Volker Diekert
Mikhail V. Volkov
Andrei Voronkov (Eds.)

LNCS 4649

Computer Science – Theory and Applications

Second International Symposium
on Computer Science in Russia, CSR 2007
Ekaterinburg, Russia, September 2007, Proceedings

Lecture Notes in Computer Science 4649

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Volker Diekert Mikhail V. Volkov
Andrei Voronkov (Eds.)

Computer Science – Theory and Applications

Second International Symposium
on Computer Science in Russia, CSR 2007
Ekaterinburg, Russia, September 3-7, 2007
Proceedings

Springer

Volume Editors

Volker Diekert
Universität Stuttart
Institut für Formale Methoden der Informatik
Universitätsstr. 38, 70569 Stuttgart, Germany
E-mail: diekert@fmi.uni-stuttgart.de

Mikhail V. Volkov
Ural State University
Department of Algebra and Discrete Mathematics
Faculty of Mathematics and Mechanics
Lenina 51, 620083 Ekaterinburg, Russia
E-mail: Mikhail.Volkov@usu.ru

Andrei Voronkov
University of Manchester
School of Computer Science
Kilburn Building, Oxford Road, Manchester, M13 9PL, UK
E-mail: andrej@voronkov.com

Library of Congress Control Number: 2007933004

CR Subject Classification (1998): F.1.1-2, F.2.1-2, F.4.1, I.2.6, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-74509-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74509-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12113064 06/3180 5 4 3 2 1 0

Preface

The 2nd International Symposium on Computer Science in Russia (CSR 2007) was held September 3–7 in Ekaterinburg, Russia, hosted by Ural State University. CSR 2007 was the second event in a series of regular international meetings that started with CSR 2006 in St. Petersburg (see LNCS 3967). The symposium was organized under the auspices of the European Association for Theoretical Computer Science.

The symposium was composed of two tracks: Theory and Applications/Technology. The opening lecture was given by Yuri Gurevich and other invited lectures were given by Scott Aaronson, Rajeev Alur, Peter Druschel, Ziyad Hanna, Bertrand Meyer, Alexei Miasnikov, Geraud Senizergues, and Geoff Sutcliffe. This volume contains the accepted papers of both tracks and the abstracts of seven invited presentations.

The scope of proposed topics for the symposium was quite broad and covered many areas of computer science and its applications. We received 95 submissions, the contributors being from 24 countries. Each submission was reviewed by at least three Program Committee members. The committee decided to accept 34 papers. The reviewing process as well as the preparation of this volume were efficiently supported by the EasyChair conference system.

The following satellite events were collocated with CSR 2007:

- Workshop on Computational Complexity and Decidability in Algebra
- Workshop on Infinite Words, Automata and Dynamics
- Russian Summer School in Information Retrieval

We thank our sponsors: Microsoft Research, Russian Foundation for Basic Research, SKB Kontur, Ural State University, and Yandex.

Yandex, the largest resource on the Russian Internet, established the Yandex the Best Paper Awards and Yandex Best Student Paper Awards for the CSR series. The inauguration of the Yandex Awards formed a part of the Business Meeting of CSR 2007 along with the presentations of the first awarded papers. The following three papers were selected by the Program Committee:

- "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth" by Artur Jez and Alexander Okhotin — Best Paper in the Theory Track
- "Estimation of the click volume by large-scale regression analysis" by Yury Lifshits and Dirk Nowotka – Best Paper in the Applications and Technology Track
- "A fast algorithm for path 2-packing problem" by Maxim Babenko — Best Student Paper

We thank the local Organizing Committee (co-chaired by Vladimir Tretjakov, President of Ural State University, and Vitaly Berdyshev, Director of Mathematics and Mechanics Institute of the Ural Branch of the Russian Academy of Sciences), especially Svetlana Goldberg, Grigoriy Povarov, and Elena Pribavkina.

June 2007

Volker Diekert
Mikhail Volkov
Andrei Voronkov

Conference Organization

Program Committee

Theory Track

Eric Allender	Rutgers University, USA
Sergei Artemov	City University of New York, USA
Eugene Asarin	Université Paris-7, France
Lev Beklemishev	Steklov Institute/Moscow, Russia
Andrei Bulatov	Simon Fraser University, Canada
Evgeny Dantsin	Roosevelt University, USA
Volker Diekert (Chair)	Universität Stuttgart, Germany
Anna Frid	Sobolev Institute/Novosibirsk, Russia
Paul Gastin	ENS de Cachan, France
Joachim von zur Gathen	Universität Paderborn, Germany
Andrew Goldberg	Microsoft Research, USA
Erich Graedel	Universität Aachen, Germany
Dima Grigoriev	CRNS, IRMAR, Rennes, France
Yuri Gurevich	Microsoft Research, USA
Tero Harju	University of Turku, Finland
Edward Hirsch	Steklov Institute/St.Petersburg, Russia
Peter Hoyer	University of Calgary, Canada
Michael Kaminski	Technion – Israel Institute of Technology, Israel
Yuri Matiyasevich	Steklov Institute/St.Petersburg, Russia
Pierre McKenzie	Université de Montréal, Canada
Alexander Razborov	IAS, USA and Steklov Institute/Moscow, Russia
Victor Selivanov	Novosibirsk State Pedagogical University, Russia
Alexander Shen	LIF Marseille, France and IITP Moscow, Russia
Denis Thérien	McGill University, Canada

Applications and Technology Track

Dines Bjørner	The Technical University of Denmark, Denmark
Nikolaj Bjørner	Microsoft Research, USA
Stéphane Bressan	National University of Singapore, Singapore
Gabriel Ciobanu	Institute of Computer Science of Iasi, Romania
Leonid Kalinichenko	IPI Moscow, Russia
Laura Kovacs	RISC Linz, Austria
Torben Ægidius Mogensen	University of Copenhagen, Denmark
Alexandre Petrenko	CRIM, Montréal, Canada
Sibylle Schupp	Chalmers University of Technology, Sweden
Oleg Sokolski	University of Pennsylvania, USA
Helmut Veith	Technische Universität Müchen, Germany
Andrey Voronkov (Chair)	University of Manchester, UK

Conference Chair

Mikhail Volkov — Ural State University, Russia

Steering Committee for CSR Conferences

Anna Frid — Sobolev Institute/Novosibirsk, Russia
Edward Hirsch — Steklov Institute/St.Petersburg, Russia
Juhani Karhumäki — University of Turku, Finland
Mikhail Volkov — Ural State University, Russia

Referees

Bogdan Aman
Sergei Avgustinovich
Maxim Babenko
Reuven Bar-Yehuda
Eli Ben-Sasson
Daniel Berend
Therese Biedl
Markus Blaeser
Cosmin Bonchis
Sergiy Boroday
Mark Braverman
Olivier Carton
Julien Cassaigne
Steve Chain
Arkadev Chattopadhyay
Rafi Chen
Christian Choffrut
Maria Chudnovsky
Veronique Cortier
Felipe Cucker
Arnaud Dury
Henning Fernau
Hasham Hallal
May Haydar
Dmitry Itsykson
Ethan Jackson
Stasys Jukna
Jarkko Kari
Dmitry Karpov
Stefan Katzenbeisser
Olga Kharlampovich
Boris Konev
Levente Kovacs
Steve Kremer
Manfred Kufleitner
Stefan Kugele
Alexander Kulikov
Orna Kupferman
Roman Kuznets
Fribourg Laurent
Leonid Levin
Ming Li
Yury Lifshits
Alexis Maciel
Guillaume Malod
Bernard Mans
Larry Moss
Frantisek Mraz
Sergey Nikolenko
Dirk Nowotka
Friedrich Otto
Eric Pacuit
Alexei Pastor
Mati Pentus
Holger Petersen
Martin Platek
Igor Potapov
Artem Pyatkin
Christian Reitwiessner
Maurice Rojas

Samer Salame
Rahul Santhanam
John E. Savage
Uwe Schöning
Bart Selman
Rocco Servedio
Jeffrey Shallit
Evgeny Skvortsov
Anatol Slissenko
Andreas Steininger
Sergey Stupnikov
Alexei Talambutsa
Suguru Tamaki
Tony Tan
Michael Tautschnig
Pascal Tesson
Thomas Thierauf
Avraham Trahtman
Nikolay Vereshchagin
Volodya Vovk
Klaus Wagner
Derry Wijaya
Alexander Wolpert

Sponsors

Microsoft Research
Russian Foundation for Basic Research
SKB Kontur
Ural State University
Yandex
ZAO Spectralus

Table of Contents

Proving Church's Thesis
(Abstract)

Yuri Gurevich

Microsoft Research

The talk reflects recent joint work with Nachum Dershowitz [4].

In 1936, Church suggested that the recursive functions, which had been defined by Gödel earlier that decade, adequately capture the intuitive notion of a computable ("effectively calculable") numerical function[1] [2]. Independently Turing argued that, for strings-to-strings functions, the same goal is achieved by his machines [11].

The modern form of Church's thesis is due to Church's student Kleene. It asserts that every computable numerical partial function is partial recursive. (Originally Church spoke of total functions.)

Kleene thought that the thesis as unprovable: "Since our original notion of effective calculability... is a somewhat vague intuitive one, the thesis cannot be proved" [7]. But he presented evidence in favor of the thesis. By far the strongest argument was Turing's analysis [11] of "the sorts of operations which a human computer could perform, working according to preassigned instructions" [7]. The argument convinced Gödel who thought the idea "that this really is the correct definition of mechanical computability was established beyond any doubt by Turing" [5].

Moreover, Gödel has been reported to have thought "that it might be possible ... to state a set of axioms which would embody the generally accepted properties of [effective calculability], and to do something on that basis" [3]. As explained by Shoenfield [10]:

> It may seem that it is impossible to give a proof of Church's Thesis. However, this is not necessarily the case.... In other words, we can write down some axioms about computable functions which most people would agree are evidently true. It might be possible to prove Church's Thesis from such axioms.... However, despite strenuous efforts, no one has succeeded in doing this (although some interesting partial results have been obtained).

