like to know the *entire* projection of allowed models on the shared variables (model *enumeration*, a much more complex problem) to distinguish private models which are consistent with external constraints from those that are not. Unfortunately, this information is *not localized* as each agent sharing variables has to contribute to find out such a set. The solution is to avoid model enumeration on shared variables by considering the current $\mathcal{F}^S_{(i)}$

as a well suited approximation of $\mathcal{M}(\mathcal{F}) \downarrow V^S$. As soon as an agent discovers models in $\mathcal{F}^S_{(i)}$ not in $\mathcal{M}(\mathcal{F}) \downarrow V^S$ he shrinks the set of allowed common models by producing (and sending) a new shared clause. The other agent has to adapt to the new information, possibly backtracking on some private choices.

The shared formula is a kind of "negative" information: in the search for a model of F it is surely useless to consider any partial assignment which is inconsistent with $\mathcal{F}^S_{(i)}$. Many complete algorithms for satisfiability work incrementally, in so as they handle growing partial assignment and try to construct a model step by step. When a shared variable is involved in one such partial assignment, a kind of "positive" communication takes place between A_1 and A_2. Let us suppose A_1 decides to assign *true* to a shared variable V_i^S. A_1 should communicate this *decision* to A_2 and A_2 should hold it in due consideration, as it is wasted time for A_2 to work with the opposite hypothesis (V_i^S assigned to *false*). This is a "positive" kind of communication aimed at synchronizing working hypotheses on the shared variables used by the two agents. This way, the truth value assigned to V_i^S is a *shared working hypothesis*.

7 Distributed Planning as a Distributed SAT Problem

An interesting solution to the distributed planning problem can be obtained by joining distributed satisfiability and SAT-based planning. Simply stated, the problem is that of cooperating during the development of a plan which involves several agents. Each agent has its goals – generally not inconsistent with others' goals. These agents are supposed to live and act in a shared environment, so they are forced to synchronize their operations, and to possibly cooperate to reach shared goals or subgoals. The interesting question is whether or not an *early cooperation mechanism* can be introduced which allows for the production of *implicitly synchronized* plans, i.e.: each agent plans on its own, but communicates with other agents in such a way that mutually consistent plans are ensured as a result.

The advantages of this approach are manifold. The need for *synchronized* courses of actions in a multi-agent environment is met by construction, provided each agent adopts a suitable distributed planning scheme. Each agent only worries about its own goals, and *coordination* is obtained as a consequence of the planning process itself. *Cooperation* also comes out as a side effect of the planning effort. The quantity and roles of agents involved in a cooperative plan need not to be fixed in the domain theory, and a flexible approach can be adopted when dealing with new agents and/or new goals. Furthermore, a distribution of the computational effort underlying the planning task is implicitly achieved.

As an example, let us reconsider the toy planning instance of the previous section, and let us suppose that two agents act in this domain as represented in Figure 5. By looking at the portion of the world one agent is interest in, she is able to produce a SAT encoding of its planning instance, according to its private goal and to the domain theory which is supposed to be known to all the participants. We can suppose that the encoding given in the previous section

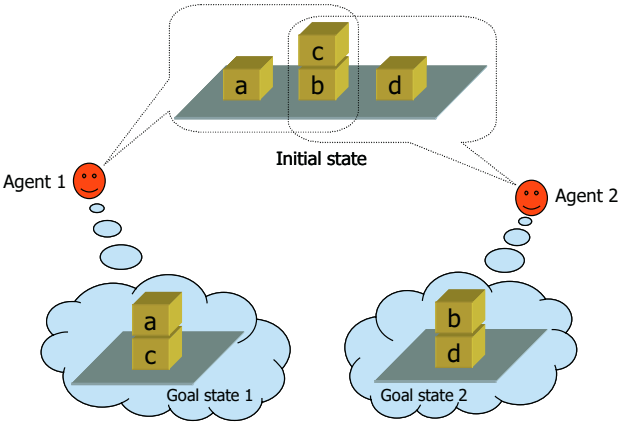


Fig. 5. Two agents thinking of their goals.

is used, so that each agent comes out with a propositional formula and with a symbol table recording correspondences between the planning instance and its propositional variables.

This first step – i.e. the generation of a SAT-encoding of the private planning problem – can be performed by each agent in isolation. Thereafter, the actual solution extraction process begins. Here it is where we want cooperation to take place, in order to ensure consistency (and possibly cooperation) among different individual plans.

To this end, our approach to distributed satisfiability presented in the previous section can be suitably exploited. The key idea is to consider the encoded instances of the two agents as two subsets $\mathcal{F}_1$ and $\mathcal{F}_2$ of a larger planning problem encoded as $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. Shared variables are those appearing in both formulas. The distributed satisfiability algorithm previously sketched is then run as represented in Figure 6. Each agent is able to build a compilation of its own planning instance, given the domain theory, the goal to be reached, and (the portion of) the initial situation she is interested in. During this encoding process, shared variables are identified as those variables related to the same (potential) event of fact in the knowledge base of both agents. Then, the distributed solving procedure is started. It consists of finding a satisfying assignment for both propositional encodings, that is consistent on shared variables. As all potential facts or events involving both agents are encoded by means of shared variables, the solution plan, if any, is such that no two contradictory plans ever come out. Finally, the decoding phase can be performed by each agent in isolation, thus embodying the shared model of the formula into a private but collaborative plan.

For the distributed planning instance presented in Figure 5, a solution is eventually extracted by each agent. The truth assignments to shared variables involve both agents (both partial models), while the truth values of private variables define private portions of the plan and facts only relevant to the agent owning such variables. Figure 7 shows the two partially overlapping plans the agents A_1 and A_2 obtain as a result of their cooperative planning effort.

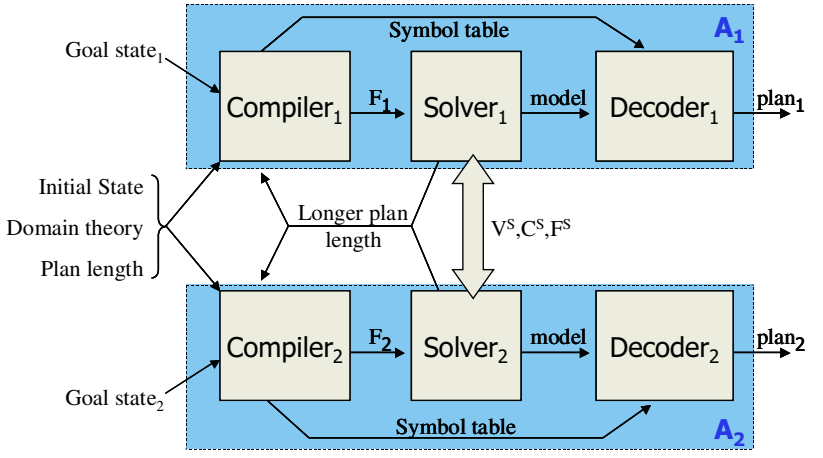


Fig. 6. Two agents cooperatively planning.

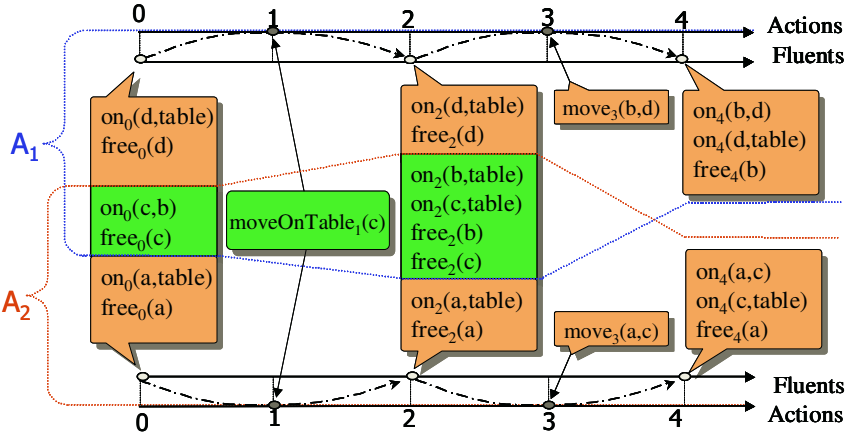


Fig. 7. Two partially overlapping plans.

The global plan obtained by merging these partially overlapping sub-plans is a plan achieving both goals: the models for the two sub-formulas (encoding the sub-problem of A_1 and that of A_2) are consistent on shared variables by construction, so that the entire assignment is a model for the global formula encoding the whole planning instance. As the global formula encodes the domain theory, the initial state and the conjunction of the goals of the two agents, a model of this formula maps to a plan which achieves both goals. In the sample plan of Figure 7 shared actions and facts are represented within dark-colored containers. The shared facts at time zero (the initial situation) are $on_0(c, b)$ and $free_0(c)$. Conversely, the facts $on_0(a, table)$ and $free_0(a)$ are private to A_1, while $on_0(d, table)$ and $free_0(d)$ are private to A_2. Four shared facts at time 2 are represented, while the final situation is made up of no shared facts. The only

shared action in the plan is that of moving block c onto the table at time 1. The variable represented in Figure 7 are those which are *true* in the satisfying assignment extracted. In addition to these shared variables, a number of further shared variables exist which are not represented in the figure; in particular, all those assigned to $false$. For example, the variable $moveOnTable_1(b)$ is shared, but both agents agree on assigning $false$ to it. Consequently, that action does not belong to the final plan and is not shown in Figure 7.

It still remains to decide which agent performs which action. For private actions, the problem is easily solved by assigning the execution of that action to the agent owning the encoding variable. Shared actions (action encoded by shared variables, hence representing cooperative portions of the plan) can be scheduled on-line during the execution phase, by assigning them to one of the owning agents, on the basis of any reasonable criterion. A different – and less preferable – approach arises in case the same action (at the same time step) is encoded by several different variables, one for each participant. In this case, the scheduling of actions is implicitly performed during the planning phase, and conflict exclusion clauses ensure that each action is performed by exactly one agent.

8 Discussion and Future Work

A number of issues need to be considered with respect to our proposal of a framework for distributed planning. All of them deserve further attention. We here briefly consider these topics in turn.

Shared Variables: The problem of selecting shared variables is a sensible one. From a syntactical point of view, the "sharedness" can be immediately detected by using a *unique name assumption*: two variables associated with the same fluent or action in the symbol tables of different agents, define a shared variable among those agents. For example, the fluent instance $on_2(b, table)$ has to be associated with the same shared variable in the encoding of every instance somewhere talking of the block b on the table at time 2: this variable represents a shared fact, and each individual plan has to agree with the other plans concerning whether or not the block b is on the table at time 2. The same holds for shared actions (i.e.: the same action instance taken at the same time in different plans). So, agents can easily agree on which variables are shared and which ones are not, once the initial situation for each agent is given.

The Initial Situation: In the simple encoding we presented in Section 5, the initial situation, together with the domain theory, determines which variables come out from the encoding, and consequently which ones are shared. The key point here is that the initial situation should be partitioned into portions, each one containing (all the) facts relevant to one single agent. Relevant facts are those talking about things one necessarily has to know in order to reach his goal. The partitioning has been empirically done in Figure 5, where the underlying rule is that an agent only knows the subset of stacks containing blocks involved in his private goal. More general rules are necessary here, to

extend the partitioning of the initial situation to other domains and other kinds of encodings. In addition to this, the ways and means of deciding private goals have to be studied. Several different solutions are possible here, also depending on the target application, ranging from a central mechanism to optimize task allocation to a distributed goal selection system.

Soundness: We would obtain soundness and completeness of the approach for free in case the conjunction of the encodings were equal to the encoding of the conjunction. But this is not the case in general. For example, when the planning instance considered in Figure 5 is tackled as a whole, the variable $on_i(a, d)$ for some i is surely generated, whereas it is not taken into account when a distributed solution is attempted, as in Figure 7. The reason why this variable (among others) is omitted, is that it relates objects of the world which are separated both in the initial state (w.r.t. stacks of blocks), and in the final one (w.r.t. goal decomposition). So, one may think of these variables as being forcefully assigned to false. It remains to be proved that variables and clauses kept out of the encoding do not affect soundness and completeness of the planning mechanism.

Effectiveness and Efficiency: Here we are mainly concerned with the construction of a really effective solution for distributed planning. On one hand, it is clear that two planning instances which do not interact at all (no shared variables, in our framework), can be solved in isolation more effectively than it is possible by considering the problem as a whole (think of two separate blocks worlds, and two planning problems in these two worlds; the joint problem obtained by conjoining the initial and final situations is still a planning problem in an enlarged blocks world, but it is harder to solve, as modularity is lost). On the other hand, it is also clear that when agents have heavily interacting goals and plans, the number of shared variables increases. The larger the number of shared variables with respect to non-shared ones, the greater the amount of time (and memory) spent in managing messages during the solving procedure. This trade-off needs to be analyzed, and it moves our attention towards the initial phase when goals are distributed among agents. Even if this topic is out of the scope of a distributed planner, it heavily impacts on the effectiveness of the approach.

Implementation: We are working on a prototype implementation of our distributed planner. Needless to say, a working implementation is the starting point for a thorough investigation of the real potentiality of our approach. Things work right in a few domain we are considering, and many of the questions that are still open can only find an "on the field" answer.

9 Conclusions

In the current AI literature, in multi agent systems, cooperative planning essentially means that each agent – participating in the cooperative achievement of a common goal – has to take into account, in its planning activity, that other agents are part of the scenario. In the paper we put forth a somewhat stronger view of cooperative planning, i.e. agents not only take into account that other

agents may affect the scenario with their actions, and that they may share the workload of executing plans to achieve common goals, but they can distribute among themselves the workload of generating plans, with the advantages coming from information sharing.

We are working on a framework for distributed planning, which relies on the "planning as satisfiability" paradigm, and allows for multi-agent cooperative planning. We aim at obtaining this by joining SAT-based planning with a particular approach to distributed propositional satisfiability.

The advantage of the proposed framework is in that each agent is enabled to plan on its own, and communicate with other agents during the planning process, in such a way that synchronized and possibly cooperative plans are ensured as a result.

We illustrated our elements of novelty over previous approaches, which either were built on ad-hoc distributed planning mechanisms (not fully exploiting the powerful techniques developed within the "single-planning-agent" framework), or did not actually distribute the planning effort in a significant way. Conversely, we are trying to develop a solution which get all these benefits with no loss of generality w.r.t. other known and well established planning techniques for single-agent domains. Much is left to do in this direction, but the key ideas reported in this paper will strongly drive the research.

By abstracting from the planning domain, further interesting applications of our proposal loom on the horizon: whenever (1) a SAT-compilation technique is available to solve problems from a certain domain of interest and (2) the domain itself is well suited for applying a multi-agent solution, then a team of cooperative agents can tackle the considered problem in a distributed way by means of a SAT-based distributed approach.

Acknowledgements

We acknowledge the support of MIUR, the Italian Ministry of Research (Project Moses) and ASI (the Italian Space Agency), that made the research reported in this paper possible.

We dedicate this paper to Joerg Siekmann, a friend and a colleague, whose research has been a source of inspiration in many occasions.

References

1. M. Benedetti. A New Way of Distributing Satisfiability. In H. R. Arabnia, editor, *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2001)*, volume 1, pages 295–300. CSREA, 2001.
2. M. Benedetti. *Bridging Refutation and Search in Propositional Satisfiability.* PhD Thesis, Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza", 2001.
3. D. L. Berre. Sat live! page: a dynamic collection of links on sat-related research. http://www.satlive.org, 2000.

4. M. Bohm and E. Speckenmeyer. A fast parallel SAT-solver – efficient workload balancing. Technical Report 94-159, University of Cologne, 1994.
5. M. P. Bonacina. A taxonomy of parallel strategies fo deduction. Technical Report May 1999, Department of Computer Science, University of Iowa, 1999.
6. S. Botelho and R. Alami. Multi-robot Cooperative Plan Enhancement. pages 100–110, 1999.
7. L. Carlucci Aiello, D. Nardi, and M. Schaerf. Reasoning about Knowledge and Ignorance. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988 (FGCS-88)*, pages 618–627. ICOT Press, 1988.
8. L. Carlucci Aiello, D. Nardi, and M. Schaerf. Reasoning about Reasoning in a Meta-Level Architecture. *International Journal of Applied Intelligence*, 1:55–67, 1991.
9. S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.
10. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
11. M. desJardins, E. Durfee, C. L. J. Ortiz, and M. Wolverton. A Survey of Research in Distributed, Continual Planning. 20(4):13–22, 1999.
12. E. H. Durfee. Planning in Distributed Artificial Intelligence. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 231–245. Wiley & Sons, 1996.
13. M. Ernst, T. Millstein, and D. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1169–1177, 1997.
14. H. Hoos and T. Stützle. Satlib – the satisfiability library. http://www.informatik.tu-darmstadt.de/AI/SATLIB, 1998.
15. B. Jurkowiak, C. M. Li, and G. Utard. Parallelizing Satz Using Dynamic Workload Balancing. In *Proceedings of Workshop on Theory and Applications of Satisfiability Testing (SAT'2001)*, volume 9 of *Electronic Notes in Discrete Mathematics*, pages 205–211. Elsevier Science, 2001.
16. H. Kautz and B. Selman. Planning as Satisfiability. pages 359–363, 1992.
17. H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, 1992.
18. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stichastic search. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 1194–1201, 1996.
19. H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning and Combinatorial Search*, pages 58–60, 1998.
20. L. Levin. Universal Sequential Search Problems. *Problems of Information Trasmission*, 9:265–266, 1973.
21. D. J. M. R. Garey. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.
22. F. Okushi. Parallel cooperative propositional theorem proving. *Artificial Intelligence*, 26:59–85, 1999.
23. D. S. Weld. Recent Advances in AI Planning. 20(Summer 1999):93–123, 1999.
24. H. Zhang, M. P. Bonacina, and J. Hsiang. Psato: a distributed propositional prover and its application to quasigroup. *Journal of Symbolic Computation*, 21:543–560, 1996.

Towards Comprehensive Computational Models for Plan-Based Control of Autonomous Robots

Michael Beetz

Munich University of Technology,
Department of Computer Science IX,
Orleanstr. 34,
D-81667 Munich, Germany

Abstract. In this paper we present an overview of recent developments in the plan-based control of autonomous robots. We identify computational principles that enable autonomous robots to accomplish complex, diverse, and dynamically changing tasks in challenging environments. These principles include plan-based high-level control, probabilistic reasoning, plan transformation, and context and resource-adaptive reasoning. We will argue that the development of comprehensive and integrated computational models of plan-based control requires us to consider different aspects of plan-based control – plan representation, reasoning, execution, and learning – together and not in isolation. This integrated approach enables us to exploit synergies between the different aspects and thereby come up with simpler and more powerful computational models.
In the second part of the paper we describe *Structured Reactive Controllers (SRCs)*, our own approach to the development of a comprehensive computational model for the plan-based control of robotic agents. We show how the principles, described in the first part of the paper, are incorporated into the SRCs and summarize results of several long-term experiments that demonstrate the practicality of SRCs.

1 Introduction

In recent years, autonomous robots, including XAVIER, MARTHA [AFH+98], RHINO [BCF+00,BAB+01], MINERVA, and REMOTE AGENT, have shown impressive performance in longterm demonstrations. In NASA's Deep Space program, for example, an autonomous spacecraft controller, called the *Remote Agent* [MNPW98], has autonomously performed a scientific experiment in space. At Carnegie Mellon University XAVIER [SGH+97], another autonomous mobile robot, has navigated through an office environment for more than a year, allowing people to issue navigation commands and monitor their execution via the Internet. In 1998, MINERVA [TBB+00] acted for thirteen days as a museum tourguide in the Smithsonian Museum, and led several thousand people through an exhibition.

These autonomous robots have in common that they perform plan-based control in order to achieve better problem-solving competence. In the plan-based approach robots generate control actions by maintaining and executing a plan that is effective and has a high expected utility with respect to the robots' current goals and beliefs. Plans are

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 514–527, 2005.

robot control programs that a robot cannot only execute but also reason about and manipulate [McD92a]. Thus a plan-based controller is able to manage and adapt the robot's intended course of action – the plan – while executing it and can thereby better achieve complex and changing tasks. The plans used for autonomous robot control are often reactive plans, that is they specify how the robots are to respond in terms of low-level control actions to continually arriving sensory data in order to accomplish their objectives. The use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities.

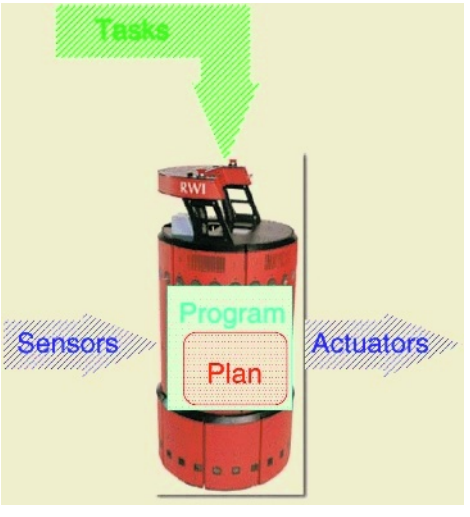


Fig. 1. Plan-based control of robotic agents. The control program specifies how the robot is to respond to sensory input to accomplish its task. The plan is the part of the control program that the robot explicitly reasons about and manipulates.

To be reliable and efficient, autonomous robots must flexibly interleave their tasks and quickly adapt their courses of action to changing circumstances. Recomputing the best possible course of action whenever some aspect of the robot's situation changes is not feasible but can often be made so if the robots' controllers explicitly manage the robots' beliefs and current goals and revise their plans accordingly. The use of plans helps to mitigate this situation in at least two ways. First, it decouples computationally intensive control decisions from the time pressure that dominates the feedback loops. Precomputed control decisions need to be reconsidered only if the conditions that justify the decisons change. Second, plans can be used to focus the search for appropriate control decisions. The can neglect control decisions that are incompatible with its intended plan of action.

In the remainder of this paper we proceed as follows. In the first part, we describe principles and building blocks of computational models for plan-based control. In the second part, we will then outline our initial steps towards such a comprehensive computational model that contains the building blocks introduced in the first part.

2 Principles of Plan-Based Control

Plans in plan-based control have two roles. They are both executable prescriptions that can be interpreted by the robot to accomplish its jobs and syntactic objects that can be synthesized and revised by the robot to meet the robot's criterion of utility. Besides having means for representing plans, plan-based controllers must also be equipped with tools that enable planning processes to (1) project what might happen when a robot controller gets executed and return the result as an execution scenario; (2) infer what

might be wrong with a robot controller given an execution scenario; and (3) perform complex revisions on robot controllers.

Let us now consider some of the key issues in the plan-based control of robotic agents: *dynamic system perspective, probabilistic reasoning, symbol grounding*, and *context and resource-adaptive reasoning.*

Dynamic System Perspective. Since flexibility and responsiveness to changing situations are important characteristics of the robot behavior, we use ***dynamic systems*** as the primary abstract model for programming the operation of the integrated plan-based controller (see figure 2). In this model, the state of the world evolves through the interaction of two processes: the controlling process – the robot's control system – and the controlled process, which comprises events in the environment, physical movements of the robot and sensing operations. For complex dynamic systems, it is often useful to further decompose the controlled process into an environment and a sensing process. The environment process changes the world state and the sensing process maps the world state into the sensor data received by the robot. This suggests making a similar decomposition of the controlling process into state estimation and action generation processes. State estimation processes compute the robot's beliefs about the state of the controlled system. Auxiliary monitoring processes signal system states for which the controlling process is waiting. An action generation process specifies the control signals supplied to the controlled process as a response to the estimated system state.

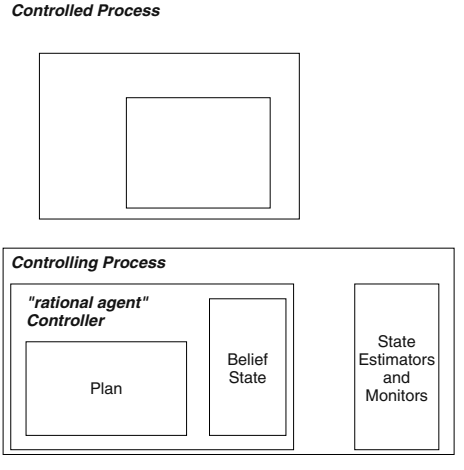


Fig. 2. Block diagram of our dynamic system model for autonomous robot control. Processes are depicted as boxes and interactions as arcs.

The main consequence of this model is that robot action plans must control concurrent and continuous processes both flexibly and reliably.

Probabilistic Reasoning. Probabilistic reasoning is a key technique employed in the control of autonomous robots. Probabilistic reasoning is employed in a number of different ways.

First, plan generation and revision methods compute plans that have a probability of achieving a given goal with a probability higher than a specified threshold or they compute plans with the best expected cost benefit trade-off [BDH98,KHW95,DHW94]. To employ such probabilistic planning techniques actions are represented through probabilistic effect models and the planning techniques consider probability distributions over world states rather than the states themselves. The advantage of these techniques is that they can properly handle the uncertainty caused by incomplete knowledge and inaccurate and unreliable sensors and the uncertainty resulting from non-deterministic

action effects. The main problem associated with these techniques are the computational cost associated with the application of these techniques.

The second area in plan-based control where probabilistic reasoning techniques are heavily used is the interpretation of sensor data acquired by the robots' sensors [Thr00]. The plan-based high-level control of robotic agents is founded on abstract perceptions of the current state of objects, the robot, and its environment. In order to derive such abstract perceptions from local and inaccurate sensors robustly, plan-based controllers often employ probabilistic state estimation techniques [SB01]. The state estimators maintain the probability densities for the states of objects over time. Whenever state information is requested by the planning component, they provide the most likely state of the objects.

The probability density of an object's state conditioned on the sensor measurements received so far contains all the information which is available about the object. Based on this density, one is not only able to determine the most likely state of the object, but one can also derive even more meaningful statistics like the variance and entropy of the current estimate. In this way, the high-level system is able to reason about the reliability of an estimate.

A third application field of probabilistic reasoning is learning. Probabilistic reasoning techniques enable robots to learn symbolic actions, probabilistic action models, and competent action selection strategies from experience.

Symbol Grounding. One of the key difficulties in the application of plan-based control techniques to object manipulation tasks is the symbol grounding or anchoring problem. In most plan representations constants used in the instantiations of plan steps denote objects in the world. This is a crude oversimplification because robots often do not have direct physical access to the objects themselves. Rather the control systems must use object descriptions that are computed from sensor data in order to manipulate objects. The use of object descriptions rather than objects to instantiate manipulation actions yields problems such as ambiguous, inaccurate, and invalid object descriptions. Powerful computational models of plan-based control must therefore have much more expressive representational means to make these problems transparent to the planning techniques. Several researchers [Fir89,McD90,CS00] have developed techniques to incorporate object descriptions into plan-based control.

Plan Transformation. Another key issue in the plan-based control of robots, in particular for those robots that are to act in dynamic and complex environments, is the fast formation and adaptation of plans. A very common idea for achieving fast plan formation is the idea of a *plan library*, a collection of canned plans for achieving standard tasks in standard situations [McD92b]. However, such plans cannot be assumed to execute optimally. In a situation where an unexpected opportunity presents itself during the execution of the robot's tasks, for example, a canned plan will have trouble testing for the subtle consequences that might be implied by an alteration to its current plan. The decision criteria to take or ignore such opportunities must typically be hardwired into the canned plans when the plan library is built.

An alternative is to equip a robot with *self-adapting plans*, which carry out plans with the constraint that, whenever a specific belief of the robot changes, a runtime plan adaptation process is triggered. Upon being triggered, the adaptors decide whether

plan revisions are necessary and, if so, perform them. Plan adaptation processes are specified explicitly, modularly, and transparently and are implemented using declarative plan transformation rules.

Context and Resource-Adaptive Operation. To make its control decisions in a timely manner the plan-based controller applies various resource-adaptive inference methods [Zil96]. These enable the controller to trade off accuracy and the risk of making wrong decisions against the computational resources consumed to arrive at those decisions. Moreover, the results of the resource-adaptive reasoning are employed to adapt the execution modes of the process in response to the robot's context [BACM98].

3 Building Blocks of Plan-Based Control

The building blocks of plan-based control are the representation of plans, the execution of plans, various forms of automatic learning, and reasoning about plans, including plan generation and transformation, and teleological, causal, and temporal reasoning.

But before we dive in and discuss the building blocks of modern plan-based control models let us first get an intuition of how traditional robot planning techniques function.

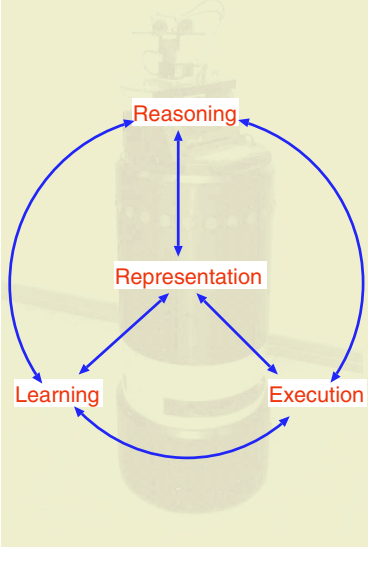


Fig. 3. The main components of plan-based control are plan representation, execution, learning, and reasoning and their interactions.

Most of these techniques are based on the problem-space hypothesis [New90]: they assume problems can be adequately stated using a state space and a set of discrete and atomic actions that transform states to successor states. A solution is an action sequence that transforms any situation satisfying a given initial state description into another state that satisfies the given goal. Plan generation is the key inferential task in this problem-solving paradigm.

As a consequence, representational means are primarily designed to simplify plan generation from first principles. Problem space plans are typically used in layered architectures [BFG+97], which run planning and execution at different levels of abstraction and time scales. In these approaches planning processes use models that are too abstract for predicting all consequences of the decisions they make and planning processes cannot exploit the control structures provided by the lower layer. Therefore they lack appropriate means for specifying flexible and reliable behavior and plans can only provide guidelines for task achievement.

Contrary to the plan space approach, plan-based control of robotic agents takes the stand that there is a number of inference tasks necessary for the control of an autonomous robot that are equally important. These inference tasks include ones that enable the competent execution of given plans, ones that allow for learning plans and

other aspects of plan-based control, and various reasoning tasks, which comprise the generation and assessment of alternative plans, monitoring the execution of a plan, and failure recovery.

These different inference tasks are performed on a common data structure: the plan. Consequently, the key design issues of plan-based control techniques are representational and inferential adequacy and inferential and acquisitional efficiency as key criteria for designing domain knowledge representations [RK91]. Transferring these notions to plan-based control, we consider the representational adequacy of plan representations to be their ability to specify the necessary control patterns and the intentions of the robots. Inferential adequacy is the ability to infer information necessary for dynamically managing, adjusting, and adapting the intended plan during its execution. Inferential efficiency is concerned with the time resources that are required for plan management. Finally, acquisitional efficiency systems is the degree to which they support the acquisition of new plan schemata and planning knowledge.

To perform the necessary reasoning tasks the plan management mechanisms must be equipped with inference techniques to infer the purpose of subplans, find subplans with a particular purpose, automatically generate a plan that can achieve some goal, determine flaws in the behavior that is caused by subplans, and estimate how good the behavior caused by a subplan is with respect to the robot's utility model. Pollack and Horty [PH99] stress the point that maintaining an appropriate and working plan requires the robot to perform various kinds of plan management operations including plan generation, plan elaboration, commitment management, environment monitoring, model- and diagnosis-based plan repair, and plan failure prediction.

It does not suffice that plan management mechanisms can merely perform these inference techniques but they have to perform them fast. The generation of effective goal-directed behavior in settings where the robots lack perfect knowledge about the environment and the outcomes of actions and environments are complex and dynamic, requires robots to maintain appropriate plans during their activity. They cannot afford to entirely replan their intended course of action every time their beliefs change.

To specify competent problem-solving behavior the plans that are reasoned about and manipulated must have the expressiveness of reactive plan languages. In addition to being capable of producing flexible and reliable behavior, the syntactic structure of plans should mirror the control patterns that cause the robot's behavior – they should be realistic models of how the robot achieves its intentions. Plans cannot abstract away from the fact that they generate concurrent, event-driven control processes without the robot losing the capability to predict and forestall many kinds of plan execution failures. A representationally adequate plan representation for robotic agents must also support the control and proper use of the robot's different mechanisms for perception, deliberation, action, and communication. The full exploitation of the robot's different mechanisms requires mechanism-specific control patterns. Control patterns that allow for effective image processing differ from those needed for flexible communication, which in turn differ from those that enable reliable and fast navigation. To fully exploit the robot's different mechanisms, their control must be transparently and explicitly represented as part of the robot's plans. The explicit representation of mechanism control

enables the robot to apply the same kinds of planning and learning techniques to all mechanisms and their interaction.

The defining characteristic of plan-based control is that these issues are considered together: plan representation and the different inference tasks are not studied in isolation but in conjunction with the other inference tasks. The advantage of this approach is that we can exploit synergies between the different aspects of plan-based control.

Plan management capabilities simplify the plan execution problem because programmers do not have to design plans that deal with all contingencies. Rather plans can be automatically adapted at execution time when the particular circumstances under which the plan has to work are known. Plan execution mechanisms can also employ reasoning mechanisms in order to get a broader coverage of problem-solving situations. The REMOTE AGENT, for example, employs propositional reasoning to derive the most appropriate actions to achieve the respective immediate goals [WN97,NW97]. On the other side, competent plan execution capabilities free the plan management mechanism from reasoning through all details. Reasoning techniques such as diagnostic and teleological reasoning are employed in transformational learning techniques in order to perform better informed learning decisions and thereby speed up the learning process [BB00]. Skill learning mechanisms have also been applied to the problem of learning effective plan revision methods [Sus77]. There is also a strong interaction between the learning and execution mechanisms in plan-based control. Learning mechanisms are used to adapt execution plans in order to increase their performance . Competent execution mechanisms enable the learning mechanisms to focus on strategical aspects of problem-solving tasks.