We will demonstrate that, under certain very natural hypotheses regarding algorithmic activity, called the "Sequential Postulates" [6], Church's Thesis is in fact provable. In brief, the postulates say the following.

I. **Sequential Time.** *An algorithm determines a sequence of "computational" states for each valid input.*

[1] For brevity we use the term numerical function to mean a function from natural numbers to natural numbers.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 1–3, 2007.

II. **Abstract State.** *The states of a computational sequence can be arbitrary (first-order) structures.*

III. **Bounded Exploration.** *The transitions from state to state in the sequence are governed by a finite description.*

For precise formulation of the three postulates see the article [6]. With Bounded Exploration, an algorithm computes in "steps of limited complexity", as demanded by Kolmogorov [8]. This postulate thereby answers Kolmogorov's implicit question: What does it mean to bound the complexity of each individual step?

The postulates are justified in the article [6]. On this ground, a (sequential) algorithm is defined there as any object satisfying the three postulates. One may worry that this definition is too liberal. To this end, the article proves that (sequential) abstract state machines, introduced earlier by the author, satisfy the three postulates, and that, for every algorithm, there is an abstract state machine that emulates the algorithm.

To focus on numerical algorithms, we add the following postulate:

IV. Only basic arithmetic operations are available initially.

Algorithms satisfying postulate IV will be called numerical.

We will show that Church's Thesis provably follows from these four postulates.

Theorem 1. *Any numerical partial function is computed by a numerical algorithm if and only if it is partial recursive.*

Thus, to the extent that one might entertain the notion that there exist non-recursive effective functions, one must reject one or more of these postulates. In a similar way, we can prove Turing's thesis from postulates I–III and a postulate.

V. Only basic string operations are available initially.

Theorem 1 generalizes to the case when oracles are present. If only oracle functions are available initially, postulates I–III suffice. No additional postulates are needed.

Our goal in this work has been to remedy the situation described thus by Montague [9]: "Discussion of Church's thesis has suffered for lack of a precise general framework within which it could be conducted." We show how the Sequential ASM Postulates provide just such a framework. As we mentioned, Gödel surmised that Church's Thesis may follow from appropriate axioms of computability. But, as far as we can ascertain, no complete axiomatization has previously been presented in the literature. In fact, the challenge of proving Church's Thesis is first in Shore's list of "pie-in-the-sky problems" for the twenty-first century [1].

References

1. Buss, S.R., Kechris, A.A., Pillay, A., Shore, R.A.: Prospects for mathematical logic in the twenty-first century. Bulletin of Symbolic Logic 7(2), 169–196 (2001)
2. Church, A.: An unsolvable problem of elementary number theory. American Journal of Mathematics 58, 345–363 (1936)
3. Davis, M.: Why Gödel didn't have Church's thesis. Information and Control 54, 3–24 (1982)
4. Dershowitz, N., Gurevich, Y.: A natural axiomatization of Church's thesis (to appear)
5. Gödel, K.: Kurt Gödel: Collected Works. In: Feferman, S. (ed.) Unpublished essays and lectures, vol. III, p. 168. Oxford University Press, Oxford (1995)
6. Gurevich, Y.: Sequential abstract state machines capture sequential algorithms. ACM Transactions on Computational Logic 1, 77–111 (2000)
7. Kleene, S.C.: Introduction to Metamathematics, North Holland, Amsterdam (1952)
8. Kolmogorov, A. N.: O ponyatii algoritma (On the concept of algorithm), Uspekhi Matematicheskikh Nauk, vol. 8(4), pp. 175–176 (1953) (in Russian). An English translation is In: Uspensky, V.A., Semenov, A.L.: Algorithms: Main Ideas and Applications, pp. 18–19. Kluwer (1993)
9. Montague, R.: Towards a general theory of computability. Synthese 12(4), 429–438 (1960)
10. Shoenfield, J.R.: Recursion Theory. Lecture Notes in Logic, vol. 1. Springer, New York (1991)
11. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society, vol. 42, pp. 230–265, 1936–37. Corrections in vol. 43, pp. 544–546 (1937)

The Limits of Quantum Computers

Scott Aaronson*

University of Waterloo

In the popular imagination, quantum computers would be almost magical devices, able to "solve impossible problems in an instant" by trying exponentially many solutions in parallel. In this talk, I'll describe four results in quantum computing theory that directly challenge this view.

First, I'll show that any quantum algorithm to decide whether a function $f : [n] \to [n]$ is one-to-one or two-to-one needs to query the function at least $\sim n^{1/5}$ times [1]. This provides strong evidence that collision-resistant hash functions, and hence secure electronic commerce, would still be possible in a world with quantum computers.

Second, I'll show that in the "black-box" or "oracle" model that we know how to analyze, quantum computers could not solve NP-complete problems in polynomial time, even with the help of nonuniform "quantum advice states" [2].

Third, I'll show that quantum computers need exponential time to find local optima—and surprisingly, that the ideas used to prove this result also yield new classical lower bounds for the same problem [4].

Finally, I'll show how to do "pretty-good quantum state tomography" using a number of measurements that increases only linearly, not exponentially, with the number of qubits [3]. This illustrates how one can sometimes turn the limitations of quantum computers on their head, and use them to develop new techniques for experimentalists.

No quantum computing background will be assumed.

References

1. Aaronson, S.: Quantum lower bound for the collision problem. In: Proc. ACM STOC, pp. 635–642, (2002)
2. Aaronson, S.: Limitations of quantum advice and one-way communication. Theory of Computing 1, 1–28 (2004)
3. Aaronson, S.: The learnability of quantum states (2006)
4. Aaronson, S.: Lower bounds for local search by quantum arguments. SIAM J. Comput. 35(4), 804–824 (2006)

* scott@scottaaronson.com

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, p. 4, 2007.

Marrying Words and Trees

Rajeev Alur

University of Pennsylvania

We discuss the model of *nested words* for representation of data with both a linear ordering and a hierarchically nested matching of items. Examples of data with such dual linear-hierarchical structure include annotated linguistic data, executions of structured programs, and HTML/XML documents. Nested words generalize both words and ordered trees, and allow both word and tree operations. We define *nested word automata*—finite-state acceptors for nested words, and show that the resulting class of regular languages of nested words has all the appealing theoretical properties that the classical regular word languages enjoy such as determinization, closure under a variety of operations, decidability of emptiness as well as equivalence, and characterization using monadic second order logic. The linear encodings of nested words gives the class of *visibly pushdown languages* of words, and this class lies between balanced languages and deterministic context-free languages. We argue that for algorithmic verification of structured programs, instead of viewing the program as a context-free language over words, one should view it as a regular language of nested words (or equivalently, as a visibly pushdown language), and this would allow model checking of many properties (such as stack inspection, pre-post conditions) that are not expressible in existing specification logics. We also study the relationship between ordered trees and nested words, and the corresponding automata: while the analysis complexity of nested word automata is the same as that of classical tree automata, they combine both bottom-up and top-down traversals, and enjoy expressiveness and succinctness benefits over tree automata. There is a rapidly growing literature related to nested words, and we will briefly survey results on languages infinite nested words, nested trees, temporal logics over nested words, and new decidability results based on visibility.

References

[Alu07] Alur, R.: Marrying words and trees. In: Proceedings of the 26th ACM Symposium on Principles of Database Systems. ACM Press, New York (2007)

[AM04] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of the 36th ACM Symposium on Theory of Computing, pp. 202–211. ACM Press, New York (2004)

[AM06] Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, p. 5, 2007.

TPTP, TSTP, CASC, etc.

Geoff Sutcliffe

University of Miami, USA
geoff@cs.miami.edu

Abstract. This paper gives an overview of activities and products that stem from the Thousands of Problems for Theorem Provers (TPTP) problem library for Automated Theorem Proving (ATP) systems. These include the TPTP itself, the Thousands of Solutions from Theorem Provers (TSTP) solution library, the CADE ATP System Competition (CASC), tools such as my semantic Derivation Verifier (GDV) and the Interactive Derivation Viewer (IDV), meta-ATP systems such as the Smart Selective Competition Parallelism (SSCPA) system and the Semantic Relevance Axiom Selection System (SRASS), and applications in various domains.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of systems that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. The dual discipline, automated model finding, develops computer programs that establish that a set of statements is consistent, and in this work we consider automated model finding to be part of ATP. These capabilities lie at the heart of many important computational tasks. For example, formal methods for software and hardware design and verification [Lam05, Das06], the analysis of network security protocols [AB04, Mit05], solving hard problems in mathematics [SFS95, McC97], and inference for the semantic web [FS005]. ATP has been highly successful when the problem is expressed in classical first order logic, so that a refutation or model of the clause normal form of the problem can be obtained. There are some well known high performance ATP systems that search for a refutation or model of a set of clauses, e.g., Darwin/DarwinFM [BFT06], E/EP [Sch02], Mace [McC03], Paradox [CS03], SPASS [WBH+02], Vampire [RV02], and Waldmeister [Hil03]. Throughout this paper (until Section 8 that describes future plans) all discussion is in terms of ATP for classical first order logic.

2 TPTP

The TPTP (Thousands of Problems for Theorem Provers) problem library [SS98] is a well known standard set of test problems for ATP systems. The TPTP supplies a comprehensive library of the ATP test problems that are available

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 6–22, 2007.

today, in order to provide an overview and a simple, unambiguous reference mechanism. The principal motivation for the TPTP is to support the testing and evaluation of ATP systems, to help ensure that performance results accurately reflect the capabilities of the ATP system being considered. The problems in the TPTP are collected from various sources. The two principal initial sources were existing electronic problem collections and the ATP literature. Since then many people and organizations have contributed to the TPTP. Users of ATP systems find that contributing samples of their problems to the TPTP provides exposure to ATP system developers, who then improve their systems' performance on the problems, which completes a cycle to provide the users with more effective tools.