4 Structured Reactive Controllers: A Computational Model of Plan-Based Control

After having described the general components of computational models of plan-based control I want to give you now a brief overview of our own approach to the development of such integrated computational models. The robot controllers that realize this computational model are called ***Structured Reactive Controllers (SRCs)*** [Bee01]. Structured Reactive Controllers are self-adapting plans that specify concurrent reactive behavior. They revise themselves during the execution of specified user commands in order to exploit opportunities and avoid predictable problems. They are also capable of experience-based learning.

Structured Reactive Controllers use a very expressive plan language, called RPL [McD91], and a number of software tools for predicting the effects of executing plans, for teleological and causal reasoning about plans, for revising plans during their execution, and for automatically learning routine plans.

Given a set of jobs, an SRC concurrently executes the default routines for each individual job. These routine activities are general and flexible and work well in standard situations. They can cope well with partly unknown and changing environments, run concurrently, handle interrupts, and control robots without assistance over extended periods. For standard situations, the execution of these routine activities causes the robot to exhibit an appropriate behaviour while achieving its purpose. While it executes rou-

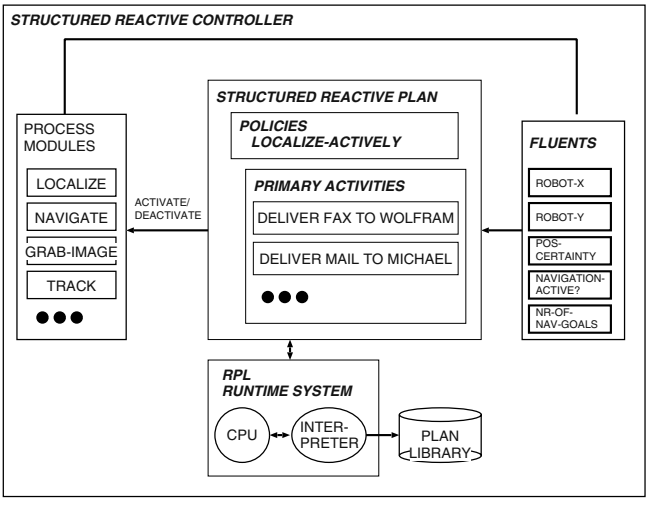


Fig. 4. Components of a structured reactive controller. The structured reactive plan specifies how the robot responds to changes of its fluents, registers that are asynchronously set by the sensing processes. The interpretation of the structured reactive plan results in the activation, parameterization, and deactivation of process modules that execute and monitor the physical continuous control processes.

tine activities, the SRC also tries to determine whether its routines might interfere with each other and monitors robot operation for non-standard situations. If one is found, it will try to anticipate behaviour flaws by predicting how its routine activities might work in these non-standard situations. If necessary, it revises its routines to make them robust for this kind of situation. Finally, it integrates the proposed revisions into the activities it is pursuing.

Transformational Planning of Concurrent Reactive Plans. Consider the following plan adaptor, which illustrates the planning techniques employed by SRCs.

With plan adaptor Whenever the robot detects an open door
that was assumed to be closed
if this situation ***is an opportunity***
then it ***changes its course of action***
to make use of the opportunity

Concurrent reactive plan

The plan adaptor is triggered by a change of its belief about an door being open or closed. Upon being triggered the adaptor decides whether a change in the intended course of activity is suitable and if so performs it. The process of plan adaptation is realized through transformational planning [McD92b,Bee00].

Transformational planning is implemented as a search in plan space. A node in the space is a proposed plan; the initial node is the default plan created using the plan library. A step in the space requires three phases. First, a plan adaptor *projects* a plan to

generate sample execution scenarios for it. Then, in the *criticism* phase, a plan adaptor examines these execution scenarios to estimate how good the plan is and to predict possible plan failures. It diagnoses the projected plan failures by classifying them in a taxonomy of failure models. The failure models serve as indices into a set of transformation rules that are applied in the third phase, *revision*, to produce new versions of the plan that are, we hope, improvements.

Prediction in Structured Reactive Controllers. Temporal projection, the process of predicting what will happen when a robot executes its plan, is essential for many robots to successfully plan courses of action. To be able to project their plans, robots must have causal models that represent the effects of their actions. These causal models should be sufficiently realistic to predict the behavior generated by modern autonomous robot controllers accurately enough to foresee a significant range of real execution problems. This can be achieved if action models reflect the facts that physical robot actions cause continuous change; that controllers are reactive systems; that the robot is executing multiple physical and sensing actions; and that the robot is uncertain about the effects of its actions and the state of the environment.

The problem of using such realistic action models is obvious. Nontrivial concurrent plans for controlling robots reliably are very complex. There are usually several control processes active. Many more are dormant, waiting for conditions that trigger their execution. The branching factors for possible future states – not to mention the distribution of execution scenarios that they might generate – are immense. The accurate computation of this probability distribution is prohibitively expensive in terms of computational resources.

Learning Symbolic Robot Plans. We have already stressed the importance of representing the plans that the robot has committed to execute explicitly as a means of economically using the limited computational resources for flexible task execution and effective action planning. However, this raises the question of how such plans can be obtained. Many autonomous mobile robots consider navigation as a Markov decision problem. They model the navigation behavior as a finite state automaton in which navigation actions cause stochastic state transitions. The robot is rewarded for reaching its destination quickly and reliably. A solution for such problems is a mapping from states to actions that maximises the accumulated reward. Such state-action mappings are inappropriate for teleological and diagnostic reasoning, which are necessary to adapt quickly to changing circumstances and quickly respond to exceptional situations.

We have therefore developed XFRMLEARN [BB00], a learning component that builds up explicit symbolic navigation plans automatically. Given a navigation task, XFRMLEARN learns to structure continuous navigation behaviour and represents the learned structure as compact and transparent plans. The structured plans are obtained by starting with monolithic default plans that are optimized for average performance and adding subplans to improve the navigation performance for the given task.

XFRMLEARN's learning algorithm works as follows. XFRMLEARN starts with a default plan that transforms a navigation problem into an MDP problem and passes the MDP problem to RHINO's navigation system. After RHINO's path planner has determined the navigation policy the navigation system activates the collision avoidance module for the execution of the resulting policy. XFRMLEARN records the resulting

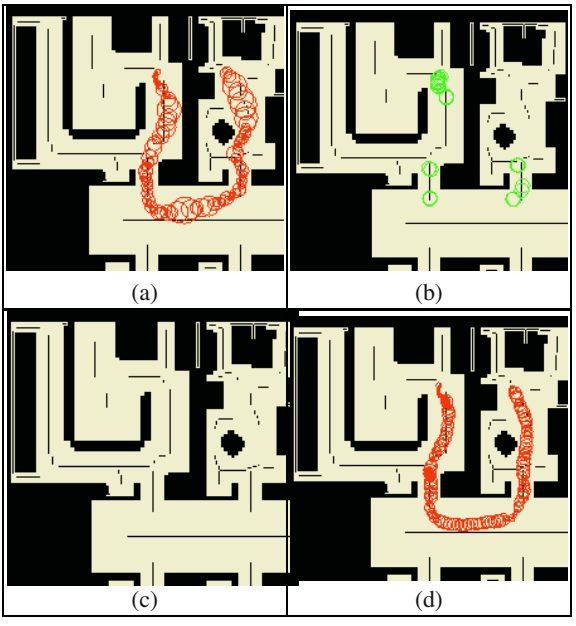


Fig. 5. The figure visualizes a summary of a learning session: A behaviour trace of the default plan (a); behavior stretches where the robot moves conspiciously slowly (b); the added subplans in the learned navigation plan (c); and a behaviour trace of the learned plan, which is on average 29% faster than the default plan (d).

navigation behaviour and looks for stretches of behaviour that could be possibly improved. XFRMLEARN then tries to explain the improvable behaviour stretches using causal knowledge and its knowledge about the environment. These explanations are then used to index promising plan revision methods that introduce and modify subplans. The revisions are subsequently tested in a series of experiments to decide whether they are likely to improve the navigation behaviour. Successful subplans are incorporated into the symbolic plan. An learning session is shown in figure 5.

Using this algorithm can autonomously learn compact and well-structured symbolic navigation plans by using MDP navigation policies as default plans and repeatedly inserting subplans into the plans that significantly improve the navigation performance. The plans learned by XFRMLEARN support action planning and opportunistic task execution by providing plan-based controllers with subplans such as traverse a particular narrow passage or an open area. More specifically, navigation plans (1) can generate qualitative events from continuous behaviour, such as entering a narrow passage; (2) support online adaptation of the navigation behaviour (drive more carefully while traversing a particular narrow passage) [Bee99], and (3) allow for compact and realistic symbolic predictions of continuous, sensor-driven behaviour [BG00].

5 Long-Term Demonstrations

This section describes several experiments (figure 6) that evaluate the reliability and flexibility of the RHINO system and the possible performance gains that it can achieve.

(a) (b) (c)

Fig. 6. The mobile robots MINERVA (a) and the RWI B21 robot RHINO (c) that are used in the experiments.

The flexibility and reliability of runtime plan management and plan transformation has been extensively tested in a museum tourguide application. The robot's purpose was to guide people through a museum, explaining the exhibits to be seen along the robot's route. MINERVA (figure 6(a)) operated in the "Smithsonian Museum" in Washington for a period of thirteen days [TBB+99]. It employed an SRC as its high-level controller. During its period of operation, it was in service for more than 94 hours, completed 620 tours, showed 2668 exhibits, and travelled over a distance of more than 44 kilometers. The SRC directed MINERVA's course of action in a feedback loop that was carried out more than three times a second. MINERVA used plan adaptors for the installment of new commands, the deletion of completed plans, and for tour scheduling. MINERVA made about 3200 execution time plan transformations while performing its tourguide job. MINERVA's plan-based controller differs from RHINO's only with respect to its top-level plans in plan library and some of the plan adaptors that are used.

In another experiment we have evaluated the capabilities of plan-based controllers to perform predictive plan management. This experiment has shown that predictive plan transformation can improve the performance by outperforming controllers without predictive capabilities in situations which require foresight while at the same time retaining their performance in situations that require no foresight. Figure 7 pictures an execution trace for a sample problem-solving scenario.

6 Conclusions

Our longterm research goal is to understand and build autonomous robot controllers that can carry out daily jobs in offices and factories with a reliability and efficiency comparable to people. We believe that many behavior patterns, such as exploiting opportunities, making appropriate assumptions, and acting reliably while making assumptions, that make everyday activity efficient and reliable require plan-based control and the specification of concurrent, reactive plans.

In this paper we have presented an overview of recent developments in the area of plan-based control of autonomous robots. Computational principles including plan-based high-level control, probabilistic reasoning, symbol anchoring, plan transforma-

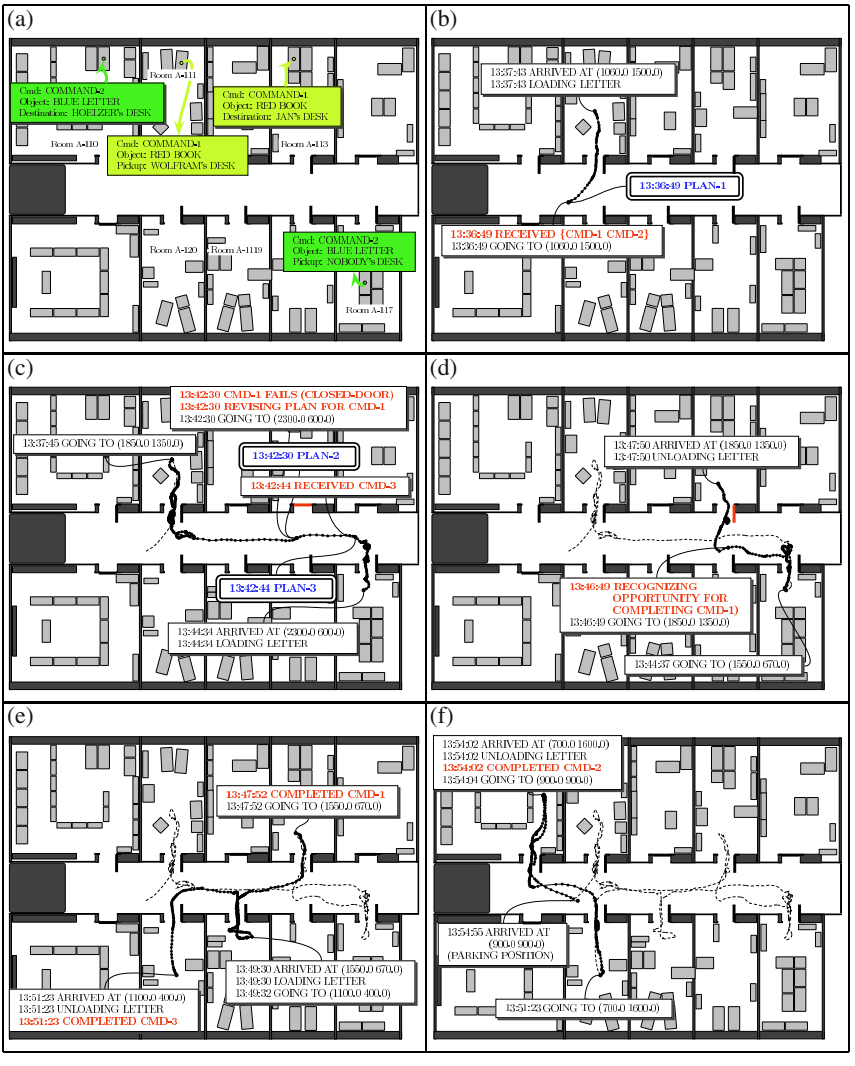


Fig. 7. Execution trace for a delivery tour. RHINO receives two commands 7(a). Upon receiving the two commands the SRC puts plans for the commands into the plan, computes an appropriate schedule, and installs it. It also adds a control process that monitors that the rooms it must enter are open. The order of the delivery steps are that RHINO starts with picking up the book (Fig. 7(b)) and delivering it in A-113. After RHINO has left room A-111, it notices that room A-113 is closed (Fig. 7(c)). Because RHINO cannot complete the delivery of the book the SRC revises the plan by transforming the completion of the delivery into an opportunity. RHINO receives a third command which is integrated into the current schedule (Fig. 7(d)). As it passes room A-113 on its way to A-119 it notices that the door is now open and takes the opportunity to complete the first command (Fig. 7(d)). After that it completes the remaining steps as planned (Fig. 7(e-f)).

tion, and context and resource-adaptive reasoning are incorporated in a number of state-of-the-art systems.

We believe that a necessary step towards more powerful plan-based robot controllers is the development of comprehensive and integrated computational models that address issues plan representation, reasoning, execution, and learning at the same time. A key component of such a computation model is the design of the plan representation language such that it allows for flexible and reliable behavior specifications, computationally feasible inference, stability in the case of runtime plan revisions, and automatic learning of symbolic plans for robot control.

Comprehensive computational models will enable us to tackle new application areas, such as the plan-based control of robot soccer teams, and longterm application challenges, for example, the robotic assistance of elderly people and the plan-based control of robotic rescue teams after disasters such as earthquakes.

References

[AFH+98] R. Alami, S. Fleury, M. Herb, F. Ingrand, and F. Robert. Multi robot cooperation in the Martha project. *IEEE Robotics and Automation Magazine*, 5(1), 1998.

[BAB+01] M. Beetz, T. Arbuckle, M. Bennewitz, W. Burgard, A. Cremers, D. Fox, H. Grosskreutz, D. Hähnel, and D. Schulz. Integrated plan-based control of autonomous service robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65, 2001.

[BACM98] M. Beetz, T. Arbuckle, A. Cremers, and M. Mann. Transparent, flexible, and resource-adaptive image processing for autonomous service robots. In H. Prade, editor, *Procs. of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 632–636, 1998.

[BB00] M. Beetz and T. Belker. Environment and task adaptation for robotic agents. In W. Horn, editor, *Procs. of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 2000.

[BCF+00] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.

[BDH98] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of AI research*, 1998.

[Bee99] M. Beetz. Structured reactive controllers – a computational model of everyday activity. In O. Etzioni, J. Müller, and J. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents*, pages 228–235, 1999.

[Bee00] M. Beetz. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, volume LNAI 1772 of *Lecture Notes in Artificial Intelligence*. Springer Publishers, 2000.

[Bee01] M. Beetz. Structured Reactive Controllers. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:25–55, March/June 2001.

[BFG+97] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1), 1997.

[BG00] M. Beetz and H. Grosskreutz. Probabilistic hybrid action models for predicting concurrent percept-driven robot behavior. In *Proceedings of the Fifth International Conference on AI Planning Systems*, Breckenridge, CO, 2000. AAAI Press.

[CS00] S. Coradeschi and A. Saffiotti. Anchoring symbols to sensor data: preliminary report. In *Proc. of the 17th AAAI Conf.*, pages 129–135, Menlo Park, CA, 2000. AAAI Press.

[DHW94] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In K. Hammond, editor, *Proc. 2nd. Int. Conf. on AI Planning Systems*. Morgan Kaufmann, 1994.

[Fir89] J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. Technical report 672, Yale University, Department of Computer Science, 1989.

[KHW95] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[McD90] D. McDermott. Planning reactive behavior: A progress report. In K. Sycara, editor, *Innovative Approaches to Planning, Scheduling and Control*, pages 450–458, San Mateo, CA, 1990. Kaufmann.

[McD91] D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[McD92a] D. McDermott. Robot planning. *AI Magazine*, 13(2):55–79, 1992.

[McD92b] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[MNPW98] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: to go boldly where no AI system has gone before. *Artificial Intelligence*, 103(1–2):5–47, 1998.

[New90] A. Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[NW97] P. Nayak and B. Williams. Fast context switching in real-time propositional reasoning. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 50–56, Menlo Park, 1997. AAAI Press.

[PH99] M. Pollack and J. Horty. There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AI Magazine*, 20(4):71–84, 1999.

[RK91] E. Rich and K. Knight. *Artificial Intelligence*. McGraw Hill, New York, 1991.

[SB01] D. Schulz and W. Burgard. Probabilistic state estimation of dynamic objects with a moving mobile robot. *Robotics and Autonomous Systems*, 34(2-3):107–115, 2001.

[SGH+97] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso. Xavier: Experience with a layered robot architecture. *ACM magazine* Intelligence, 1997.

[Sus77] G. Sussman. *A Computer Model of Skill Acquisition*, volume 1 of *Aritficial Intelligence Series*. American Elsevier, New York, NY, 1977.

[TBB+99] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[TBB+00] S. Thrun, M. Beetz, M. Bennewitz, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, 2000. to appear.

[Thr00] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[WN97] B. Williams and P. Nayak. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1178–1185, San Francisco, 1997. Morgan Kaufmann Publishers.

[Zil96] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

Agents with Exact Foreknowledge*

Jim Doran

Department of Computer Science,
University of Essex, Colchester, UK, CO4 3SQ
doraj@essex.ac.uk

Abstract. Computational experiments are reported involving the concept of *foreknowledge*, an agent's direct, unmediated and accurate, but possibly incomplete, awareness of its future including states and events involving the agent itself. Foreknowledge is used here as a conceptual tool with which to explore certain issues around time and rationality. We first explain how foreknowledge in this sense may be given to the agents in an agent society on a computer. The generation of a world history is viewed as a process of solving a constraint satisfaction problem. Then we seek to understand the circumstances in which foreknowledge may be either beneficial or detrimental to a society of rational agents. A special case is when agents have foreknowledge of their own "deaths". An experimental interpretation of this special case has been implemented, and results are presented and discussed. Finally, the work reported is briefly discussed in the context of current theories of time.

1 Introduction

This paper is about *societies of* agents located in a (simulated) physical environment or *artificial world*, created within a digital computer. Typically the agents within the societies act, move, perceive locally and have (reactive or deliberative) processes of cognition. There may also be some form of inter-agent communication. Interest commonly lies in the interaction between the agents and their environment, in processes of inter-agent co-operation and competition, and in macro-level emergent behaviour and its dependence upon micro-level properties.

Artificial societies in artificial worlds will be used to address certain questions that arise from a consideration of *foreknowledge*, taken to mean an agent's unmediated and fully accurate (if incomplete) awareness of future states or events. An agent with foreknowledge may be reliably aware of (some of) its own future actions and decisions, even thought these are yet to be chosen. Foreknowledge is more than and different in nature from prediction, which relies on knowledge only of the past, although foreknowledge and prediction may deliver identical results in particular circumstances, for example, if the predictive process is sound and has sufficient information available to it, and if the world is indeed predictable.

* Part of the content of this article was presented at the International Conference on Computer Simulation and the Social Sciences held in Cortona (Italy) in September 1997 [1].

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 528–542, 2005.

We address foreknowledge not as something for which there is any compelling scientific evidence in reality[1], but as a conceptual tool with which to explore certain issues around time and rationality.

Specifically we shall ask:

- *Is it possible for agents in an artificial world to have foreknowledge?*
- *Can agents with foreknowledge in an artificial world have choice?*
- *Is foreknowledge beneficial or detrimental to an artificial society of (rational) agents?*

These questions are posed in the realm of artificial worlds on a computer but touch upon major and much debated philosophical issues. The use of the concept of foreknowledge to provoke interesting questions in the theory of time is unusual but not unprecedented (see, for example, [4]).

1.1 Views of Time

Foreknowledge as defined here is intimately bound up with notions of time. The most common technical view taken of time (e.g. [10]) is that of "block time" or a "space-time manifold of events". This corresponds to a complete history irrespective of any particular generating process. From this perspective time does not "flow" or "pass", but is integrated with space in a fixed space-time structure itself outside any time. This concept has, of course, proved extremely powerful at the heart of the general theory of relativity. It contrasts with the more intuitive and "man in the street" view of time in which a transient "now" separates an experienced and unchangeable "past" from an un-experienced and malleable "future", and in which future inexorably becomes present, and present past [6].

Another model of time is often deployed within artificial intelligence research. If histories are non-deterministic, then we might naturally take as our focus of study not a single history, but a set of such histories represented as a structure branching from the "past" into the "future" (cf. [13]). Sometimes the non-deterministic branching of histories is taken to reflect agents' choice (see, for example, [7]), although this last idea seems to give an inappropriately privileged status to agents. Development of these various notions of time is often by way of precise sets of axioms that capture the elements of time (typically *moments*, *intervals* or *events*) and the relationships between them (see, for example, [5]).

1.2 World Histories and the Agents Within Them

A particular trajectory of an artificial world generated, for example, by a particular "execution" of a testbed on a computer, may be called an *artificial world history* (sometimes called a *chronicle*). Such a history may be formalised as a fixed sequence of *world states*, each represented by a set of assertions that specify

[1] But, of course, belief in some kind of foreknowledge has always been widespread in non-scientific thought.

its contents. A formal approach to the generation of such histories, using a version of temporal logic, has been followed in, for example, Concurrent Metatem [3].

In world histories agents are definable and recognisable as localised combinations of world state properties that persist along all or part of the state sequence and which conform to standard notions of an agent. Notice that in this formulation the agents are an integral part of the world history, not external to it. That part of a world state that corresponds to a particular agent will be, in effect, a "snapshot" of the agent's physical and cognitive processes (compare [13]).

The detailed definition of an agent may, of course, take a variety of forms and may or may not exclude, for example, agents that overlap or have physically separated parts. In general deliberative agents will function in terms of beliefs, desires and intentions, with all the well-known complexities that follow. A *rational* agent will act having due regard to its goals and to what it knows and believes about the world. One particular type of rational agent uses predictive planning, that is, it predicts alternative futures, selects a future which it judges attainable and compatible with its goals and tries to bring that particular future about – striking a balance between conflicting objectives as may be appropriate.

1.3 Agents with Choice in an Artificial World

At this point we may pose the preliminary question: *can agents in an artificial world have choice?* At first sight the answer might seem straightforwardly to be "no" since a digital computer is a deterministic machine. But we answer on the contrary, "yes", provided that the meaning of "choice" is that agents generate representations of alternative courses of action, and exhibit processes that select and act upon one of them. The key observation is that although these internal agent processes are part of the world history and are therefore themselves determined, nevertheless they are still recognisably processes of choice. If the world history is conceptually not fully deterministic (so that a particular history is appropriately regarded as a single branch of a history tree) this need not alter our conclusion.

2 World Histories as Solutions to CSPs

For reasons that will shortly become clear, the generation of artificial world histories, embodying artificial societies, may insightfully be viewed as a process of finding solutions to a constraint satisfaction problem [12]. The problem variables and their domains define the possible sequence of world states and the problem constraints specify the allowable structures (including agents' cognitive structures) within world states, and specify also the allowable changes from one world state to the next. It is natural to call these constraints *world constraints*. Some further constraints are in the nature of boundary conditions, for example, specifying the content of the first or the last state in the sequence. As we add extra world constraints, for example as we specify a certain more restricted type of agent locomotion or of agent cognitive processing, then so the range of solutions, that is, of possible alternative histories, is reduced. Ultimately if the set

of world constraints is sufficiently restrictive then no world histories consistent with the constraints may be possible. The situation will be clarified by a simple example.

2.1 An Example – The Gladiators Scenario

Consider the following very simple artificial society, in which "gladiators" "fight" and "die" on a large but finite square grid.

The gladiators have attributes but no internal structure, and are initially located arbitrarily in cells of the grid.
Repeatedly they move arbitrarily, independently and in synchrony into adjacent cells (horizontally, vertically or diagonally adjacent).
The available amount of time, equivalent to the number of moves each gladiator may make, is finite.
Whenever two (or more) gladiators arrive in the same square, then they both (or all) "fight and die" and cease moving.

What histories are possible for this society? Clearly there are a very large, but finite, number of combinations of paths and collisions that the gladiators may follow. If the gladiators are further restricted to move only into a horizontally or vertical adjacent grid cell, then the number of possible histories is clearly reduced.

In the Appendix a precise instance of this scenario is precisely formulated as a constraint satisfaction problem, and a possible history for the gladiators is found using standard techniques. We shall consider a more complex and interesting version of the scenario shortly.

3 Agents with Foreknowledge

As stated earlier, *foreknowledge* is the direct acquisition by an agent of knowledge of future facts or events. It is analogous to a form of perception. Foreknowledge may well be incomplete but by definition is never in error. For foreknowledge to be possible at all there is, seemingly, a key requirement. The relevant aspects of the future must, in some way, exist "at the same time as" the present – though it is apparent that we are now using some notion of meta-time. That in turn means that the world history in question cannot merely be generated sequentially from early to late. This is why it is appropriate to regard a world history from a CSP standpoint.

The first question we posed at the outset of this discussion was this: *Is it possible for agents in an artificial world to have foreknowledge?* It is. All that is required is that additional constraints be imposed on the world histories which enforce that agents, to some desired degree, *either* act in accordance with aspects of the future (in the case of reactive agents) *or* have and appropriately use internal representations that do indeed reflect the future (in the case of deliberative agents). In the following section the former of these two alternatives

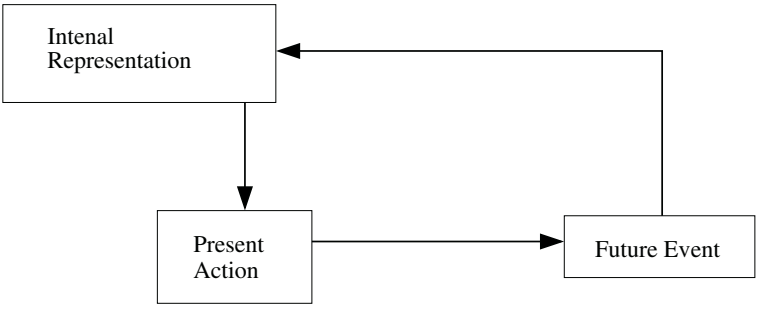


Fig. 1. The inter-relationship between a future event, an agent's present internal representation of it, and an action performed by the agent in the light of that representation. The direction of the arrows indicates (a type of) causality.

is illustrated by reference again to the Gladiators Scenario. The latter alternative, in which an agent's action is decided by reference to internal representations, is illustrated in Figure 1.

3.1 Gladiators with Foreknowledge

Consider again the Gladiators Scenario. Suppose that the gladiators must begin to "salute"[2] some fixed period of time *before* they die. How can this possibly be achieved? They require foreknowledge of their deaths if they are to start saluting at the correct time!

The formulation of the Gladiators Scenario given in the Appendix includes constraints (6 & 8) that impose just this foreknowledge. It is perhaps worthy of special note that these foreknowledge constraints *act at a distance in time*. A solution to the scenario, obtained using a general CSP solution algorithm, is also given in the Appendix.

The reader should note that the order in which the variables are labelled as this CSP formulation of the full Gladiators Scenario is solved, that is, the order in which the variables are assigned values consistent with the constraints, does not, indeed cannot, correspond to the temporal order of the world states which the variables express. This observation is significant because it contradicts our natural assumption that an artificial history is generated from past to future.

3.2 Rational Agents with Foreknowledge and Choice

Our second question was: *Can agents with foreknowledge in an artificial world have choice?* As we have interpreted the question, the answer is clearly "yes". All that foreknowledge does is to constrain a rational agent's predictions of the future to incorporate what is foreknown to occur. It does not negate the process of choice itself. In human terms, if I *know* that at 3 p.m. tomorrow I shall be drinking a cup of coffee in my office in Colchester, that still permits choice of

[2] Ave Caesar! Morituri te salutant!

action before and after the fixed event. But what, it will be asked, If I choose to defy my foreknowledge – if, for example, I immediately head for the airport and a plane to Beijing? The answer is clear. Firstly, I am being irrational – the whole point about foreknowledge is that I *know* what is going to happen. It is certain. Secondly, somehow or other I shall still end up drinking coffee in my office at 3pm tomorrow. Foreknowledge guarantees that. However, in the extreme case that the extent of the foreknowledge is *total*, the agent then has no choice simply because it can rationally consider no future other than the one that it knows actually occurs.

4 Foreknowledge and Evolutionary Advantage

The preceding "gladiators" example was very simple. In particular, the agents had no cognitive structure. We now describe a somewhat more complex scenario involving agents with foreknowledge, which enables us to address a restricted form of our final question: *Is foreknowledge beneficial or detrimental to an artificial society of (rational) agents?* It is easy to accept that foreknowledge is beneficial in some circumstances at least to the individual involved. It seems beneficial that an agent can, for example, identify the location of buried "treasure" that would otherwise not be detectable, by foreknowledge of the actual excavation of the treasure in a particular location. On the other hand, it is equally plausible that foreknowledge is the enemy of action and therefore of achievement. What is the point of action, if all is foreknown and therefore predetermined? In particular, it is intuitively plausible that foreknowledge of one's own death may (rationally) lead to selfish inactivity, to the detriment of society as a whole[3]. It is therefore interesting, and potentially informative, to try to examine more closely the social impact of foreknowledge by agents of their own deaths.

4.1 SCENARIO-3F

The software testbed SCENARIO-3F (a development of SCENARIO-3 [2]) has been written in the programming language C. It supports in simulation:

- *a two dimensional spatial environment*
- *mobile agents, immobile resources and immobile hazards*
- *spatially limited agent perception of their surroundings*
- *agent internal representations of resources and hazards, with adjustable memory limitation and forgetting*
- *agents competitively moving towards and harvesting resources for energy*
- *hazards killing agents, and agents destroying hazards*
- *agent death by starvation or by ageing*
- *(asexual) reproduction amongst agents with offspring variation*

[3] There is, for example, a well-known Irish folk tale to this effect [9].

Most movements and events are partly indeterminate. For example, when an agent attempts to move to a particular locality, or to kill a hazard, there are chance factors that may intervene and bring about an "unintended" outcome or movement.

Of course, words such as "harvesting", "killing" and "energy" are used here to aid intuitive understanding. They denote relatively simple events and quantities within the testbed. For example, "harvesting" is said to occur when an agent located at a resource reduces the "energy" level of the resource to zero, and increments its own internal energy store by a corresponding amount. Energy is used by an agent (reducing its energy store) it moves around in the testbed. "Killing" of an agent by a hazard occurs when an agent moves close to a hazard and, by chance, is killed by it. The "dead" agent is then deleted from the world. An agent "maintains a representation" of another entity (a hazard or a resource or even another agent) when it holds information (not necessarily accurately) about the entity.

The passing of time is simulated within the testbed as a sequence of "times", within each of which events at different locations (e.g. the movements of agents) take place simultaneously.

Many important SCENARIO-3F system parameters are under the control of the experimenter. These include the perceptual range of the agents, their rate of movement, the range at which hazards are dangerous, the range over which agents can communicate one with another, and their memory capacity. On the other hand, many potentially complex sequences of sub-events, for example those that in principle could mediate each particular instance of agent reproduction, are bypassed by an appeal to simulated chance.

4.2 Foreknowledge and Rationality in SCENARIO-3F

At the heart of the SCENARIO-3F formulation is that the "death" of a particular agent as the result of an attack by a particular hazard may be foreknown by the agent concerned and therefore that the rational behaviour of an agent may be made to depend upon when and how it will die. Just how far in advance of the event such knowledge is available to an agent is determined by the parameter *fspan* that varies from one agent to another.

Agents in SCENARIO-3F have the individual (and entirely selfish) goals of staying alive (by avoiding hazards) and of trying to acquire energy by harvesting resources. At any particular moment an agent has a choice between attempting to harvest some resource or attacking and trying to kill some hazard. Either choice may be the rational one on a particular occasion. But *agents that know when they are to die are assumed never to choose to attack hazards*, for it is only rational to expend time and energy to attack hazards when they pose a threat that can possibly be varied. A foreknown death is fixed and unavoidable, so that the hazards in the world then become irrelevant. This is where foreknowledge has its effect in this scenario.