The problems in the TPTP are classified into domains that reflect the natural hierarchy of scientific domains, as presented in standard subject classification literature. The current TPTP (v3.3.0) has thirty domains, in the fields of logic, mathematics, computer science, science and engineering, and social sciences, with domains ranging from combinatory logic to Smullyanesque puzzles. Each problem has a unique name that reflects its domain and encoding. Each problem file has a header section that contains information for the user, such as references, the problem rating (see Section 2.1), the problem status (see Section 2.2), etc. The logical formulae are wrapped with annotations that provide a unique name for each formula in the problem, a user role (axiom, conjecture, etc), and auxiliary user information. The logical formulae themselves use a consistent and easily understood notation. The syntax (see Section 2.3) shares many features with Prolog, a language that is widely known in the ATP community. Indeed, with a few operator definitions, units of TPTP data can be read in Prolog using a single `read`/1 call, and written with a single `writeq`/1 call.

A key to the initial and ongoing success of the TPTP is the TPTP2X utility. The most important feature of TPTP2X is the conversion of TPTP format problems to formats used by existing ATP systems. This functionality provides a very low entry barrier to using the TPTP with existing ATP systems that cannot read the TPTP format.

The availability of the TPTP has provided a stable basis for the meaningful evaluation of ATP systems, and published results can be readily compared with new results to determine progress in the field. Although other test problems do exist and are sometimes used, the TPTP is now the de facto standard for testing first order ATP systems.

2.1 Ratings

An important feature of the TPTP is the problem ratings [SS01]. The ratings provide an accurate measure of how difficult the problems are for state-of-the-art ATP systems. To rate problems, the performance of contemporary ATP systems on the problems is analyzed. The performance data comes from the TSTP, described in Section 3. The unbiased problems of the TPTP are divided into Specialist Problem Classes (SPCs) - syntactically identifiable classes of problems for which certain ATP techniques or systems have been observed to be especially well suited. Rating is done separately for each SPC, to provide a rating that

compares "apples with apples". A partial order between systems is determined according to whether or not a system solves a strict superset of the problems solved by another system. If a strict superset is solved, the first system is said to subsume the second system. The union of the problems solved by the non-subsumed systems defines the state-of-the-art - all the problems that are solved by any system. The fraction of non-subsumed systems that fail on a problem is the difficulty rating for the problem. Problems that are solved by all non-subsumed systems get a rating of 0.00, and are considered to be easy; problems that are solved by just some of the non-subsumed systems get a rating between 0.00 and 1.00, and are considered difficult; problems that are unsolved get a rating of 1.00.

The analysis done for problem ratings also provides ratings for the ATP systems. The fraction of the difficult unbiased problems that a system solves is the rating for that system. Systems that subsume all other systems get a rating of 1.00, and systems that solve only easy problems get a rating of 0.00.

2.2 SZS

In order to use ATP systems' results as input to other tools, it is necessary that the results correctly and precisely specify what has been established. The SZS ontology [SZS04] provides a fine grained ontology of result and output forms that are used to specify what has been established about a given ATP problem. The ontology also recommends the precise way in which the ontology values should be reported in the output from systems and tools. Figure 1 shows an extract from the top of the result ontology (the full ontology is available as part of the TPTP distribution). Each value has a full name and a three letter acronym that is useful for tables of data.

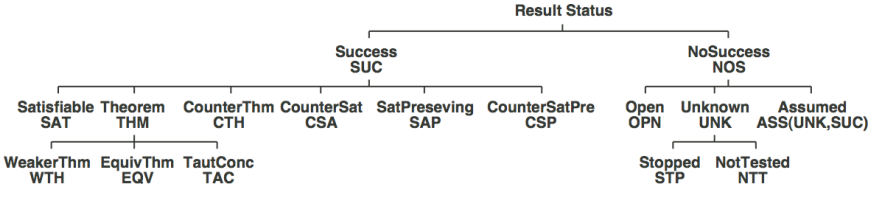


Fig. 1. SZS Ontology

At the top level the result ontology splits into two. The `Success` part catalogs semantic relationships between the axioms and conjecture (or its negation) of a problem. For example, if all models of the axioms are models of the conjecture then the status is `Theorem` with code `THM`, if some models of the axioms are models of the negation of the conjecture then the status is `CounterSatisfiable` with code `CSA`, and if there is a bijection between the models of the axioms and the models of the conjecture (as in a Skolemization step) then the status is `SatisfiabilityBijection` with code `SAB`. The `NoSuccess` part of the result ontology catalogs reasons that a system or tool could be not successful. For

example, if a system stopped because the CPU time limit ran out then the status is `Timeout` with code `TMO`, and if a system has not attempted a problem but might in the future then the status is `NotTestedYet` with code `NTY`.

The output ontology catalogs forms of output from ATP systems and tools. For example, a prover might output a `CNFRefutation` with code `CRf`, and a model finder might output a `FiniteModel` with code `FMo`.

2.3 BNF

One of the keys to the success of the TPTP and related projects is their consistent use of the TPTP language, which enables convenient communication between different systems and researchers. TPTP v3.0.0 introduced a new version of the TPTP language [SSCVG06]. The language was designed to be suitable for writing both ATP problems and ATP solutions, to be flexible and extensible, and easily processed by both humans and computers.

A principal goal of the development of the language grammar was to make it easy to translate the BNF into `lex/yacc/flex/bison` input, so that construction of parsers (in languages other than Prolog) can be a reasonably easy task [VGS06]. To this end the language definition uses a modified BNF meta-language that separates syntactic, semantic, lexical, and character-macro rules. The separation of syntax from semantics eases the task of building a syntactic analyzer. At the same time, the semantic rules provide the detail necessary for semantic checking.

The latest release of the grammar provides further structuring that allows users to build a parser for chosen components according to their need, including the FOF and CNF core, the TFF extension, extensions for theories (e.g., arithmetic), the THF core, and various THF extensions (see Section 8 regarding the TFF and THF languages).

3 TSTP

The TSTP (Thousands of Solutions from Theorem Provers) solution library [SutRL], the "flip side" of the TPTP, is becoming known as a resource for contemporary ATP systems' solutions. In particular, the TSTP contains solutions to problems from the TPTP. One use of the TSTP is for ATP system developers to examine solutions to problems and thus understand how they can be solved, leading to improvements to their own systems.

A key development from the old (pre-v3.0.0) TPTP language to the new one was the addition of features for writing ATP solutions [SSCVG06], in a format consistent with ATP problems (see Sections 2 and 2.3). This enables output from ATP systems to be seamlessly used as input to further systems or tools. The features were designed for writing derivations, but their flexibility makes it possible to write a range of DAG structures. Additionally, there are features of the language that make it possible to conveniently specify finite interpretations.

At the time of writing this paper, the TSTP contains the results of running 44 ATP systems and system variants on all the problems in the TPTP. The results

are classified according to the TPTP problem domains, then by TPTP problem. Each result file has a header section that contains information for the user, such as the system command line, information about the computer used, the SZS result and output status (see Section 2.2), and statistics about the solution. The output logical formulae use the TPTP language. Additional information that specifies a derivation's DAG structure, and details of inference steps, is used to annotate each formula, for use by tools such as GDV (see Section 5.4) and IDV (see Section 5.5).

4 CASC

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, CASC (the CADE ATP System Competition) [SS06] is held at each CADE conference. CASC evaluates the performance of sound, fully automatic, ATP systems – it is the world championship for such systems. The primary purpose of CASC is a public evaluation of the relative capabilities of ATP systems. Additionally, CASC aims to stimulate ATP research in general, to stimulate ATP research towards autonomous systems, to motivate implementation of robust ATP systems, to provide an inspiring environment for personal interaction between ATP researchers, and to expose ATP systems within and beyond the ATP community. Fulfillment of these objectives provides stimulus and insight for the development of more powerful ATP systems, leading to increased and more effective usage.

The design of CASC is linked to the problem and system rating scheme described in Section 2.1. The divisions and problem categories of CASC are similar to the SPCs used in the rating scheme. The problem ratings make it possible to select appropriately difficult problems for CASC, to differentiate between the systems. The rating scheme provides the principles for the CASC rating scheme, which provides a realistic and stable ranking of the systems. Table 1 lists the division winners over the years.

Through successive refinement of the competition design, CASC has been of significant benefit to the development of ATP [Nie02]. CASC has had two main effects on ATP system development. First, new strategies and techniques have been developed to increase the range of problems that can be solved by individual systems, and second, the quality of implementations has improved. Possibly the most important improvement has been in the selection of strategies according to the characteristics of the given problem – the "auto-mode"s now available in almost all ATP systems. There have been significant developments in this area, including a deeper understanding of what problem characteristics are important for what aspects of strategy selection, the examination of the input to detect the domain structure of the problem, the use of machine learning techniques to optimize the choice of strategy, and the use of strategy scheduling. Other effects of CASC include increased interest in the production and verification of ATP system output, the development and refinement of FOF to CNF converters,

Table 1. CASC division winners

	FOF	CNF	SAT	EPR	UEQ
J3	Vampire 8.1	Vampire 8.1	Paradox 1.3	Darwin 1.3	Waldmeister 806
20	Vampire 8.0	Vampire 8.0	Paradox 1.3	DCTP 10.21p	Waldmeister 704
J2	Vampire 7.0	Vampire 7.0	Gandalf c-2.6-SAT	DCTP 10.21p	Waldmeister 704
19	Vampire 5.0	Vampire 6.0	Gandalf c-2.6-SAT	DCTP 1.3-EPR	Waldmeister 702
18	Vampire 5.0	Vampire 5.0	Gandalf c-2.5-SAT	E-SETHEO csp02	Waldmeister 702
JC	E-SETHEO csp01	Vampire 2.0	GandalfSat 1.0	E-SETHEO csp01	Waldmeister 601
17	VampireFOF 1.0	E 0.6	GandalfSat 1.0		Waldmeister 600
16	SPASS 1.00T	Vampire 0.0	OtterMACE 437		Waldmeister 799
15	SPASS 1.0.0a	Gandalf c-1.1	SPASS 1.0.0a		Waldmeister 798
14	SPASS 0.77	Gandalf	SPASS 0.77		Waldmeister
14		E-SETHEO			Otter 3.0.4z

systems that are robust in terms of installation and execution, and improved engineering and data structures in ATP systems.

5 Tools

The TPTP and TSTP are supported by a suite of tools for preparing and solving problems in TPTP format, and for processing solutions in TPTP format. Five of these tools are described in this section, along with WWW interfaces that provide global access to the TPTP, the TSTP, and the tools.