4.3 Implementation of Foreknowledge in SCENARIO-3F

In principle the SCENARIO-3F scenario could be formulated and solved explicitly as a constraint satisfaction problem as was the Gladiators Scenario. However, this is too complex to be tractable. An *ad hoc* process has therefore been implemented within the SCENARIO-3F testbed which may be viewed as a crude but effective constraint satisfaction algorithm tailored to this particular problem.

If an agent is to be killed by a hazard at a certain time, both the agent and the hazard must be protected until that time. This requirement includes, for example, ensuring that the agent does not starve. In order to achieve this a "core-history" of foreseeable events is generated in advance of the full history. There is then a process, called *intervention*, whereby the inherent uncertainty in the events that occur (e.g. the chance component in agent movements and deaths) is exploited to ensure that the constraints imposed by foreknowledge are met. The effect of intervention is that the only histories that satisfy the requirements of foreknowledge are generated. As a full history is generated, the total number of specific interventions required is counted yielding an *intervention score* for that particular history.

A consequence of intervention is that the probabilities that are part of the specification of the history generation process (e.g. that when an agent moves at random, each direction of movement is to have the same chance of being chosen) are modified. This "distortion" is acceptable because our focus is strictly on the properties of the set of all those histories that satisfy the relevant foreknowledge constraints, rather than with the properties of some arbitrary history generation process.

4.4 The Experimental Objective

As stated, the aim in using the SCENARIO-3F testbed is to explore the impact of foreknowledge of "death" in a society of agents. Do agents with foreknowledge of their own deaths have an evolutionary advantage over those without it? Or are they at a disadvantage? The natural way to seek to answer these questions is to conduct systematic experiments in which a mixed population of agents, equipped with varying degrees of foreknowledge of their deaths, are allowed to compete for the available resources, and to reproduce through many generations, so that their differing survival abilities may be directly observed and measured. Of course, we expect the effectiveness and survival of agents to depend both upon the degree of foreknowledge which they possess and also upon the particular characteristics of the environment in which they exist.

4.5 A Problem of Measurement

There is a problem that at first sight seems paradoxical. How can we measure the impact on agent survival of foreknowledge when agent survival is predetermined, as it seemingly must be if agents are to foreknow their death?

The way out of this dilemma is a little intricate. First of all we simplify by identifying long-term agent effectiveness with individual life expectancy. We then ask not "What is the life expectancy of agents with and without foreknowledge?" but rather *"How frequently do histories with particular type of relationship between agent life expectancy and degree of foreknowledge (of death) appear within the set of all consistent histories?"* We ask this question because we may reasonably conjecture that there will be significant differences in the frequencies with which histories with particular characteristics occur in the entire set of consistent histories, reflecting the impact of those characteristics on the probability with which a history with them will actually occur. Note that the set of all possible consistent histories may be taken to be well defined and finite (if very large).

Unfortunately there seems to be no straightforward way to draw a random sample from the set of all possible consistent histories. However, a measure of the frequency with which a certain type of history will occur is the amount of intervention needed to obtain examples of it. Thus we may proceed by recording the intervention score needed to achieve consistent histories with particular combinations of foreknowledge and agent life expectancy.

4.6 Experiments and Results

It has proved possible to demonstrate coherent use of foreknowledge in a Scenario-3F agent society involving tens of agents and hazards existing over hundreds of cycles. However, as with all experiments with artificial societies, many system parameters influence the detailed behaviour of the system.

A series of experiments has been conducted to explore the relationship between agent foreknowledge and agent life span, using the following experimental

Table 1. The more important SCENARIO-3F parameter settings used in the experiments described in the text

environment size	100 x 100
number of resources	50
initial number of hazards	50
initial number of agents	25
maximum number of agents	25
number of times in the history	200
agent age limit	50
initial population mean for fspan	20
fspan drift	3
probability agent predestined to die	0.5
At each time:	
reproduction probability	0.2
probability agent attacks hazard	0.1
probability hazard kills agent	0.3
probability move goes astray	0.1

design and parameter settings. Three populations of agents reproducing through time were studied, each with an agent's value of *fspan* (the parameter that determines how far ahead an agent can see its death) randomly "drifting" from parent to offspring. Of the three populations, one had an imposed *positive relationship* between the parameter *fspan* and its actual life span, so that the greater *fspan*, the greater the life span, one had *no relationship*, and one had an imposed *negative relationship* (the greater *fspan*, the smaller the life span). For each of these three populations five histories were generated differing only in (pseudo-)random decisions. Important parameter settings are listed in Table 1.

The results, presented in Table 2, indicate that a relatively high level of intervention is needed, given the assumptions made, to construct world histories in which agents who have foreknowledge of their deaths well in advance come to predominate. The implication is that for these particular parameter settings foreknowledge is an evolutionary liability.

The interpretation of these results is that the more foreknowledge the agents have of their deaths, the more hazards survive on average until the end of the history (because more foreknowledge implies fewer attacks by agents on hazards) and this is "bad" for the population as a whole. Of course, this outcome depends strongly, not only on the notion of rational behaviour embodied in an agent, but also on the parameters of the created world, notably the degree of threat posed by a typical hazard. If hazards are not very dangerous at all, then relatively little intervention is needed to protect agents against them, and hence world histories in which most agents have foreknowledge and therefore do not attack hazards (and thus it is unnecessary to protect hazards against agents!) need relatively little intervention to create and are correspondingly relatively well represented in the set of all world histories. The results presented in Table 3 illustrate this point.

Table 2. The results of the SCENARIO-3F experiments. Each column shows means over five histories of an evolving agent population with a specific relationship, imposed between *fspan*, extent of agent foreknowledge, and agent life span (an * indicates that one history out of five was excluded from the mean calculations because one population became extinct). The histories differ only in the pseudo-random number stream. The first row indicates the degree of foreknowledge that the population ends with, the second the number of hazards surviving at the end of the history, and the third the amount of intervention required for that history. The greater the intervention the less, intuitively, is the probability of that history occurring by chance. Thus we see foreknowledge tending to reduce life expectancy.

	positive relationship*	**no relationship**	**negative relationship***
final mean over population of fspan parameter	18.2	9.1	1.8
number of hazards finally surviving	50.0	24.8	14.5
total intervention score	18034	14218	9979

Table 3. Intervention scores showing the effect of reducing the threat posed by hazards from a probability of 0.3 (figures reproduced from Table 1) to a probability of 0.01 (mean of three histories in each cell). The much lower intervention scores for a probability of 0.01 reflect the much-reduced need for intervention to preserve agents from premature death. The degree of foreknowledge available to agents, and hence the frequency with which they attack hazards, is now not significant.

probability hazard kills agent	**positive relationship**	**no relationship**	**negative relationship**
0.3	18034	14218	9979
0.01	256	239	267

We can now offer the beginnings of an answer to the third question that we posed at the outset: *Is foreknowledge beneficial or detrimental to an artificial society of (rational) agents?* This little set of results suggests that our question is coherent, but that its answer depends greatly upon the detailed properties of the agent society and of its environment.

5 Discussion

The experiments reported here have something in common with the thought experiments found widely in the literature of the philosophy of time. In these, certain ideas and assumptions are adopted and their consequences explored in detail, and in consequence new questions are provoked. Highly relevant examples appear in [11, chapter 4], and in [8].

Postulating that agents have foreknowledge seems to require that the future co-exists, in some sense, with the present. This requirement leads naturally to the consideration of block-time and to regarding world histories as entities that may be created, manipulated and destroyed as wholes. This in turn leads to a notion of what may be called *meta-time*[4]: the time within which a history exists and changes and which is quite distinct from the time which is one dimension of the history. However, foreknowledge with its associated constraints, rules out generation of a block-time world history systematically from past to future as our subjective experience suggests. This is why constraint satisfaction problem solving in which a world history is created by a process that exists unambiguously in meta-time seems as relevant as, for example, temporal logics in the AI tradition e.g. [3, 5].

It is a testament to the wide range of phenomena that can be captured and studied within an artificial society on a computer that it is even possible to explore the evolutionary advantage or otherwise of foreknowledge. The limited amount of experimentation reported here suggests, unfortunately, that there are no simple generalisations to be had. Furthermore, there are complex issues, for example around the concept of probability in a world history that is created other than in the "past to future" way we find intuitive. Is the probability of an

[4] Meta-time, in this sense, is an old notion, going back at least as far as Boethius.

event how likely it is to occur as an outcome of a present event (think of tossing a coin)? Or is it rather a matter of how frequently that event follows certain specified conditions when observed over an entire world history? In the former perspective there is a notion of a non-deterministic generative process that is building the history, in the latter there is not.

Finally, we should note a difficulty with foreknowledge, related to the previous remark that concerns the somewhat elusive idea of causality. It is a commonplace that at the heart of foreknowledge is *reverse causality* (and causality at a distance). A cause at one time (e.g. the death of an agent) has an effect at an earlier time (the formation within that agent of an appropriate representation of, or belief, about the death). But it seems a strange and arbitrary restriction to envisage backward causation whose effect is *only* upon agent cognition. This seems to require a dualist view of the world, and to be decidedly anthropocentric. We suggest that foreknowledge is a flawed concept, not because there is little compelling evidence for it in reality, but because it gives a very special status (a locus of reverse causality) to just one type of complexity in world histories, those that we are prepared to label "agents".

6 Conclusions

The work reported here has demonstrated that:

- it is possible to implement without incoherence a simple form of agent foreknowledge in artificial societies in artificial worlds
- foreknowledge naturally leads to a view of space-time based upon constraint satisfaction problem solving
- it is possible, if technically complex, to explore on a computer the exact circumstances in which foreknowledge does or does not confer evolutionary advantage upon a society of agents.

We do not suggest that the investigation reported here has practical significance. We have deployed foreknowledge merely as a means to explore certain theoretical issues around time and rationality. Yet it is surely a tantalizing thought that if humanity has ever had a capacity for some degree of foreknowledge, there might have been good reason for it to be evolved away!

References

1. Doran, J.: Foreknowledge in Artificial Societies. In: Conte, R., Hegselmann, R., Terna, P. (eds.): Simulating Social Phenomena. LNEMS 456. Springer-Verlag, Berlin (1997) 457-469
2. Doran, J. E.: Simulating Collective Misbelief. Journal of Artificial Societies and Social Simulation, Vol. 1(1) <http://www.soc.surrey.ac.uk/JASSS/1/1/3.html> (1998)
3. Fisher, M.: A Survey of Concurrent METATEM: the Language and its Applications. In: Gabbay, D. M., Ohlbach, H. J. (eds.): Temporal Logic. LNAI 827, Springer-Verlag, Berlin (1994) 480-505

4. Horwich, P.: Closed Causal Chains. In: Savitt(1995)
5. Knight, B., Ma, J.: Time Representation: a Taxonomy of Temporal Models. Artificial Intelligence Review 7 (6) (1994) 401-419
6. Le Poidevin, R., MacBeath, M. (eds.): The Philosophy of Time. Oxford Readings in Philosophy. Oxford University Press, Oxford (1993)
7. McDermott, D. V.: A Temporal Logic for Reasoning about Processes and Plans. Cog. Sci. 6: (1982) 101-155
8. Newton-Smith, W. H.: The Structure of Time. Routledge and Kegan Paul, London (1980)
9. O'Sullivan, S. (ed. & trans.): Folktales of Ireland. University of Chicago Press, Chicago (1966)
10. Savitt, S. F. (ed.): Time's Arrows Today. Cambridge University Press, Cambridge, England (1995)
11. Schlesinger, G. N.: Aspects of Time. Hackett Publishing Company, Indianapolis (1980)
12. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London (1993)
13. Wooldridge, M.: Time, Knowledge and Choice (preliminary report) In: Wooldridge, M., Muller, J., Tambe, M. (eds.): Intelligent Agents II: Agent Theories, Architectures, and Languages. LNAI 1037. Springer-Verlag, Berlin (1996)

Appendix

A Formulation of the Gladiators Scenario

The following is a formulation of the Gladiators Scenario as a constraint satisfaction problem.

```
INDICES
      i and j index gladiators (just 3 here)
      p and q are the coordinates of a gladiator in the grid
      k indexes world states --- therefore, in effect, time
      a indicates whether or not a gladiator is alive
      s indicates whether or not a gladiator is saluting

      where
            1 <= i,j <= 3
            0 <= k <= 99

VARIABLES

      G[i,k].p
      G[i,k].q
      G[i,k].a
      G[i,k].s

Thus there are 1200 variables in all.
```

```
DOMAINS

      The variables G[i,k].p and G[i,k].q can take any
      integer in [-100, 100]

      The variables G[i,k].a and G[i,k].s can take only
      the values t or f


CONSTRAINTS
```

The following sets of constraints apply where the above specified ranges for indices make them well defined. A constraint in the form IF A THEN B declares to be invalid any set of variable value assignments which satisfies A but not B. Any set of variable value assignments which does not satisfy A does satisfy the constraint.

The constraints specify the allowable movements of the gladiators, the persistence of their properties, and the circumstances in which they die and salute. In particular, constraints 6 and 8 express the gladiators' foreknowledge.

```
1.    IF (G[i,k].p = G[j,k].p)
      AND (G[i,k].q = G[j,k].q)
      THEN G[i,k].a = f AND G[j,k].a = f     ( i =/= j )
2.    IF G[i,k].a = t THEN G[i,k].p = G[i,k+1].p +- 1 AND
      G[i,k].q = G[i,k+1].q +- 1
            {where +- means either + or - is allowable}
3.    IF G[i,k].a = f THEN G[i,k].p = G[i,k+1].p AND
      G[i,k].q = G[i,k+1].q
4.    IF G[i,k].a = f THEN G[i,k+1].a = f
5.    IF G[i,k].s = t THEN G[i,k+1].s = t
6.    IF G[i,k+3].a = f THEN G[i,k].s = t
7.    IF G[i,k].a = t AND NOT (( G[i,k+1].p = G[j,k+1].p)
      AND ( G[i,k+1].q = G[j,k+1].q ))
      THEN G[i,k+1].a = t
8.    IF G[i,k].s = f AND NOT G[i,k+4].a = f
      THEN G[i,k+1].s = f


BOUNDARY CONDITION

      G[1,0].s = G[2,0].s = G[3,0].s = f
```

A Solution to the Gladiators Scenario

In principle the Gladiators Scenario as formulated above may be solved by any CSP algorithm. The following simple solution was obtained (with the kind assistance of James Borrett) using an AC-lookahead algorithm [12, p. 133].

	Gladiator 1	Gladiator 2	Gladiator 3
	(p q a s)	(p q a s)	(p q a s)
World State 0	-99, -99, t, f	-99, -97, t, f	-97, -99, t, f
World State 1	-100, -100, t, t	-100, -98, t, t	-98, -100, t, t
World State 2	-99, -99, t, t	-99, -97, t, t	-97, -99, t, t
World State 3	-100, -100, t, t	-100, -98, t, t	-98, -100, t, t
World State 4	-99, -99, f, t	-99, -99, f, t	-99, -99, f, t

with thereafter all three gladiators immobile, dead and forever saluting.

Self-organisation in Holonic Multiagent Systems

Klaus Fischer

DFKI GmbH,
Stuhlsatzenhausweg 3,
D–66123 Saarbrücken
Klaus.Fischer@dfki.de
http://www.dfki.de/~kuf

Abstract. With the ever growing usage of the world-wide ICT networks, agent technologies and multiagent systems (MAS) are attracting more and more attention. Multiagent systems are designed to be open systems. Therefore, agent technologies aim at the design of agents that perform well in environments that are not necessarily well-structured and benevolent. Looking at the problem solving capacity of MAS, emergent system behaviour is one of the most interesting phenomena one can investigate. However, there is more to MAS design than the interaction between a number of agents. For an effective system behaviour we need structure and organisation. To specify the organisation of a MAS at design time turns out to be a difficult task. If the structure of the MAS needs to adapt to changes in the environment it can turn out to virtually impractical. This paper presents basic concepts of a theory for holonic multiagent systems with the aim to define the building blocks of a theory that can explain organisation and dynamic reorganisation in MAS. In doing so it tries to contribute to solving the well-known micro-macro gap in MAS theories. The applicability of the basic concepts are illustrated with three application scenarios: flexible manufacturing, order dispatching in haulage companies, and train coupling and sharing.

1 Introduction

The increasing importance of the world-wide telecommunication and computer networks, especially the Internet and the World Wide Web (WWW) is one of the reasons why agent technologies have attracted so much attention in the past few years. Although in many of todays applications individual agents are trying to fulfil a task on behalf of a single user, these agents are doing so in a multiagent context. It is obvious that the problem solving capabilities of multiagent systems (MAS), which have been developed since research on MAS began in the late 70s, will become more and more important. However, the implementation of MAS for interesting real-world application scenarios tend to be very complex. The basic approach to tackling this complexity is to base problem solving on emerging bottom-up behaviour. This is achieved by giving the agents specific abilities which leads to emergent problem solving behaviours when the agents interact with each other. Although this approach works well in many cases, the solutions tend to be sub-optimal.

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 543–563, 2005.