5.1 SystemOnTPTP

SystemOnTPTP is a utility that allows an ATP problem or solution to be easily and quickly submitted in various ways to a range of ATP systems and tools. The utility uses a suite of currently available ATP systems and tools, whose properties (input format, reporting of result status, etc) are stored in a simple text database. The utility allows the input to be selected from the TPTP or TSTP library, or provided in TPTP format by the user. One or more systems or tools may be applied to the input.

The implementation of SystemOnTPTP uses several subsidiary tools to preprocess the input, control the execution of the chosen ATP system(s), and postprocess the output. On the input side TPTP2X (see Section 2) is used to prepare the input for processing.[1] A strict resource limiting program called `TreeLimitedRun` is used to limit the CPU time and memory used. `TreeLimitedRun` monitors processes more tightly than is possible with standard operating system calls. (`TreeLimitedRun` is also used in CASC (see Section 4).) Finally a program called `X2tptp` converts an ATP system's output to TPTP format, if requested by the user.

5.2 Prophet

Prophet uses the syntax of an axiom formula F_a to gauge the potential for the axiom to contribute to a proof of a conjecture F_c, in the context of a set S of

[1] In some situations a faster, recently developed, alternative called TPTP4X is used.

axioms and the conjecture. First, the contextual direct relevance between all formulae in the set is measured by

$$\frac{\sum_{s \in (sym(F_a) \cap sym(F_c))} \left(1 - \frac{|\{f : f \in S, s \in sym(f)\}|}{|S|}\right)}{|sym(F_a) \cup sym(F_c)|}$$

Next, the contextual path relevance of every path $F_a = F_1 \cdot F_2 \cdot \ldots \cdot F_n = F_c$ from F_a to F_c is calculated as the smallest contextual direct relevance in the path, divided by the length of the path. Finally, the contextual indirect relevance between F_a and F_c is taken as the maximal contextual path relevance over all paths connecting F_a to F_c.

Contextual indirect relevance can be used as a heuristic for selecting formulae to use in an ATP system – this is done in the SRASS system described in Section 6.2. A upgraded version of Prophet, based on deeper information retrieval concepts [Sah06], is being developed.

5.3 AGInT

AGInT (Automatic Generation of Interesting Theorems) [PGS06] is a tool that discovers interesting theorems of a given set of axioms. AGInT uses a deductive approach to discovery - it uses an ATP system to generate CNF logical consequences of the axioms, filters the logical consequences to extract interesting theorems, and then computes an interestingness rating for each theorem. This basic process takes place in the context of an outer level control loop that regularly refocuses the generation of logical consequences, thus enabling AGInT to proceed deeply into the search space of logical consequences. The overall architecture of AGInT is shown in Figure 2. The final output from AGInT is an ordered list of the interesting theorems retained in the interesting theorems store.

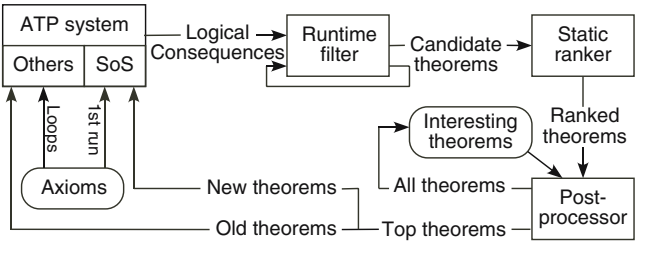


Fig. 2. AGInT Architecture

The runtime filter measures up to eight "interestingness" features of the formulae (some features are inappropriate in some situations): preprocessing detects and discards obvious tautologies, obviousness estimates the difficulty of proving a formula, weight estimates the effort required to read a formula, complexity estimates the effort required to understand a formula, surprisingness measures new relationships between function and predicate symbols in a formula, intensity

measures how much a formula summarizes information from its leaf ancestors, adaptivity measures how tightly the universally quantified variables of a formula are constrained, and focus measures the extent to which a formula is making a positive or negative statement about the domain. Formulae that pass the majority of the runtime filters are passed to the static ranker. The static ranker combines the measures from the runtime filter with a measure of usefulness, which measures how much an interesting theorem has contributed to proofs of further interesting theorems. The scores are then normalized and averaged to produce an interestingness score.

AGInT has been evaluated in several domains and applications, ranging from puzzles to set theory. A particularly useful application has been in generating proof synopses in IDV, described in Section 5.5.

5.4 GDV

ATP systems are complex pieces of software, and thus may have bugs that make them unsound or incomplete. While incompleteness is common (sometimes by design) and tolerable, when an ATP system is used in an application it is important, typically mission critical, that it be sound. GDV (my Derivation Verifier) [Sut06] is a tool that uses structural and then semantic techniques to verify a derivation in TPTP format.

Structural verification checks that inferences have been done correctly in the context of the derivation. The structural checks include: checking that the specified parents of each inference step do exist, checking that the derivation is acyclic, checking that refutations end with a $false$ formula, checking that assumptions are discharged, checking that split refutations are not mutually dependent, and checking that introduced symbols (e.g., in Skolemization) are distinct.

The core technique in semantic verification is to encode the expected semantic relationship between each inferred formula and its parent formulae into logical obligations, in the form of ATP problems. The obligations are then discharged by having trusted ATP systems solve the ATP problems. The required semantic relationship between an inferred formula and its parent formulae depends on the intent of the inference rule used. For example, deduction steps are verified by checking that the inferred formula is a logical consequence of its parent formulae. This intent is recorded as an SZS annotation to each inferred formula in TPTP format derivations (see Sections 2.2 and 3). GDV uses SystemOnTPTP to control the trusted ATP systems.

5.5 IDV

The proofs output by automated reasoning systems provide useful information to users, e.g., the proof structure, lemmas that may be useful in future proofs, which axioms are most used, etc. However, the proofs output by automated reasoning systems are often unsuitable for human consumption, due to, e.g., the conversion to CNF, use of proof by contradiction, and use of fine grained inference steps. IDV (Interactive Derivation Viewer) [TPS06] is a tool for graphical rendering of

derivations in TPTP format. IDV provides an interactive interface that allows the user to quickly view various features of the derivation, and access various analysis facilities.

The lefthand side of Figure 3 shows the rendering of the derivation output by EP 0.99 for the TPTP problem PUZ001+1.[2] The IDV window is divided into three panes: the top control strip pane provides control buttons and sliders, the main middle pane shows the rendered DAG, and the bottom pane gives the text of the annotated formula for the node pointed to by the mouse.

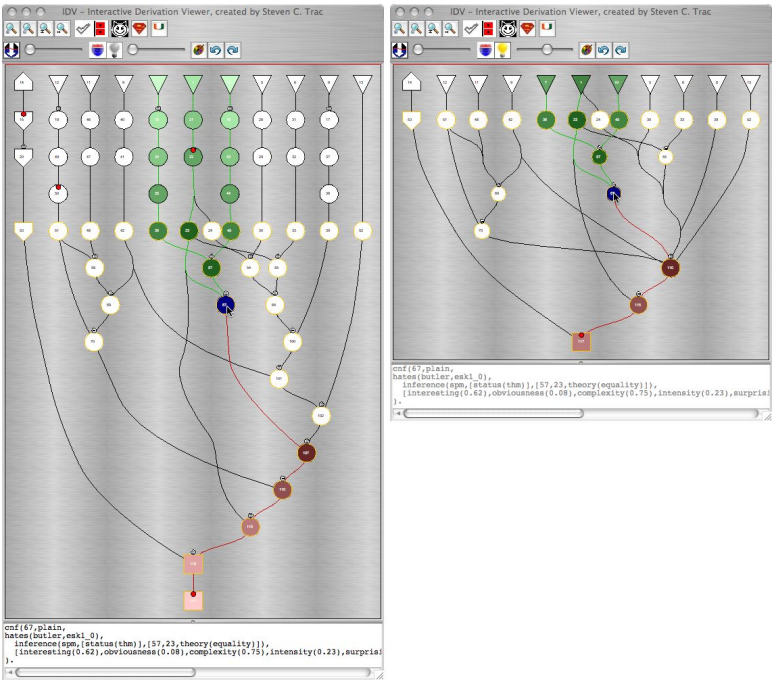


Fig. 3. EP's Proof by Refutation of PUZ001+1

The rendering of the derivation DAG uses shape and color to visually provide information about the derivation. The shape of a node gives the user role of the formula, the color indicates FOF or CNF, and tags indicate features of the inference step. The user can interact with the rendering in various ways. Mousing over a node highlights its ancestors and descendants, clicking on a node pops up a window with details of the inference and with access to GDV to verify the inference, control-clicking on a node opens a new IDV window with just that node and it's parents, and shift-control-clicking on a node opens a new IDV window with that node and its ancestors.

[2] PUZ001+1 is the "Aunt Agatha" problem, a scenario in which one of the three people who live in the mansion killed Aunt Agatha. The goal is to prove that Aunt Agatha killed herself.

The buttons and sliders in the control strip pane provide a range of manipulations on the rendering – zooming, hiding and displaying parts of the DAG according to various criteria, access to GDV for verification of the whole derivation, and access to the SystemOnTSTP interface described in Section 5.6. A particularly novel feature of IDV is its ability to provide a synopsis of a derivation by identifying interesting lemmas within a derivation, and hiding less interesting intermediate formulae. This is implemented by calling AGInT to evaluate the interestingness of each formula, and then using the interestingness slider to select an interestingness threshold to hide nodes for less interesting formulae. After extracting a synopsis it is possible to zoom in with the redraw button, rendering only the "interesting" nodes. A synopsis is shown on the righthand side of Figure 3.

5.6 WWW

All of the data and tools described in the preceding sections, and a few more besides, are accessible via a suite of online interfaces. The TPTP and TSTP interfaces provide access to all the problems, solutions, and documents related to the libraries, as well as subprojects and proposals for forthcoming extensions to the libraries. The SystemB4TPTP, SystemOnTPTP [Sut00], and SystemOnTSTP interfaces provide access to an online service that uses the SystemOnTPTP utility to run selected ATP systems and tools on input specified by the user. The service can also be accessed directly via `http POST` requests. The input can be selected from the TPTP or TSTP library, or provided in TPTP format by the user as text, a file, or a URL source.