With his Watchmaker's parable Simon demonstrated that a hierarchy offers another useful paradigm for tackling complexity [18]. The hierarchical solution to a problem is built up from modules which form stable sub-solutions, allowing one to construct a complex system out of less complex components. Control in such a hierarchy can be designed in a centralised or a decentralised way. The decentralised model offers robustness and agility with respect to uncertainties in task execution. The major advantages of the introduction of centralised planning and control instances are predictability, opportunities for performance optimisation, and an easier migration path from current to distributed systems [4].

To design and implement systems that include both hierarchical organisational structures as well as decentralised control the concepts of *fractal* and *holonic* design were proposed [21, 7]. The word *holon* [14] is derived from the Greek *holos* (whole) and the suffix *on*, which means particle or part. A holon is a natural or artificial structure that is stable, coherent, and consists of several holons as substructures. No natural structure is either *whole* or *part* in an absolute sense. A holon is a complex whole that consists of substructures as well as it is a part of a larger entity. In both approaches, in fractal as well as holonic design, we have the ideas of recursively nested self-similar structures which dynamically adapt themselves to achieve the design goals of the system. We adopt the notion of *holonic multiagent systems* to transfer these ideas to MAS design. In a holonic MAS autonomous agents group together to form holons. However, in doing so they do not loose their autonomy completely. The agents can leave a holon again and act autonomously or rearrange themselves as new holons. According to this view a holonic agent consists of sub-agents, which can separate and rearrange themselves and which may themselves be holons.

The paper starts with a general formalism to describe MAS in Section 2, introduces the architecture InteRRaP for the specification of individual agents in Section 3, and gives a formal specification of holonic MAS in Section 4. The usefulness of the proposed concepts is then demonstrated by three application scenarios: (1) flexible manufacturing, (2) order dispatching in haulage companies, and (3) train coupling and sharing. These application scenarios differ in the sense that in (1) holons are formed dynamically because agents with different abilities have to work together to achieve a common goal. In (2) the abilities of the agents partly overlap. Finally, in (3) all agents have the same abilities.

2 Abstract Specification of Multiagent Systems

As it is the case with any software system, when designing a system, we can distinguish the static specification of the system and its runtime instance. While in present days concepts and theories for the static specification of software systems is reasonably well-understood, concepts and theories for the specification and analysis of the dynamic behaviour of a software system are by for not that sophisticated. This is especially true if we look at MAS, in which self-organisation is an important aspects. This makes MAS different form software systems that are designed according to a more traditional development paradigm.

To allow self-organisation in a MAS, we assume that some infrastructure which supports the agents in the process of self-organisation is available. The

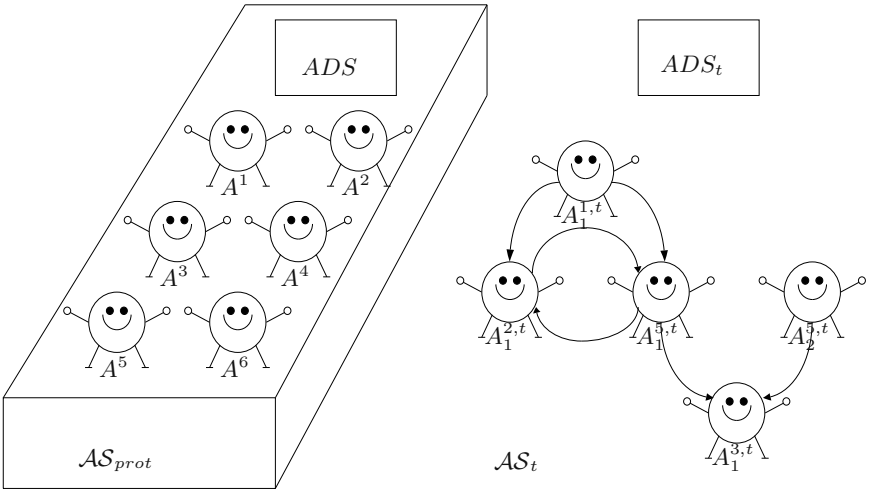


Fig. 1. Static specification $\mathcal{AS}_{prot}$ of the solution for a problem and dynamic execution $\mathcal{AS}_t$.

FIPA[1] initiative tries to establish standards for such infrastructure and for MAS in general in an open environment. This paper takes a more abstract point of view, which assumes that there is an agent directory service (ADS) which allows the agents to find out how they can contact other agents that are available in the MAS. This means that we require that the ADS provides at minimum a white pages service, e.g. agents can ask about addresses of other agents. Other services like yellow pages, i.e. the information on services offered by specific agents, might be also provided by the ADS. However, this and possibly other general services might also be introduced by other specialised agents of the MAS.

To describe a concrete MAS for a given application domain, a set of prototypical agents is specified. This static description of the MAS is given by the prototype agent system $\mathcal{AS}_{prot} := (\mathcal{AS}_{prot}, ADS)$, where

$\mathcal{A}_{prot}$ is the set $\{A^1, \ldots A^n\}, n \in \mathbb{N}$ of prototypical agents, instances of which can dynamically be introduced in the system. These agents are the potentially available problem solvers. Several instances of a specific prototypical agent can be created.

ADS is a specialised prototypical agent providing an agent directory service.

With modern concepts like component-based software development this view to a MAS might already look out-dated. We cannot really assume that the prototypical descriptions of the agent types are self-contained, they are rather created from a set of components. However, conceptually we can still assume that there is a finite set of agent types available, instances of which can be dynamically introduced into the MAS that actually executes (i.e. works on a specific problem) in a given application domain.

[1] See http://www.fipa.org/

The process of problem solving starts with the initial agent system

$$\mathcal{AS}_{init} = (\mathcal{A}_{init}, ADS_{init}) \text{ where}$$

$$\mathcal{A}_{init} = (A_1^1, \ldots, A_{k_1}^1, \ldots A_1^n, \ldots, A_{k_n}^n), k_1, \ldots k_n \in \mathbb{N} \text{ and}$$

$$\forall i, j \in \mathbb{N} : A_j^i \in \mathcal{A}_{init} : A^i \rhd A_j^i \wedge A^i \in \mathcal{A}_{prot}.$$

$A \rhd A'$ denotes that A' is an instance of the prototypical agent A which means that A' inherits its behaviour and initial knowledge from A but gets also some additional knowledge like for example its unique identification which can be used as an address to communicate with A'.

Please note that the fact that ADS_{init} is explicitly introduced dose not necessarily mean that the MAS is is closed in the sense that the system engineer is in control of all parts of the system. We can assume that ADS_{init} represents some ADS, which is already available and which has the state of ADS_{init} at the point in time when the first agent of the part of the system that is under the control of the system engineer is started. Along the same line of reasoning we can assume that some of the agents in $\mathcal{A}_{init}$ were also not designed by the software engineer but represent agents that are avaliable in the open environment. Let us without loss of generality assume that $\mathcal{A}_{open} = \{A_1^1, \ldots A_{k_1}^1, \ldots A_1^m, \ldots A_{k_l}^m\}$ for some $1 \leq m < n$ represents the set of these agents. The specifications for the corresponding prototypical agents $A^1, \ldots A^m$ is likely to be not complete in the sense that the system engineer who designs $\mathcal{A}_{prot}$ only needs to have the information about $A^1, \ldots A^m$ which is actually needed for the rest of the agents in $\mathcal{A}_{prot}$ to use services that are offered by the former set of agents.

From $\mathcal{AS}_{init}$ the dynamic MAS $\mathcal{AS}_t$ evolves

$$\mathcal{AS}_t = (\mathcal{A}^t, ADS_t) \text{ where}$$

$$\mathcal{A}_t = (A_1^{1,t}, \ldots, A_{l_1}^{1,t}, \ldots A_1^{n,t}, \ldots, A_{l_n}^{n,t}), l_1, \ldots, l_n \in \mathbb{N} \text{ and}$$

$$\forall i, j \in \mathbb{N} : A_j^{i,t} \in \mathcal{A}_t : A^i \blacktriangleright A_j^{i,t} \wedge A^i \in \mathcal{A}_{prot}.$$

$\blacktriangleright$ denotes $\rhd \circ \rightsquigarrow^*$ which means that we have $A^i \rhd A_j^i$ where $A_j^i \in \mathcal{A}_{init}$ and $A_j^i \rightsquigarrow^* A_j^{i,t}$ where $\rightsquigarrow$ denotes the transformation of A_j^i by a single step of computation.

The computation goes on while the agents send messages to each other. New agents may be introduced and some of the active agents might be killed. Each of the agent has a unique identification, i.e. address, which can be used in messages, thus allows any desired organisation of the MAS to be achieved. All agents automatically know the the identification of the ADS agent. An agent can make its own identification accessible to all of the other agents by registering at the ADS agent. In the example of Fig. 1 the system started with $\mathcal{AS}_{init} = ((A_1^1, A_1^5), ADS_{init})$. A_1^1 creates A_1^2 and A_1^5 and A_2^5 creates A_1^3. At time t, A_1^1 told A_1^5 the identification of A_1^2 and A_1^5 told A_1^2 its own identification. A_1^5 got the identification of A_1^3 because A_1^3 registered its identification at the ADS agent and A_1^5 extracted this information from the ADS agent.

3 The Specification of Individual Agents

In agent-oriented computing the key definitional problem relates to the term agent. There is still an ongoing debate, and little consensus, about exactly what constitutes agenthood. However, an increasing number of researchers and industry practitioners find the following characterisation useful:

> An agent is an encapsulated computational system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [24].

In this definition flexible requires the agent to exhibit pro-active, reactive, and social behaviour. We can therefore define the key properties of agenthood as [23, 25]:

Autonomy: agents are clearly identifiable problem solving entities – with well-defined boundaries and interfaces – which have control both over their internal state and over their own behaviour

Reactivity: agents are situated (embedded) in a particular environment, i.e. they receive inputs related to the state of their environment through sensors. They then respond in a timely fashion, and act on the environment through effectors to satisfy their design objectives.

Pro-activeness: agents do not simply act in response to their environment, they are designed to fulfil a specific purpose, i.e. they have particular objectives (goals) to achieve. Agents are therefore able to exhibit goal-directed behaviour by taking the initiative and opportunistically adopting new goals.

Social Ability: agents are able to cooperate with humans and other agents in order to achieve their design objectives.

If we compare this definition with an object-oriented (OO) approach to the design of software systems it is safe to assume that almost all more recent implementations of MAS will be based on some OO programming environment like for example Java or C++. If we look at a single threaded OO program, autonomy in the sense defined above is clearly a property that we can ascribe to such a program (see Fig. 2 top left). If we put such a program into the context of a concurrent OO environment, e.g. CORBA [17], all objects in this environment are able to interact with each other and therefore display a limited form of social abilities (see Fig. 2 top right). However, if all of the objects are single threaded, we face a problem. Once an object starts to work on a request from an other object it is not able to react to further request of the same or other objects and therefore lacks the property of being reactive. To introduce reactiveness, each object needs the ability to at least do two activities concurrently: work on requests from other objects and react to further requests (see Fig. 2 bottom left). There are several ways to technically achieve this. Languages for concurrent OO programming introduced solutions for this at an early stage [26]. Still if we have objects that are able to act on two concurrent activities we need yet another thread of activity to introduce pro-activeness. The three activities (being reactive, pro-active, and interacting) can be directly deduced

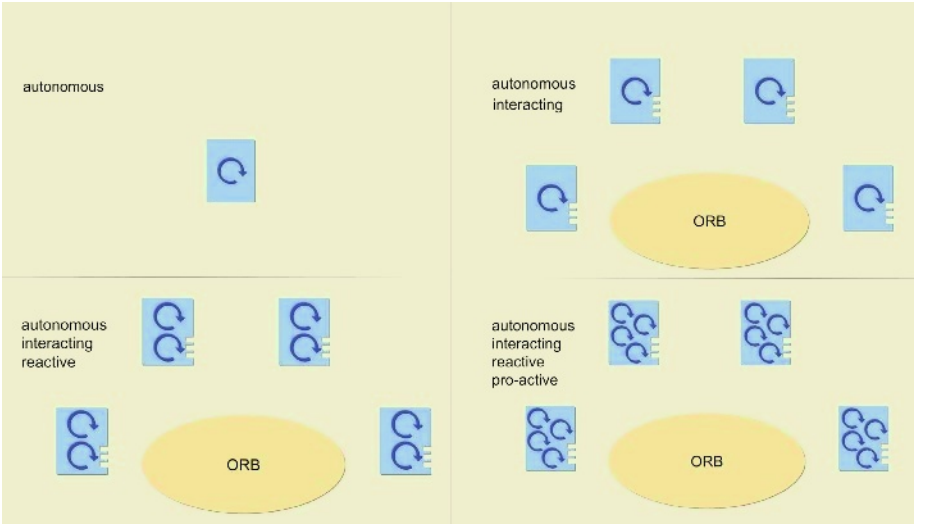


Fig. 2. Agent Architectures Introduce Structure into Concurrent Object-Oriented Software Design.

form the definition above. However, every interaction with another agent/object can give reason to introduce a new concurrent activity (see Fig. 2 bottom right). How the scheduling between these activities is actually implemented (one thread that concurrently works on all these activities or a true mutli-threaded computational model) is not that important. The important point to note is that on an abstract level agents are by definition multi-threaded objects that run in a concurrent OO environment. Unfortunately, concurrency is a computational construct that introduces complexity. To control concurrency in an individual agent we need structures that are well-understood, like it was the case in structured and OO programming. For agent-oriented programming these structures are agent architectures. A number of agent architectures have been suggested in MAS literature (for an overview see Chapter 1 in [22]). Our approach to the design of individual agents is the agent architecture InteRRaP (see Figure 3).

The main idea of InteRRaP [16] is to define an agent by a set of functional layers, linked by a communication-based control structure and a shared hierarchical knowledge base: (1) the co-operative planning layer (CPL); (2) the local planning layer (LPL); and (3) the behaviour-based layer (BBL).

The CPL contains mechanisms for devising joint plans and has access to protocols, a joint plan library, and knowledge about communication strategies. The LPL contains a planning mechanism, which is able to devise local single-agent plans. The plans are non-linear data structures the nodes of which can be either new sub-plans, executable patterns of behaviour, or primitive actions. Thus, the plan-based layer may activate patterns of behaviour in order to achieve certain goals. The BBL implements the basic behaviour of the agent using behavioural scripts called patterns of behaviour. The BBL is closely linked to the world in-

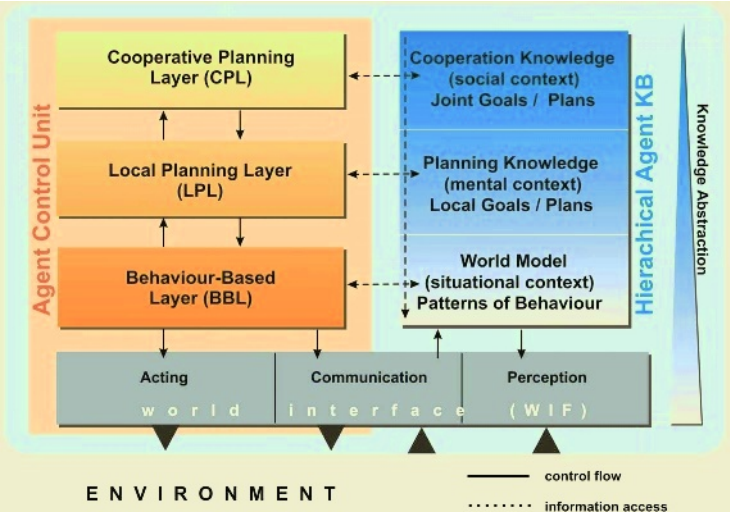


Fig. 3. The Agent Architecture InteRRaP.

terface, and thus, to the actions and changes in the world. Patterns of behaviour can be activated both by the plan-based layer (top-down activation) and by external trigger conditions (bottom-up activation).

Corresponding to the control layers of the agent, the agent's knowledge base consists also of three layers. The lowest layer, which belongs to the BBL, contains facts representing the world model of the agent. Layer two, which corresponds with the LPL, represents the mental model, which contains local goals and local plans. Finally, layer three on the CPL specifies the social model, which contains knowledge of and strategies for co-operation, e.g. beliefs about other agents' goals. The basic idea is that information is passed only from lower layers of the knowledge base to higher layers. For example, the plan-based layer can access information about the world model, whereas the BBL does not have access to planning or co-operation information.