In addition to using the SystemOnTPTP utility, the SystemB4TPTP interface provides access to tools to convert from other input formats to TPTP format. The SystemOnTPTP interface additionally provides system reports, and recommendations for systems to use on a given problem, based on the system ratings (see Section 2.1). The SystemOnTPTP interface also has direct access to SSCPA (described in Section 6.1) to run multiple systems in competition parallel.

6 Meta-ATP

One of the benefits of having the common TPTP format for data, and the SZS ontology for status (see Section 2), is that it becomes easily possible to seamlessly integrate ATP systems and tools into more complex and effective reasoning systems. Two examples of such systems are described in this section.

6.1 SSCPA

One approach to developing more powerful ATP systems is the use of parallelism. SSCPA (Smart Selective Competition Parallelism ATP) [SS99] is an uncooperative multiple calculus competition parallelism ATP system. SSCPA runs multiple sequential ATP systems concurrently, in an SMP environment. This approach is motivated by the observation, e.g., in CASC, that no individual ATP system performs well on all problems. There is convincing evidence that the specialization

of ATP systems is due to the deduction techniques used in relation to the syntactic characteristics of the problems. SSCPA uses the syntactic characteristics of a given problem to classify it into one of the specialist problem classes described in Section 2.1. SSCPA uses the system ratings (also described in Section 2.1) to determine which of the available ATP systems perform well for that class. A selection of the recommended systems are then run concurrently under the control of SystemOnTPTP (see Section 5.1), in one of several user selectable modes.

SSCPA was evaluated by entering it into the demonstration division of CASC-16 (see Section 4), running under the same conditions as the regular competition entries. In the mixed CNF and satisfiable CNF divisions SSCPA solved more problems than the competition winner. SSCPA also performed reasonably well in the first-order and unit equality divisions.

6.2 SRASS

In recent years the ability of ATP systems to reason over large theories – theories in which there are many functors and predicates, many axioms of which typically only a few are required for the proof of a theorem, and many theorems to be proved from the same set of axioms – has become more important. Large theory problems are becoming more prevalent as large knowledge bases, e.g., ontologies and large mathematical knowledge bases, are translated into forms suitable for automated reasoning [Qua92, Urb07, RPG05], and mechanical generation of ATP problems becomes more common, e.g., [DFS04, MP06]. SRASS (Semantic Relevance Axiom Selection System) [SP07] is a system for selecting necessary axioms, from a large set also containing superfluous axioms, to obtain a proof of a conjecture.

The basic algorithm of SRASS is to start with a set containing the negation of the conjecture to be proved, then repeatedly find a model of the set and augment the set with an axiom that is $false$ in the model, until no model exists. In this state the conjecture is a logical consequence of the selected axioms. SRASS augments this basic algorithm with extensions that improve the implemented performance. One of the keys to the success of SRASS is the use of Prophet (see Section 5.2) to measure the relevance of the axioms to the conjecture, to determine the order in which axioms are considered.

The implementation of SRASS uses a range of conventional ATP systems to implement the various tests and evaluations required: a theorem prover (currently E/EP 0.99) to test for (counter)theoremhood, test for unsatisfiability, and to find explicit proofs; a finite model builder (currently DarwinFM 1.3g) to test for (counter)satisfiability and build models; and a saturating system (currently SPASS 2.2) to further test for (counter)satisfiability in cases where the finite model builder fails and it is necessary only to establish the existence of a model. The various ATP systems are run under the control of SystemOnTPTP (see Section 5.1).

SRASS was tested in a conservative configuration on several problem sets from the TPTP – problems in logical calculi, problems in set theory, and problems in software verification, all of which are known to have superfluous axioms. In

summary, of the 71 problems that were difficult enough to be eligible for comparative testing, SRASS solves 54 while the underlying ATP system (E/EP 0.99) solves only 39 without the benefit of axiom selection. SRASS was also tested in a less conservative configuration on the MPTP Challenge problems [US06]. In the bushy division of the challenge SRASS solves 171 of the 252 problems, compared to E/EP's 141, and in the chainy division (in which the problems have many more superfluous axioms) SRASS solves 127, compared to E/EP's 91.

7 Applications

The TPTP language and tools have been adopted for a range of user applications. The users employ ATP systems as embedded components of some larger process. By using the TPTP framework the users are not distracted by idiosyncrasies of automated reasoning, and can focus on their application. This section describes three such applications.

7.1 NASA

Research scientists in the Robust Software Engineering Group of the Intelligent Systems Division of NASA Ames have developed, implemented, and evaluated a certification approach that uses Hoare-style techniques to formally demonstrate the safety of aerospace programs that are automatically generated from high-level specifications [DF03, DFS04]. The focus is on automated – as opposed to interactive or (the auto-modes of) tactic-based – systems, since the aim is to have a fully automated push-button tool.

In this work the code generator was extended so that it simultaneously generates code and detailed annotations, e.g., loop invariants, regarding safety conditions. A verification condition generator processes the annotated code, and produces a set of safety obligations in the form of TPTP format problems that are provable if and only if the code is safe. The obligation problems are discharged using SSCPA (see Section 6.1), selecting up to three ATP systems, to produce TPTP format proofs that serve as safety certificates for authorities like the FAA. The derivations are verified by GDV (see Section 5.4). The individual derivations from GDV, which verify each inference step, provide explicit evidence that none of the individual tool components yield incorrect results and, hence, that the certificates are valid.

7.2 MPTP

The goal of the MPTP project [Urb07] is to make the large formal Mizar Mathematical Library (MML) [Rud92] available to current ATP systems (and vice versa), and to boost the development of domain-based, knowledge-based, and generally AI-based ATP methods. The MPTP converts Mizar format problems to a extended TPTP language that adds term-dependent sorts and abstract (Fraenkel) terms to the TPTP syntax. Problems in the extended language are transformed to standard TPTP format using relativization of sorts

and deanonymization of abstract terms. Finding proofs for these problems provides cross verification of the underlying Mizar proofs. It is interesting that some of the ATP proofs correspond to shorter Mizar proofs of the original theorems, and therefore are likely to be used for MML refactoring.

Mizar proofs are also exported, as TPTP format derivations, allowing a number of ATP experiments and use of TPTP tools. An example of this is the combination of the Mizar WWW view with IDV (see Section 5.5) [UTSP07]. This allows a user to view and interact with Mizar level proofs, and to export these to IDV and beyond to view and interact with the corresponding first-order form.

7.3 SUMO and Cyc

In recent years there has been a growing interest in translating large ontological knowledge bases into first-order logic, so that ATP systems can be used to reason over the knowledge. Two examples of this are the translation of the Suggested Upper Merged Ontology (SUMO) [NP01] and of Cyc [MJWD06].

The translation of SUMO [PS07] requires dealing with some apparently and some truly second order constructs, adding guards to impose sort constraints, and converting from SUMO's SUO-KIF language to the TPTP language. Testing on a suite of reasoning tasks provided feedback on what choices in the translation process provide the most easily solved first-order problems. The translation has been integrated into the Sigma ontology development environment [Pea03], with access to IDV (see Section 5.5) for displaying derivations.

The FOLification of Cyc [RPG05] translated about 90% of ResearchCyc into first-order logic, in the TPTP format. As with the SUMO translation, special treatment of higher order constructs was necessary. The translation produced 1,253,117 axioms over 132,116 symbols (not including strings or numbers). This very large background theory presented practical difficulties for using ATP systems. With the exception of E/EP, none of the ATP systems tried were able to load more than 20% of the axioms without failing due to memory errors. This highlighted the need for reengineering of ATP systems, in order to cope with such large problems. A second translation of Cyc is now underway, in order to generate problems that can be added to the TPTP library as challenges to ATP systems.

8 Future

The TPTP and related projects are ongoing efforts, continuously aiming to extend the range and scope of TPTP compliant data and tools. Three main developments are planned for the near future.

Following discussions at the workshop on Empirically Successful Higher Order Logic (ESHOL) [BHS05], a typed higher-order TPTP syntax has been developed - the THF syntax. The THF syntax is divided into levels, starting with a simply typed Church lambda calculus core, and providing three layers of more complex constructs. This layered approach lowers the entry barrier for adopting the THF

syntax, but also provides a rich language for ongoing development. With the syntax in place it is now planned to extend the TPTP and TSTP to include higher-order problems and solutions. The THF effort also resulted in completion of the typed first-order syntax (the TFF syntax), and the two are compatible.

Many applications of automated reasoning, including all those described in Section 7, require some simple reasoning over numbers. An extension of the TPTP language to provide interpreted arithmetic functors and predicates has been designed, aligned with the theory of integers in the Satisfiability Modulo Theories (SMT) library [RT06]. It is planned to extend the TPTP and TSTP to include problems and solutions that involve arithmetic.

The TPTP, TSTP, and tools, provide a stable environment for using ATP systems. While the ability to find solutions to ATP problems is useful directly, in many applications further features are necessary. Two such features are answer variables - the ability to extract an answer to a question that has been framed as a conjecture, and access to provenance information – information regarding information sources, assumptions, learned information, and answers, as an enabler for trust. Preliminary work on these and similar topics is now underway. Cyc's handling of answer variables [MJWD06] provides a starting point for determining the features and capabilities of answer variables in the TPTP, and issues of provenance information are being inspired by the work in the Inference Web [MPdS04].

9 Conclusion

This paper has given an overview of activities and products that stem from the TPTP problem library for ATP systems. The TPTP language, the SZS ontology, and the tools developed, provide an homogeneous environment for ongoing research, development, and application of automated reasoning. Contributions and feedback to improve the TPTP world are always welcome.

Acknowledgements. Many people have contributed to this work. Most salient are: Christian Suttner, the co-developer of the TPTP library and CASC; Stephan Schulz, who influenced the development of the new TPTP language; Allen Van Gelder who wrote the core of the BNF; the ARTists Yi Gao, Yury Puzis, and Steven Trac, who co-designed and implemented some of the tools; Petr Pudlak, who inspired SRASS, Bernd Fischer, Josef Urban, Adam Pease, and the Cyclists at Cycorp, who developed the applications described.

References

[AB04] Abadi, M., Blanchet, B.: Analyzing Security Protocols with Secrecy Types and Logic Programs. Journal of the ACM (2004)

[BFT06] Baumgartner, P., Fuchs, A., Tinelli, C.: Implementing the Model Evolution Calculus. International Journal on Artificial Intelligence Tools 15(1), 21–52 (2006)

[BHS05] Proceedings of the Workshop on Empirically Successful Higher-Order Logic. In: 12th International Conference on Logic for Programming Artificial Intelligence and Reasoning, vol. 0601042 of arXiv (2005)

[CS03] Claessen, K., Sorensson, N.: New Techniques that Improve MACE-style Finite Model Finding. In: Baumgartner, P., Fermueller, C. (eds.) Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications (2003)

[Das06] Das, M.: Formal Specifications on Industrial-Strength Code - From Myth to Reality. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, p. 1. Springer, Heidelberg (2006)

[DF03] Denney, E., Fischer, B.: Correctness of Source-level Safety Policies. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 894–913. Springer, Heidelberg (2003)

[DFS04] Denney, E., Fischer, B., Schumann, J.: Using Automated Theorem Provers to Certify Auto-generated Aerospace Software. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 198–212. Springer, Heidelberg (2004)

[FS005] Fages, F., Soliman, S. (eds.): PPSWR 2005. LNCS, vol. 3703. Springer, Heidelberg (2005)

[Hil03] Hillenbrand, T.: Citius altius fortius: Lessons Learned from the Theorem Prover Waldmeister. In: Dahn, I., Vigneron, L. (eds.) Proceedings of the 4th International Workshop on First-Order Theorem Proving. Electronic Notes in Theoretical Computer Science, vol. 86.1 (2003)

[Lam05] Lam, W.: Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, Englewood Cliffs (2005)

[McC97] McCune, W.W.: Solution of the Robbins Problem. Journal of Automated Reasoning 19(3), 263–276 (1997)

[McC03] McCune, W.W.: Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA (2003)

[Mit05] Mitchell, J.: Security Analysis of Network Protocols: Logical and Computational Methods. In: Barahona, P., Felty, A. (eds.) Proceedings of the 7th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, pp. 151–152 (2005)

[MJWD06] Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In: Baral, C. (ed.) Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, pp. 44–49 (2006)

[MP06] Meng, J., Paulson, L.: Translating Higher-Order Problems to First-Order Clauses. In: Sutcliffe, G., Schmidt, R., Schulz, S. (eds.) Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning, CEUR Workshop Proceedings, vol. 192, pp. 70–80 (2006)

[MPdS04] McGuinness, D., Pinheiro da Silva, P.: Explaining Answers from the Semantic Web: The Inference Web Approach. Journal of Web Semantics 1(4), 397–413 (2004)

[Nie02] Nieuwenhuis, R.: The Impact of CASC in the Development of Automated Deduction Systems. AI Communications 15(2-3), 77–78 (2002)

[NP01] Niles, I., Pease, A.: Towards A Standard Upper Ontology. In: Welty, C., Smith, B. (eds.) Proceedings of the 2nd International Conference on Formal Ontology in Information Systems, pp. 2–9 (2001)

[Pea03] Pease, A.: The Sigma Ontology Development Environment. In: Giunchiglia, F., Gomez-Perez, A., Pease, A., Stuckenschmidt, H., Sure, Y., Willmott, S. (eds.) Proceedings of the IJCAI-03 Workshop on Ontologies and Distributed Systems, CEUR Workshop Proceedings, vol. 71 (2003)

[PGS06] Puzis, Y., Gao, Y., Sutcliffe, G.: Automated Generation of Interesting Theorems. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference, pp. 49–54. AAAI Press, Stanford, California, USA (2006)

[PS07] Pease, A., Sutcliffe, G.: First Order Reasoning on a Large Ontology. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories (2007)

[Qua92] Quaife, A.: Automated Development of Fundamental Mathematical Theories. Kluwer Academic Publishers, Dordrecht (1992)

[RPG05] Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized ResearchCyc: Expressiveness and Efficiency in a Common Sense Knowledge Base. In: Shvaiko, P. (ed.) Proceedings of the Workshop on Contexts and Ontologies: Theory, Practice and Applications (2005)

[RT06] Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2. Technical Report Technical Report, Department of Computer Science, The University of Iowa, Iowa City, USA (2006)

[Rud92] Rudnicki, P.: An Overview of the Mizar Project. In: Proceedings of the 1992 Workshop on Types for Proofs and Programs, pp. 311–332 (1992)

[RV02] Riazanov, A., Voronkov, A.: The Design and Implementation of Vampire. AI Communications 15(2-3), 91–110 (2002)

[Sah06] Sahami, M.: Mining the Web to Determine Similarity Between Words, Objects, and Communities. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference, pp. 14–19. AAAI Press, Stanford, California, USA (2006)

[Sch02] Schulz, S.: E: A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)

[SFS95] Slaney, J.K., Fujita, M., Stickel, M.E.: Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. Computers and Mathematics with Applications 29(2), 115–132 (1995)

[SP07] Sutcliffe, G., Puzis, Y.: SRASS - a Semantic Relevance Axiom Selection System. In: Pfenning, F. (ed.) Proceedings of the 21st International Conference on Automated Deduction. LNCS (LNAI), vol. 4603, pp. 295–310. Springer, Heidelberg (2007)

[SS98] Sutcliffe, G., Suttner, C.B.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)

[SS99] Sutcliffe, G., Seyfang, D.: Smart Selective Competition Parallelism ATP. In: Kumar, A., Russell, I. (eds.) Proceedings of the 12th International FLAIRS Conference, pp. 341–345. AAAI Press, Stanford, California, USA (1999)

[SS01] Sutcliffe, G., Suttner, C.B.: Evaluating General Purpose Automated Theorem Proving Systems. Artificial Intelligence 131(1-2), 39–54 (2001)

[SS06] Sutcliffe, G., Suttner, C.: The State of CASC. AI Communications 19(1), 35–48 (2006)

[SSCVG06] Sutcliffe, G., Schulz, S., Claessen, K., Van Gelder, A.: Using the TPTP Language for Writing Derivations and Finite Interpretations. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 67–81. Springer, Heidelberg (2006)

[Sut00] Sutcliffe, G.: SystemOnTPTP. In: McAllester, D. (ed.) Automated Deduction - CADE-17. LNCS, vol. 1831, pp. 406–410. Springer, Heidelberg (2000)

[Sut06] Sutcliffe, G.: Semantic Derivation Verification. International Journal on Artificial Intelligence Tools 15(6), 1053–1070 (2006)

[SutRL] Sutcliffe, G.: The TSTP Solution Library. http://www.TPTP.org/TSTP

[SZS04] Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In: Zhang, W., Sorge, V. (eds.) Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Frontiers in Artificial Intelligence and Applications, vol. 112, pp. 201–215. IOS Press, Amsterdam (2004)

[TPS06] Trac, S., Puzis, Y., Sutcliffe, G.: An Interactive Derivation Viewer. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of the 7th Workshop on Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning. Electronic Notes in Theoretical Computer Science, vol. 174, pp. 109–123 (2006)

[Urb07] Urban, J.: MPTP 0.2: Design, Implementation, and Initial Experiments. Journal of Automated Reasoning 37(1-2), 21–43 (2007)

[US06] Urban, J., Sutcliffe, G.: The MPTP $100 Challenges (2006), http://www.tptp.org/MPTPChallenge/

[UTSP07] Urban, J., Trac, S., Sutcliffe, G., Puzis, Y.: Combining Mizar and TPTP Semantic Presentation Tools. In: Proceedings of the Mathematical User-Interfaces Workshop 2007 (2007)

[VGS06] Van Gelder, A., Sutcliffe, G.: Extending the TPTP Language to Higher-Order Logic with Automated Parser Generation. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 156–161. Springer, Heidelberg (2006)

[WBH+02] Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topic, D.: SPASS Version 2.0. In: Voronkov, A. (ed.) Automated Deduction - CADE-18. LNCS (LNAI), vol. 2392. Springer, Heidelberg (2002)

Abstract Modeling and Formal Verification of Microprocessors

Ziyad Hanna

ziyad.hanna@intel.com

Abstract. Moore's Law continues to drive a severe increase in the number of transistors that can be integrated onto a single microprocessor chip. Computer architects and designers continue to look for ways to take advantage from it to produce ever more complex microprocessors. Meanwhile, market forces are dictating a shorter time to market, a proliferation of product and steeper volume ramps in production. However, it is evident that logic correctness is one of the main challenges that computer engineers usually face during the design and validation of such systems.

In the last 20 years, researchers and industrial experts invented several modeling and validation technologies, such as logic simulation, fast hardware emulation engines, and formal methods. However the design size and complexity continue to grow and outstrip what the validation techniques can do for producing high quality and correct systems. As a results, the validation problem is becoming more complex to solve and is indeed the main limiter for producing a high quality silicon products.

Design abstraction and high level modeling is a fundamental design strategy to cope with system complexity. The basic idea is to hide design implementation details, while focusing on design specification, capturing the pure logic properties and behaviors of the system. Doing this, we believe that the size of the design model can be dramatically decreased because none functional or physical properties of the design are excluded. Therefore the design representation is purely logical and have a clear semantics which it becomes easier to understand, easier and faster to validate using dynamic or formal techniques, so design errors can be detected earlier before going into detailed implementation and thus avoid a costly design iterations due to late soundness issues.

In this talk we will present abstract modeling techniques and their verification challenges. In particular we will describe the Abstract State Machines approach for modeling and verification of high level models. In addition we will outline several research topics in this domain to encourage the academic community to take an active part in exploring and developing new verification methods that can cope with the increasing complexity of microprocessors' design.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, p. 23, 2007.