4 The Multiagent System Level

From the very beginning of MAS research MAS were introduced as a new problem solving paradigm [2]. Rather than explicitly specifying at design time how a specific problem will be solved, the general idea in this thread of research is to achieve problem solving by the interaction of the individual agents in the agent society. The solutions to the given problem emerges from this interaction. Looking at nature, an ant hive is the most intuitive example to explain emergent problem solving behaviour (see Figure 4). It is not possible to explain the overall behaviour of an ant hive by looking at an individual ant, nor does the removal of even a significant part of the hive necessarily influence the overall behaviour. Though some parts of the hive seem to be more important than others. Although

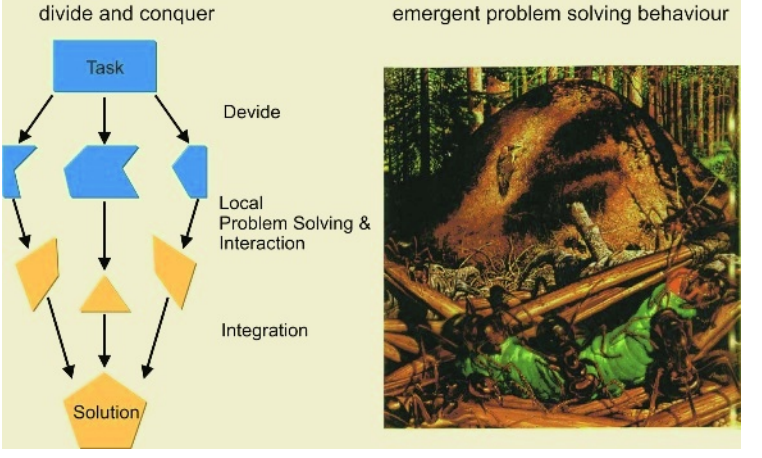


Fig. 4. Competing Problem Solving Paradigms.

interesting results have been presented using this approach to problem solving, emergent problem solving behaviour has also been criticised to provide inefficient or even undesirable results.

Divide and conquer (see Figure 4) is widely accepted problem solving paradigm of general computer science. Here a centralised problem solving entity takes a task, separates it into sub-tasks and distributes these sub-tasks to decentralised problem solvers. The problem solvers produce solutions for the sub-problems and send these solutions back to the centralised problem solving entity which integrates the solutions for the sub-problems into an overall solution for the original task. This approach to problem solving is much more structured than the pure emergent problem solving behaviour. The contract net protocol [20] which is a widely-accepted problem solving model in distributed AI research uses the divide and conquer model. The centralised problem solving entity, called the manager for the task, separates the overall task into sub-tasks. The manager uses a bidding procedure, first price sealed bid auction, to find the most appropriate decentralised problem solver for each of the sub-problems. The integration of the solution to the sub-problems, which are provided by the decentralised problem solvers, into an overall solution is again done by the manager. This procedure can be recursively nested, i.e. the decentralised problem solvers can again use the contract net model to find a set of further problem solvers who are able to solve the given sub-task.

In the contract net protocol as it was just described the concrete assignment of sub-tasks to problem solving entities is an emergent property. By allowing individual agents to find out on their own which part of the task they would like to work on and bid for this part of the task, the separation of the task into subtasks becomes an emergent phenomenon. In this model the manager announces the complete task to the problem solving agents. These agents evaluate the task and send back bids for parts of the task they would like to work on. The model works

best if the tasks can be separated in virtually any desired set of sub-tasks and these sub-tasks are uniform with respect to the problem solving abilities that are required for producing a solution. However, although we definitively have to assume a common understanding of dividing the overall tasks into sub-tasks among the problem solving agents, we can find a whole spectrum of problem solving models where the situation in which at one extreme the manager does the complete job of dividing the task into sub-tasks and at the other extreme the devision of the tasks into sub-tasks is done completely by the agents bidding for the task. The first case corresponds to a top-down approach to problem solving which can be very efficient but sometimes difficult to install in the first place and possibly even more difficult to adapt to a changing environment. The latter case corresponds to a bottom-up approach to problem solving which is flexible and provides the ability to adapt to changes. However, it also introduces complexity up to the degree of combinatorial explosion.

4.1 Holonic Multiagent Systems

As already discussed in Section 1 the concepts of *fractal* and *holonic* system design were proposed [21, 7] to combine both hierarchical organisational structures in a top-down manner as well as decentralised control, which takes the bottom-up perspective. Although it would be possible to organise the holonic structures in a completely decentralised manner, for efficiency reasons it is more effective to use an individual agent to represent a holon. In some cases it is possible to select one of the already present agents as the representative of the holon by using an election procedure. In other cases a new agent is explicitly introduced to represent the holon just for its lifetime. In both cases the representative agent has to have the ability to represent the shared intentions of the holon and to negotiate about these intentions with the agents in the holon's environment as well as with the agents internal to the holon.

When we pick up again the formal description of MAS of Section 2, we can at any given point in time identify holonic structures in the dynamic MAS $\mathcal{AS}_t$. A holon H behaves in its environment like any of the agents in $\mathcal{A}_t$. However, when we take a closer look it might turn out that H is built up by a set of holons itself. Let $\mathbf{atomic} : \mathcal{H} \rightarrow \mathbb{B}$ be a function that tells us whether a given holon H is built up of other holons or whether $H \in \mathcal{A}_t$. For the set of holons $\mathcal{H}$ we then have:

$$\forall H \in \mathcal{H} : \begin{cases} \mathbf{atomic}(H) : & H \in \mathcal{A}_t \\ \neg\mathbf{atomic}(H) : & \forall h_i \in H \Rightarrow h_i \in \mathcal{H} \end{cases} \tag{1}$$

Like any agent a holon has a unique identification so that it is possible to communicate with the holon by just sending messages to its address.

As it was already discussed in [14] we assume that the reason for the agents to form a holonic structure is that the agents which make up the holon have to co-operatively solve a set of tasks. This means that in our MAS the modelling of holonic agents is realised by commitments (see e.g. [19]) of the sub-agents to cooperate and work towards a common goal to execute a set of tasks in a

corporate way. Because of this common interest the entities form *shared goals* and *shared intentions* [13, 6]. We use the term *shared* because we do not want to go into the details of the discussion whether joint knowledge can or cannot be achieved in a MAS [9]. We assume that *shared* means that the agents are aware of the fact that other agents are involved and that these agents maintain knowledge, belief, goals, and intentions about the goals and intentions that are believed to be shared. More formally we can say that all sub-holons $h_i, i \in \mathbb{N}$ of a holon h commit themselves to co-operatively execute action a where each of the h_i performs action a_i and we assume that at any point in time t

$$\bigwedge_{h_i \in h} \mathbf{succeeded}(a_i, t) \rightarrow \mathbf{succeeded}(a, t) \tag{2}$$

holds. To execute action a is a shared intention of the sub-holons h_i of h. Because of (2) we have

$$\bigwedge_{h_i \in h} \mathbf{achieved}(\mathit{intent}(\mathit{execute}(a_i), t)) \rightarrow \mathbf{achieved}(\mathit{intent}(\mathit{execute}(a), t)) \tag{3}$$

What makes things difficult to further describe the holonic structures is that the tasks which have to be executed by the agents get known in the system some time before the point at which the execution starts. This implies that the agents have to make commitments on participating in a holon at a point in time at which the execution of the task lies in the future. This means that we can distinguish between the period in which the agents negotiate about in which holon they want to participate and the actual execution of the tasks. However, this distinction does not have any influence on the actual structure of the holon, since the holonic structure is formed when the commitments for the shared intentions are made. For the settings in this paper a new holon is formed as soon at least two agents commit themselves to co-operatively perform an action. The holonic structure remains at least until either the joint action is performed or the commitment to do the joint action is retracted.

To make things even more complicate, we can assume that costs are involved with the execution of tasks. On the other hand agents get some benefit for executing tasks. To give the agents an incentive to actually work on tasks we assume that they try to optimise their local situation with respect to a local performance profile, which depends on the set of tasks the agent is committed to and the sequence in time in which the tasks are executed. The agents can try to continously optimise their local situation by committing or de-committing to tasks and by changing the plan for the execution of the tasks, e.g. by negotiating about the deadline until the task has to be finished. Figure 5 gives an example of the situation that we face. In this example we have two agents and we see how a change in parameter X influences the local performance profile of agent 1 as well as the overall performance that is displayed in the global performance profile. In the given example the change in the parameter X increases the performance of both agents. From agent 2 point of view performance stays at the point of the local maximum but agent 2's performance is nevertheless increased by the change of the parameter X in agent 1.

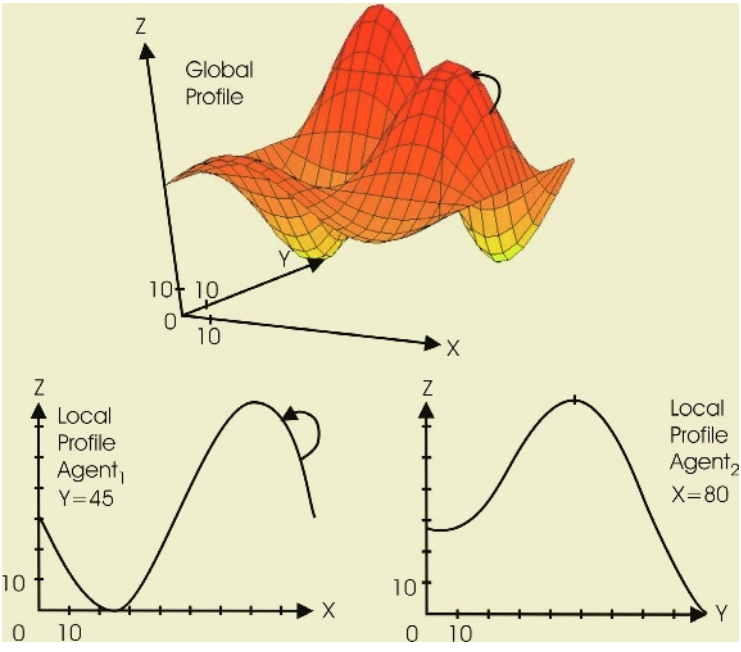


Fig. 5. Global and local performance profiles.

In how far the global performance profile is explicitly represented in an agent with a more centralised problem solving perspective depends on the concrete MAS model for a specific scenario in an application domain. In the next section we will see different models for decentralised and more centralised decision making. We can further classify different scenarios by investigating in how far agents accept changes in their local performance by proposals of other agents. At one extreme we can have a totally cooperative setting in which all agents accept any proposal as long as the performance of the whole agent society is increased. At the other extreme we can have a totally competitive setting in which any agent would never accept a proposal if the proposal effects the agent's local performance in a negative way. However, even in the latter setting there is a chance of increasing the overall system performance of the agent society by negotiation (cf. [10]).

5 Applications

In this section we illustrate the formation of holonic structures and shared intentions in three application scenarios: (1) flexible manufacturing, (2) order dispatching in haulage companies, and (3) train coupling and sharing. These application scenarios differ in the sense that in (1) holons are formed dynamically because agents with different abilities have to work together to achieve a common goal. In (2) the abilities of the agents partly overlap. Finally, in (3) all agents have the same abilities.

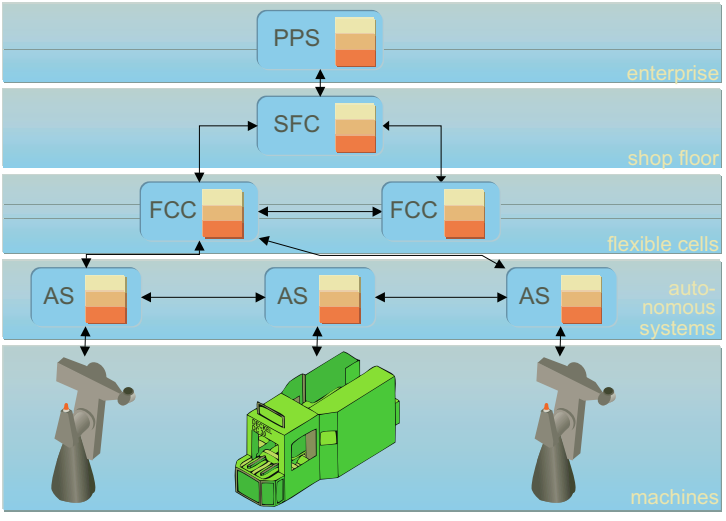


Fig. 6. Planning and control layers in a flexible manufacturing system.

5.1 Flexible Manufacturing Systems

There are already well-established layers of abstraction in the control of a flexible manufacturing system (FMS) (see Fig. 6): production planning and control (PPC), shop floor control (SFC), flexible cell control (FCC), autonomous system control, and machine control. Each of these layers has a clearly defined scope of competence. In Fig. 6 we can see holons on each of the five layers: at the lowest layer, the physical body of an autonomous system (i.e. an autonomous robot or a machine tool) together with its controlling agent. On the layer of the flexible cells we have the flexible cell control system together with the holons that are formed by the physical systems that belong to the flexible cell. On the SFC layer we have the agent that represents the SFC for a specific production unit together with all the holons that belong to it. Finally, on the enterprise layer we have all the holons that are present at a specific site of the company. We can even go further and represent holons of companies with several sites. However, in this paper we do not want to further elaborate on this. Most of the holonic structures which were just described are quite stable. However, especially on the layer of the flexible cells it is very important to have efficient mechanisms to dynamically form new holons. In describing this situation we have the conceptual problem that autonomous systems such as mobile robots might interact with flexible cells. To have a more uniform view we assume that pure autonomous systems such as mobile robots and autonomous (guided) vehicles are represented as holons on the FCC layer, too. For the rest of this section we refer to all these systems as flexible cell holons (FCH).

The SFC system passes tasks to the lower FCC layer as soon as it is determined by the production plan that a task can be executed because all of the preceding steps in the working plan have been successfully completed. From

these tasks the FCHs on the lower layers derive their shared and local intentions. The SFC system does not care whether it is possible for a group of FCHs to execute this task immediately or if they are currently engaged in the execution of a task. The SFC system just inserts the task into a list which is accessible to all of the FCHs and the FCHs decide by themselves when it will actually be executed. By executing a specific task, several FCHs have to co-operate. Each FCH has to play a part to solve a specific task. No FCH may believe that it is the only one that wants to play a certain part for a specific task. Therefore, the FCHs must co-ordinate their intentions to play parts in different tasks. The main problem to be solved is to find a consistent group of FCHs which together are able to solve a specific task. We call such a group a complete holon for a task. Only tasks for which a complete holon is formed can actually be executed.

The FCHs can be separated into three groups: $\mathcal{R}$ mobile manipulation systems (mobile robots), $\mathcal{T}$ transport systems, and $\mathcal{C}$ flexible cells such as machining centres that might have locally fixed robots. In some settings even the workpieces which are to be processed are able to move autonomously, for example when they are installed on a transportation device. In these settings it is reasonable to control these workpieces by FCHs. We therefore introduce the set of workpieces $\mathcal{W}$.

Mobile manipulation systems are able to work flexibly on the given tasks. Each time a mobile manipulation system finishes the execution of a task it can start working on any task it is able to regardless of its current configuration. Locally fixed robots, machining centres, and flexible cells are much more restricted in their ability to choose tasks to be executed than FCHs in $\mathcal{R} \cup \mathcal{T}$ because FCHs in $\mathcal{C}$ have a fixed location. An FCH f in $\mathcal{C}$ depends on FCHs of $\mathcal{R} \cup \mathcal{T}$ if all the devices needed for a specific task are not already present within f. We therefore introduce the precedence relations $\mathcal{W} \prec \mathcal{C} \prec \mathcal{T} \prec \mathcal{R}$. $\mathcal{T} \prec \mathcal{R}$ means that a member of $\mathcal{R}$ may only join a holon for a specific task if all of the members of set $\mathcal{T}$ have already joined the holon for this specific task. The precedence relation $\prec$ is transitive which means that, for example, $\mathcal{W} \prec \mathcal{R}$ is valid too. The idea behind this definition is that the FCHs which are able to execute tasks flexibly may react to the decisions of FCHs which lack this flexibility in task execution.

To find a complete holon, the FCHs examine the list of tasks, which are announced by the SFC system, and try to reserve the task they would like to execute next for themselves. When an FCH is able to reserve a task successfully for itself, this FCH becomes the representative of the holon for this task. The representative r of a holon for a task t has responsibility to complete the holon for this task. r does this by sending messages to the other FCHs which ask these FCHs to join the holon for task t. A conflict occurs if two representatives send each other messages in which each of them asks the other one to join its own holon. It is possible to describe conflict resolution protocols for this situation which guarantee liveness and fairness of the system.

5.2 TeleTruck

Order dispatching in haulage companies is a complex task. For a system which solves this problem in the real world it is not enough to compute routes for a fleet of trucks for a given set of customer orders. A system supporting the real-world

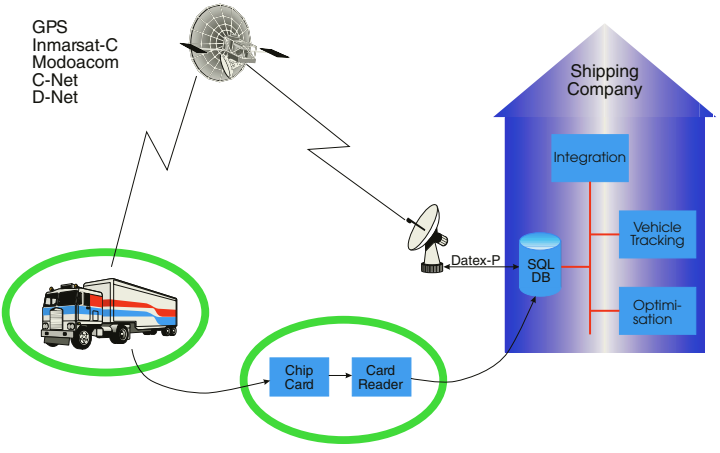


Fig. 7. The online fleet scheduling system TeleTruck.

setting has to cope with an online scheduling problem, in which at any point in time new orders can arrive and in which the system is able to react to problems in the execution of the computed plans. The TeleTruck system (see Fig. 7) implements an online dispatching systems using telecommunication technologies (e.g., satellite or mobile phone communication and global positioning). [11, 12] demonstrated that a MAS approach to model an online dispatching system is feasible and can compete with operation research approaches with respect to quality of the provided solution. However, in these scientific settings the transportation is done by self-contained entities. In practice we see that truck drivers, trucks, and (semi-)trailers are autonomous entities with their own objectives. Only an appropriate group of these entities can all together perform the transportation task. For this reason a holonic approach had to be used to model the agent society of TeleTruck[5].