Sequences of Level $1, 2, 3, \ldots, k, \ldots$

Géraud Sénizergues

LaBRI and UFR Math-info,
Université Bordeaux1 351 Cours de la libération -33405- Talence Cedex
Fax: 05-56-84-66-69
ges@labri.u-bordeaux.fr
http://dept-info.labri.u-bordeaux.fr/~ges

Abstract. Sequences of numbers (either natural integers, or integers or rational) of level $k \in \mathbb{N}$ have been defined in [FS06] as the sequences which can be computed by deterministic pushdown automata of level k. We extend this definition to sequences of *words* indexed by *words*. We give characterisations of these sequences in terms of "higher-order" L-systems. In particular sequences of rational numbers of level 3 are characterised by polynomial recurrences (which generalize the P-recurrent sequences studied in [Sta80]). The equality problem for sequences of rational numbers of level 3 is shown decidable.

Keywords: Iterated pushdown automata, recurrent sequences equivalence problems.

1 Introduction

The class of pushdown automata of level k (for $k \geq 1$) has been introduced in [Gre70], [Mas74] as a generalisation of the automata and grammars of [Aho68], [Aho69], [Fis68] and has been the object of many further studies: see [Mas76], [ES77], [Dam82], [Eng83], [ES84], [EV86], [DG86], and more recently [Cau02], [KNU02], [CW03], [Fra05].

The class of *integer* sequences computed (in a suitable sense) by such automata was defined in [Fra05], [FS06] (we denote it by $\mathbb{S}_k$).

The class $\mathcal{F}(\mathbb{S}_k)$ consisting of all the sequences of *rational* numbers which can be decomposed as $\frac{a_n - b_n}{a'_n - b'_n}$ for sequences $a, b, a', b' \in \mathbb{S}_k$ was also introduced.
These classes of number sequences fulfil many closure properties and generalize some well-known classes of recurrent sequences (or formal power series). The level 3, for example, contains all the so-called P-recurrent sequences of rational numbers, corresponding also to the D-finite formal power series (see [Sta80] for a survey and [PWZ96] for a thorough study of their algorithmic properties).

We give here several characterisations of the classes $\mathbb{S}_k$ for $k \geq 1$. These characterisations go through generalizations of the above classes $\mathbb{S}_k$ to their analogues for sequences of words, formal power series with non-commutative undeterminates and, finally, mappings from words to words. Let us denote by $\mathbb{S}_k(A^*, B^*)$ the class of mappings from A^* to B^* computed by pushdown automata of level

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 24–32, 2007.

k. The elements of $\mathbb{S}_k(A^*, B^*)$ can be characterised by some kind of Lindenmayer-systems of "order k" that we introduce here. As a corollary, S_3 is characterised by polynomial recurrences. The equality problem for two sequences in $\mathcal{F}(\mathbb{S}_3)$ can thus be solved by a suitable reduction to polynomial ideal theory (namely to the construction of Gröbner bases). The present text is an extended abstract: it gives the main definitions and states the main results but does not provide any formal proof.

2 Preliminaries

We introduce here some notation and basic definitions which will be used throughout the text.

2.1 Automata

Beside the usual notions of finite automaton and pushdown automaton, we shall consider here the notion of *pushdown automaton of level k*. Such automata are an extension of classical pushdown-automata to a storage structure built iteratively. This storage structure can be described as follows:

Definition 1 (k-iterated pushdown store). *Let Γ be a set. We define inductively the set of* k-iterated pushdown-stores over Γ *by:*

$$0-\mathrm{pds}(\Gamma)=\{\varepsilon\}\quad (k+1)-\mathrm{pds}(\Gamma)=(\Gamma[k-\mathrm{pds}(\Gamma)])^*\quad \mathrm{it}-\mathrm{pds}(\Gamma)=\bigcup_{k\geq 0} k-\mathrm{pds}(\Gamma).$$

The elementary operations that a k-pda can perform are:

- pop of level j (where $1 \leq j \leq k$), which consists of popping the leftmost letter of level j and all the structure which is "above" this letter
- push of level j (where $1 \leq j \leq k$), which consists of pushing a new letter C on the left of the leftmost letter D of level j and copying above this new letter C all the structure which was "above" the letter D.

A transition of the automaton consists, given the word γ made of all the leftmost letters of the k-pushdown (the one of level 1, followed by the one of level 2, ..., followed by the one of level k), the state q and the leftmost letter b (or, possibly, the empty word ε) on the input tape, in performing one of the above elementary operations. More formally,

Definition 2 (k-*pdas*). *Let $k \geq 1$, let $POP = \{\mathrm{pop}_j \,|j \in [k]\}$, $PUSH(\Gamma) = \{\mathrm{push}_j(\gamma)|\gamma \in \Gamma^+, j \in [k]\}$, and $TOPSYMS(\Gamma) = \Gamma^{(k)} - \{\varepsilon\}$.*
A k-iterated pushdown automaton over a terminal alphabet B is a 6-tuple $\mathcal{A} = (Q, B, \Gamma, \delta, q_0, Z_0)$ where

- *Q is a finite set of states, $q_0 \in Q$ denoting the initial state,*
- *Γ is a finite set of pushdown-symbols,$Z_0 \in \Gamma$ denoting the initial symbol,*
- *the transition function δ is a map from $Q \times (B \cup \{\varepsilon\}) \times TOPSYMS(\Gamma)$ into the set of finite subsets of $Q \times (PUSH(\Gamma) \cup POP)$ such that:*
  if $(q, \mathrm{push}_j(\gamma)) \in \delta(p, \bar{b}, \gamma)$ then $j \leq |\gamma| + 1$ and if $(q, \mathrm{pop}_j) \in \delta(p, \bar{b}, \gamma)$ then $j \leq |\gamma|$.

(see [FS06] for more details). The automaton $\mathcal{A}$ is said *deterministic* iff, for every $q \in Q, \gamma \in \Gamma^{(k)}, b \in B$

$$\mathrm{Card}(\delta(q, \varepsilon, \gamma)) \leq 1 \text{ and } \mathrm{Card}(\delta(q, b, \gamma)) \leq 1, \tag{1}$$

$$\mathrm{Card}(\delta(q, \varepsilon, \gamma)) = 1 \Rightarrow \mathrm{Card}(\delta(q, b, \gamma)) = 0. \tag{2}$$

In order to define a useful notion of map *computed* by a k-pda we introduce the following stronger condition: $\mathcal{A}$ is called *strongly deterministic* iff, for every $q \in Q, \gamma \in \Gamma^{(k)}$,

$$\sum_{\bar{b} \in \{\varepsilon\} \cup B} \mathrm{Card}(\delta(q, \bar{b}, \gamma)) \leq 1. \tag{3}$$

In other words, the automaton $\mathcal{A}$ is *strongly deterministic* iff, the leftmost contents γ of the memory and the state q completely determine the transition of $\mathcal{A}$, in particular what letter b (or possibly the empty word) can be read. Therefore, such an automaton $\mathcal{A}$ can accept at most one word w from a given configuration. We say that $\mathcal{A}$ is *level-partitioned* iff Γ is the disjoint union of subsets $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ such that, in every transition of $\mathcal{A}$, every occurrence of a letter from Γ_i is at level i. It is easy to transform any k-pushdown automaton $\mathcal{A}$ into another one $\mathcal{A}'$ which recognizes the same language and is level-partitioned. Moreover, if $\mathcal{A}$ is strongly deterministic then $\mathcal{A}'$ is strongly deterministic.

Definition 3 (k-computable mapping). *A mapping $f : A^* \mapsto B^*$ is called* k-computable *iff there exists a strongly deterministic k-pda $\mathcal{A}$, over a pushdown-alphabet Γ which is level-partitioned, such that Γ contains $k-1$ symbols $U_1, U_2, \ldots, U_{k-1}$, the alphabet A is a subset of Γ_k and for all $w \in A^*$:*

$$(q_0, f(w), U_1[U_2 \ldots [U_{k-1}[w]] \ldots]) \vdash_{\mathcal{A}}^* (q_0, \varepsilon, \varepsilon).$$

One denotes by $\mathbb{S}_k(A^*, B^*)$ the set of k-computable mappings from A^* to B^*.

The particular case where $\mathrm{Card}(A) = \mathrm{Card}(B) = 1$ was studied in [FS06].

2.2 Number Recurrences

Definition 4 ($\mathbb{N}$-rational formal power series). *A mapping $f : A^* \to \mathbb{N}$ is $\mathbb{N}$-rational iff there is an homomorphism $M : A^* \to \mathbb{N}^{d \times d}$ and two vectors L in $\mathbb{B}^{1 \times d}$ and T in $\mathbb{B}^{d \times 1}$ such that, for every $w \in A^*$*

$$f(w) = L \cdot M(w) \cdot T. \tag{4}$$

The map f can also be denoted by $\sum_{w \in A^*} f(w) \cdot w$ which explains our terminology.

Definition 5 (Polynomial recurrent relations). *Given a finite index set* $I = [1, n]$ *and a family of mappings* $f_i : A^* \to \mathbb{N}$ *(for* $i \in I$*), we call* system of polynomial recurrent relations *a system of the form*

$$f_i(aw) = P_{i,a}(f_1(w), f_2(w), \ldots, f_n(w)) \text{ for all } i \in I, a \in A, w \in A^*$$

where $P_{i,a} \in \mathbb{N}[X_1, X_2, \ldots, X_n]$.

A similar definition can be given for mappings $f_i : A^* \to \mathbb{Z}$ (for $i \in I$) and polynomials $P_{i,a} \in \mathbb{Z}[X_1, X_2, \ldots, X_n]$.

2.3 Word Recurrences

When considering mappings into *words* instead of integers, one is lead to consider the following kind of recurrent relations.

Definition 6 (catenative recurrent relations). *Given a finite index set* $I = [1, n]$ *and a family of mappings* $f_i : A^* \to B^*$ *(for* $i \in I$*), we call* system of catenative recurrent relations *a system of the form*

$$f_i(aw) = \prod_{j=1}^{\ell(i,a)} f_{\alpha(i,a,j)}(w) \text{ for all } i \in I, a \in A, w \in A^*$$

where $\ell(i, a) \in \mathbb{N}, \alpha(i, a, j) \in I$.

One can check that, in the case where B is reduced to one letter, a mapping $f : A^* \to B^*$ is the first mapping of a family fulfilling a system of catenative recurrent relations iff f is a rational series.