For each of the physical components (trucks, truck tractors, chassis, and (semi-)trailers) of the forwarding company as well as for each of its drivers there is an agent, which administrates the resources the component or the driver supplies. These agents have their own plans, goals, and communication facilities in order to provide their resources for the transportation plans according to their role in the society. The agents have to form appropriate holons (further on referred to as 'vehicle holons' (VH))in order to execute the orders at hand.

Building a new VH is not just about collecting the needed resources. The components that merge to a VH have to complement each other and match the requirements of the transportation task. For each component an incompatibility list is represented that specifies the incompatibilities to other components, properties of components or orders. These constraints represent technical and legal restrictions and demands for VHs.

The main things that need to be agreed between agents participating in a VH are to go to a specific place at a specific point in time and to load and unload goods. From these activities shared intentions for the agents participating in the

VH can be derived. An new agent representing a *Plan'n'Execute Units* (PnEUs) for the VH is explicitly introduced to maintain the shared intentions of the VH. The PnEU coordinates the formation of the VH representing the transportation entity and plans the vehicle's routes, loading stops, and driving times. The PnEU represents the VH to the outside environment and is authorised to reconfigure it. A PnEU is equipped with planning, coordination, and communication abilities, but does not have its own resources. Each VH that has at least one task to do is headed by such a PnEU. Additionally, there is always exactly one idle PnEU with an empty plan that coordinates the formation of a new VH from idle components if needed.

For the assignment of the orders to the VHs a bidding procedure is used [8]. The dispatch officer in the shipping company interacts with a dispatch agent. The dispatch agent announces the newly incoming orders, specified by the dispatch officer, to the PnEUs via an *extended contract net protocol* (ECNP) [11]. The PnEUs request resources from their components and decide whether the resources are sufficient to fulfil the task or not. If they are sufficient, the PnEU computes a plan, calculates its costs, and bids for the task. If the resources supplied by the components that are already member of the VH are not sufficient – which is trivially the case for the idle PnEU – the task together with the list of missing resources and a set of constraints that the order or the other members of the VH induce is announced to those agents which could supply such resources. These agents calculate which of their resources they can actually supply, and, if necessary, again announce the task and the still missing resources. This is iterated until all the needed resources are collected. The task of collecting the needed resources is not totally left to the PnEU because the components have local knowledge about how they can be combined, e.g. if a driver always drives the same truck it is local knowledge of the components and not of the PnEU.

Thus the ECNP is used on the one hand by the dispatch agent to allocate tasks to the existing VHs and on the other hand by the free PnEU which, itself, uses the protocol to form a new VH. Using the ECNP the dispatch agent initiates the allocation of a transportation order to one or more VHs. The semantics of the announcement is the invitation of the VHs to bid for and execute a task. Fig. 8 shows a dispatch agent announcing a new transportation task to two VHs and the idle PnEU. The complete VH on the left hand side cannot incorporate any further components. Hence, the head requests for the necessary resources from its components and, if these resources are sufficient, calculates the costs for the execution of the task. The second VH could integrate one further component. However, its head first tries to plan for the task using only the resources of the components, the VH already has incorporated. If the resources are not sufficient, the head tries to collect the missing resources by performing an ECNP with idle components that supply such resources. The idle PnEU, which has not yet any resources on its own, first of all performs an ENCP with those idle components that offer loading space; in the example a truck and a trailer. The trailer supplies loading space and chassis, therefore, it needs a motor supplying component. Hence, it announces the task to the truck. The truck which received

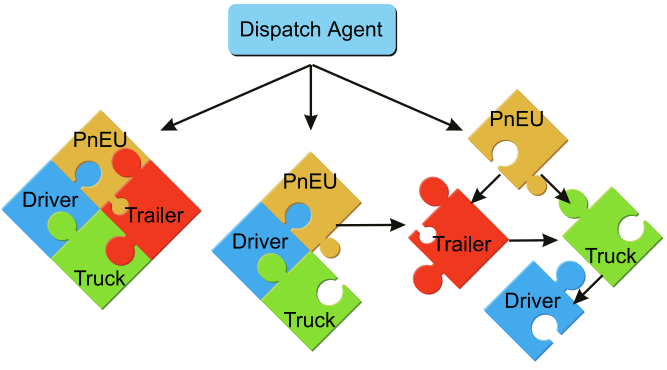


Fig. 8. Holonic Planning in TeleTruck.

two different announcements for the same task – one by the trailer and one by the PnEU directly – can bid in both protocols since it can be sure that only one of the protocols will be successful. Therefore, the truck agent looks for a driver, computes the costs for the two different announcements, and gives a bid both to the PnEU and to the trailer. Obviously, the costs for executing the task with a vehicle that consists of a driver and a truck are less than the costs of executing the same task with the same truck and driver and, in addition, a trailer. Therefore, the idle PnEU will pass the bid of the truck to the dispatch agent. If the task is granted to the idle PnEU, the PnEU merges with the components to a VH and a new PnEU will be created for further bidding cycles. Whenever the plan of a VH is finished the components separate and the PnEU terminates.

Because the tour plans that are computed in the ECNP procedure are sub-optimal [11, 12], the *simulated trading* procedure [1] is used to improve the sub-optimal initial solution stepwise towards globally optimal plans [12]. Simulated trading is a randomised algorithm that realises a market mechanism where the vehicles optimise their plans by successively selling and buying tasks. Trading is done in several rounds. Each round consists of a number of decision cycles. In each cycle the truck agents submit one offer to sell or buy a task. At the end of each round the dispatch agent tries to match the sell and buy offers of the trucks such that the costs of the global solution decrease. This implements a kind of hill-climbing algorithm. Like in the case of simulated annealing, a derivation that decreases from round to round can be specified such that in early rounds the dispatch agent is willing to accept a worsening of the global solution which is helpful to leave local maxima in the solution space. Nevertheless, maxima that are left are saved, such that, when the algorithm terminates before a better solution is found, the best solution hitherto is returned. Hence, simulated trading is an interruptible anytime algorithm.

In order to allow the optimisation not only of the plans but also of the combination of components we extended the simulated trading procedure. It might be the case that a good route plan is not efficient because the allocation of resources to the plan is bad, e.g. a big truck is not full while a smaller truck could

need some extra capacity to improve its own plan. We divided a trading round into three phases. The first phase consists of order trading cycles as explained above; in the middle phase the VHs can submit offers to exchange components. The third phase is, like the first phase, an order trading phase. After the third phase is finished the dispatch agent matches the sell, buy and the component exchange offers.

This application scenario is a perfect instance of the model for local and global optimisation as it was discussed in Section 4. Please note that the local performance profiles in the VHs (i.e. the local cost functions) are used for optimisation as well as a reorganisation of the whole agent society, by switching components among VHs. The ECNP and the simulated trading procedure produce a balance between decentralised and centralised optimisation.

5.3 Train Coupling and Sharing

In this application scenario we have a set of train modules that are able to drive on their own on a railway network. However, if all the train modules drive separately, the capacity utilisation of the railway network is not acceptable. The idea is that the module trains join together and jointly drive some distance (*train coupling and sharing* (TCS) [15], see Fig. 9). The overall goal is to reduce the cost for a given set of transportation tasks in a railway network. Each task is specified as a tuple consisting of the origin and the destination node, the earliest possible departure time and the latest allowed arrival time. As in the TeleTruck system not all of the tasks are announced to the system at the same time. New tasks can come in at any point in time.

We assume that a set of transportation tasks is given, which have to be served by the train modules, and that each task can be served by an individual module, i.e. there is no need to hook two or more modules together to serve an individual task. Likewise, we also assume that a module cannot serve more than one task at a time. All tasks occurring in the system are transportation requests

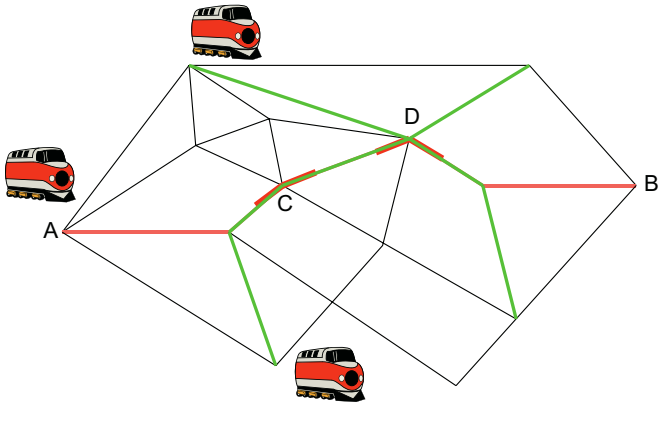


Fig. 9. Routing of Trains in the TCS System.

in a railway network, which is represented as a graph consisting of several nodes connected via so-called *location routes*.

Whenever a module serves a transportation task, it computes the path from the origin to the destination node with a shortest path algorithm. The module then rents the intermediate location routes for a certain time window from the *net manager*. The time window for each location route is uniquely determined by the earliest departure time and the latest arrival time of the transportation task. When a location route is allocated by a certain module, the route is blocked for other modules during this time interval. In order to reduce route blocking, however, two or more modules can decide to share a particular location route.

Route sharing means, that two or more modules hook together at the beginning of a location route (or of a sequence of consecutive routes) and split up afterwards. Route sharing has two advantages: firstly, it increases the average utilisation of location routes because it enables more than one module to use a location route at the same time. Secondly, the cost for renting a location route are reduced for an individual module by distributing the full cost among the participating modules.

We have two natural candidates for becoming the agents in the TCS scenario: the individual modules and the unions (see Fig. 6 and Fig. 7) that emerge when two or more modules decide to share a location route. However, each additional abstraction increases the complexity of the resulting implementation and therefore, we have decided to model the unions as the agents in our system. A single module is encapsulated in a (so-called *degenerated*) union and thus we avoid the additional complexity in the system design. The advantage of applying this scheme is that we do not have to distinguish modules and unions; every active entity in the system is a union.

The main reason for the unions to form shared intentions is that they need to share location routes to reduce cost. This peer matching is again achieved by negotiation processes between the agents in the agent society. As in the TeleTruck system the *contract-net protocol* [8] is used whenever a new task is announced to the system. New tasks are incrementally integrated in the existing schedule which guarantees, that always a solution for the problem (as far as it is known to the system) exists. Note that we assume that there is a sufficient number of modules, i.e. degenerated units, available to execute all tasks. The contract-net protocol is executed whenever a new (degenerated) union has com-

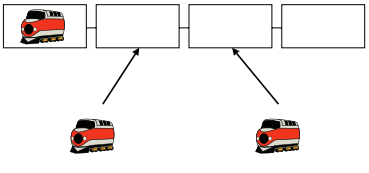


Fig. 10. Train modules group together in unions.

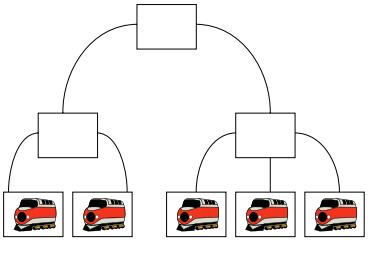


Fig. 11. Even unions can group and form recursively nested holons.

puted its local plan. The union then initiates the contract-net protocol as the manager and offers the plan to the other currently active unions. These unions check if they contain one or more modules that are a potential sharing peers and if this is the case, they offer a sharing commitment to the new union. The new union collects these offers and selects the one that has the largest cost saving potential. It then transfers the module to the winning union and ceases to exist because it does not contain other modules. If no union offers a sharing commitment, the new union remains active as degenerated union. However, as in the TeleTruck system, this solution may be (and usually is) not optimal. In order to improve the quality of the existing solution, the *simulated trading* [1] procedure is run on the set of tasks (or the respective modules) currently known to the system. Unfortunately, executing the simulated trading protocol is a computationally expensive operation and so it is executed only periodically – either after a fixed number of new tasks has been added to the existing solution or explicitly triggered by a user request.

6 Conclusion

The paper presented a general framework to describe self-organisation in holonic MAS. At the first glance there is nothing special about a holon because it behaves to the outside world as any other agent. Most important, one can communicate with a holon by just sending a message to a single address. The concept becomes interesting only when we realise that holons represent a whole group of agents. Compared with an object-oriented programming approach there is no comparable programming construct that would support the design of such systems. We therefore have to further clarify from a theoretical and practical point of view how the formation of holons should take place. In our experience the main reason for agents to form a holon is that they share intentions about how to achieve specific goals. The paper clarifies how the concept of shared intentions can be specified from a theoretical point of view and gives application scenarios in which the process of establishing shared intentions in holonic MAS is illustrated. The application scenarios differ in the sense that in the first one (flexible manufacturing) agents form holons because they have different abilities and can only as a group achieve the task at hand. In the second example (order dispatching in haulage companies) the agents forming a holon have partly overlapping abilities. The last example (train coupling and sharing) demonstrates that even in a setting where we have agents with identical abilities holonic structures can be beneficial.

References

1. A. Bachem, W. Hochstättler, and M. Malich. Simulated Trading: A New Approach For Solving Vehicle Routing Problems. Technical Report 92.125, Mathematisches Institut der Universität zu Köln, Dezember 1992.
2. A. Bond and L. Gasser, editors. *An Analysis of Problems and Research in DAI.* In Bond and Gasser [3], 1988.

3. A. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, Los Angeles, CA, 1988.
4. Luc Bongaerts, László Monostori, Duncan Mc Farlane, and Botond Kádár. Hierarchy in distributed shop floor control. In *IMS-EUROPE 1998, the First Open Workshop of Esprit Working group on IMS*, Lausanne, 1998.
5. H.-J. Bürckert, K. Fischer, and G. Vierke. Transportation scheduling with holonic mas – the teletruck approach. In *Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM'98)*, 1998.
6. L. Cavedon and G. Tidhar. A logical framework for modelling multi-agent systems and joint attitudes. Technical Report Tech. Rep. 66, Australian Artificial Intelligence Institute, 1995.
7. J. Christensen. Holonic manufacturing systems – initial architecture and standard directions. In *Proc. of the 1st European Conference on Holonic Manufacturing Systems*, Hannover, December 1994.
8. R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63 – 109, 1983.
9. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, Massachusetts – London, England, 1995.
10. K. Fischer and J. P. Müller. A decision-theoretic model for cooperative transportation scheduling. In W. van de Velde and J. W. Perram, editors, *Agents Breaking Away – Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, volume 1038 of *LNAI*, pages 177–189. Springer-Verlag, 1996.
11. K. Fischer, J. P. Müller, and M. Pischel. A model for cooperative transportation scheduling. In *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS'95)*, pages 109–116, San Francisco, June 1995.
12. K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling: an application domain for DAI. *Journal of Applied Artificial Intelligence. Special issue on Intelligent Agents*, 10(1):1–33, 1996.
13. D. Kinny, M. Ljungberg, A. S. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems – Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-92 (LNAI Volume 830)*, pages 226–256. Springer-Verlag: Heidelberg, Germany, 1992.
14. A. Koestler. *The Ghost in the Machine*. Arkana Books, 1989.
15. J. Lind, K. Fischer, J. Böcker, and B. Zierkler. Transportation scheduling and simulation in a railroad scenario: A multi-agent approach. In *Proc. of the* 4^{th} *int. Conf. on The Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 325–344, London, UK, 1999. The Practical Application Company Ltd.
16. J. P. Müller. *The Design of Intelligent Agents : A Layered Approach*, volume 1177 of *Lecture Notes in AI*. Springer, 1996.
17. J. Siegel, editor. *CORBA Fundamentals and Programming*. John Wiley and Sons, 1996.
18. H. A. Simon. *The Sciences of the Artificial*. MIT Press, 6 edition, 1990.
19. M. P. Singh. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, 1997.
20. R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transaction on Computers*, number 12 in C-29, pages 1104–1113, 1980.

21. Hans-Jürgen Warnecke. *Aufbruch zum fraktalen Unternehmen – Praxisbeispiele für neues Denken und Handeln*. Springer-Verlag, 1995.
22. G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT – Press, 1999.
23. M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
24. Michael Wooldridge. Agent-based software engineering. *IEEE Proceedings on Software Engineering*, 144(1):26–37, February 1997.
25. Michael Wooldridge. Intelligent agents. In Weiss [22], pages 27–87.
26. A. Yonezawa, editor. *ABCL – An Object-Oriented Concurrent System*. The MIT Press: Cambridge, MA, 1990.

Author Index