2.4 Recurrences in a Monoid

Mezei and Wright developed a general notion of grammar defining languages in general algebras ([MW67]). These ideas lead naturally to the following adaptation to arbitrary monoids of the notion of catenative recurrent relations. Let $(M, \cdot, 1)$ be some monoid.

Definition 7 (recurrent relations in M**).** *Given a finite index set* $I = [1, n]$ *and a family of mappings* $f_i : A^* \to M$ *(for* $i \in I$*), we call* system of recurrent relations in M *a system of the form*

$$f_i(aw) = \prod_{j=1}^{\ell(i,a)} f_{\alpha(i,a,j)}(w) \text{ for all } i \in I, a \in A, w \in A^*$$

where $\ell(i, a) \in \mathbb{N}, \alpha(i, a, j) \in I$ *and the symbol* $\prod$ *stands for the extension of the binary product in* M *to an arbitrary finite number of arguments.*

The monoids $(Hom(C^*, C^*), \circ, \mathrm{Id})$, for finite alphabets C, will be of particular interest for studying mappings of level $k \geq 3$.

2.5 L-Systems

The following notion ([KRS97]) turns out to be crucial for describing all k-computable mappings as compositions of simpler mappings.

Definition 8 (HDT0L sequences). *Let $f : A^* \to B^*$. The mapping f is called a HDT0L sequence iff there exists a finite alphabet C, a homomorphism $H : A^* \to Hom(C^*, C^*)$, an homomorhism $h \in Hom(C^*, B^*)$ and a letter $c \in C$ such that, for every $w \in A^*$*

$$f(w) = h(H^w(c)).$$

(here we denote by H^w the image of w by H). The mapping f is called a DT0L when $B = C$ and the homomorphism h is just the identity; f is called a HD0L when A is reduced to one element.

3 Sequences of Level 1

Let us mention, just for sake of completeness, the description of level 1 of our hierarchy of mappings.

Theorem 1. *The elements of $\mathbb{S}_1(A^*, B^*)$ are exactly the generalized sequential mappings from A^* to B^*.*

4 Sequences of Level 2

N. Marin has shown in her Master thesis ([Mar07]) that $\mathbb{S}_2(A^*, B^*)$ has several nice characterisations.

Theorem 2 ([Mar07]). *Let us consider a mapping $f : A^* \to B^*$. The following properties are equivalent:*

1- $f \in \mathbb{S}_2(A^*, B^*)$
2- There exists a finite family $(f_i)_{i\in[1,n]}$ of mappings $A^* \to B^*$ which fulfils a system of catenative recurrent relations and such that $f = f_1$
3- f is a HDT0L sequence.

This theorem specializes as follows in the particular cases where A or B is reduced to one letter.

Corollary 1. *Let us consider a mapping $f : A^* \to \mathbb{N}$. The following properties are equivalent:*

1- $f \in \mathbb{S}_2(A^*, \mathbb{N})$
2- f is a $\mathbb{N}$-rational power series with non-commutative undeterminates in A

Corollary 2. *Let us consider a mapping $f : \mathbb{N} \to B^*$. The following properties are equivalent:*

1- $f \in \mathbb{S}_2(\mathbb{N}, B^*)$
2- There exists a finite family $(f_i)_{i\in[1,n]}$ of sequences $\mathbb{N} \to B^*$ which fulfils a system of catenative recurrent relations and such that $f = f_1$
3- f is a HD0L sequence.

Since J. Honkala has proved that the equivalence for HDT0L sequences is decidable ([Hon00]), theorem 2 implies

Theorem 3. *The equality problem is decidable for mappings in $\mathbb{S}_2(A^*, B^*)$.*

5 Sequences of Level 3

Theorem 4. *Let us consider a mapping $f : A^* \to B^*$. The following properties are equivalent:*

1- $f \in \mathbb{S}_3(A^*, B^*)$
2- There exists a finite family $(H_i)_{i\in[1,n]}$ of mappings $A^* \to Hom(C^*, C^*)$ which fulfils a system of recurrent relations in $(Hom(C^*, C^*), \circ, \mathrm{Id})$, an element $h \in Hom(C^*, B^*)$ and a letter $c \in C$ such that, for every $w \in A^*$:

$$f(w) = h(H_1(w)(c)).$$

3- f is a composition of a DT0L sequence $g : A^* \to C^*$ by a HDT0L sequence $h : C^* \to B^*$.

This theorem specializes as follows in the particular cases where B is reduced to one letter i.e. when the mapping f is a formal power series.

Corollary 3. *Let us consider a mapping $f : A^* \to \mathbb{N}$. The following properties are equivalent:*

1- $f \in \mathbb{S}_3(A^*, \mathbb{N})$
2- There exists a finite family $(H_i)_{i\in[1,n]}$ of mappings $A^* \to Hom(C^*, C^*)$ which fulfils a system of recurrent relations in $(Hom(C^*, C^*), \circ, \mathrm{Id})$, an element $h \in Hom(C^*, \mathbb{N})$ and a letter $c \in C$ such that, for every $w \in A^*$:

$$f(w) = h(H_1(w)(c)).$$

3- f is composition of a DT0L sequence $g : A^* \to C^*$ by a rational series $h : C^* \to \mathbb{N}$.
4- There exists a finite family $(f_i)_{i\in[1,n]}$ of mappings $A^* \to \mathbb{N}$ fulfilling a system of polynomial recurrent relations and such that $f = f_1$.

Definition 9. *Let $\mathbb{S}$ be a set of mappings $A^* \to \mathbb{N}$. We denote by $\mathcal{D}(\mathbb{S})$ the set of mappings of the form:*

$$f(w) = g(w) - h(w) \quad \textit{for all } w \in A^*,$$

for some mappings $g, h \in \mathbb{S}$. We denote by $\mathcal{F}(\mathbb{S})$ the set of mappings of the form:

$$f(w) = \frac{g(w) - h(w)}{f'(w) - g'(w)} \quad \text{for all } w \in A^*,$$

for some mappings $f, g, f', g' \in \mathbb{S}$.

Using point 4 of corollary 3 we can prove the following

Theorem 5. *The equality problem is decidable for formal power series in $\mathcal{F}(\mathbb{S}_3(A^*, \mathbb{N}))$.*

The method consists, in a way similar to [Sén99] or [Hon00], in reducing such an equality problem to deciding whether some polynomial belongs to the ideal generated by a finite set of other polynomials.

6 Sequences of Level k

Theorem 6. *Let us consider a mapping $f : A^* \to B^*$. The following properties are equivalent:*

1- $f \in \mathbb{S}_k(A^*, B^*)$
2- f is a composition of $k-1$ HDT0L sequences $g_1 : A^* \to C_1^*, \ldots, g_i : C_{i-1}^* \to C_i^*, \ldots, g_{k-1} : C_{k-2}^* \to B^*$. Moreover the $g_1, \ldots, g_{k-2}$ can be chosen to be DT0L's.

From this theorem follows easily the fact that the inclusion $S_k(\mathbb{N}, \mathbb{N}) \subset S_{k+1}(\mathbb{N}, \mathbb{N})$ is strict.

7 Examples and Counter-Examples

We examine here four examples of mappings $A^* \to \mathbb{N}$ and locate them in the classes $\mathbb{S}_k$ or some related classes.

Example 1. The Fibonacci sequence F_n defined by

$$F_0 = 1, F_1 = 1, F_{n+2} = F_{n+1} + F_n \text{ for all } n \geq 0$$

is clearly in $\mathbb{S}_2$ since $\sum_{n=0}^{\infty} F_n X^n$ is a rational series.

Example 2. Let $G : \{0,1\}^* \to \mathbb{N}$ be defined by

$$G(w) = F_{\nu(w)}$$

where $\nu(w)$ is the natural number expressed by w in base 2. Since $\sum_{n=0}^{\infty} \nu(w) w$ is a rational series, G fulfils point 3 of our characterisation of $\mathbb{S}_3(\{0,1\}^*, \mathbb{N})$.

Example 3. Let us consider the sequence of Catalan numbers: $C_n = \frac{1}{n+1}C_{2n}^n$. This sequence is not residually ultimately periodic ([Ber03]) while we can prove that every sequence in $\bigcup_{k\in\mathbb{N}} \mathbb{S}_k(\mathbb{N}, \mathbb{N})$ is residually ultimately periodic. By the same arguments $(C_n)_{n\in\mathbb{N}}$ cannot belong to $\mathcal{D}(\mathbb{S}_k)$ for any $k \geq 1$. Nevertheless $(C_n)_{n\in\mathbb{N}}$ belongs to $\mathcal{F}(\mathbb{S}_3)$.

Example 4. Let us consider the sequence $D_n = n^n$. This sequence belongs to $\mathbb{S}_4$. It does not belong to $\mathcal{F}(\mathbb{S}_3)$ because, for every $r \geq 1$, the sequences $(D_n)_{n\in\mathbb{N}}, (D_{n+1})_{n\in\mathbb{N}}, \ldots, (D_{n+r-1})_{n\in\mathbb{N}}$ are algebraically independent over $\mathbb{Q}$.

References

[Aho68] Aho, A.V.: Indexed grammars-an extension of context-free grammars. J. Assoc. for Comput. Mach. 15, 647–671 (1968)

[Aho69] Aho, A.V.: Nested stack automata. J. Assoc. for Comput. Mach. 16, 383–406 (1969)

[Ber03] Berstel, J.: Personnal communication (2003)

[Cau02] Caucal, D.: On infinite terms having a decidable monadic theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)

[CW03] Carayol, A., Wöhrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)

[Dam82] Damm, W.: The IO- and OI-hierarchies. TCS 20, 95–207 (1982)

[DG86] Damm, W., Goerdt, A.: An automata-theoretical characterization of the OI-hierarchy. Information and control 71, 1–32 (1986)

[Eng83] Engelfriet, J.: Iterated pushdown automata and complexity classes. In: Theory Comput, Assoc. Comp. Mach., Proc. 15th Annu. ACM Sympos., pp. 365–373. ACM Press, New York (1983)

[ES77] Engelfriet, J., Schmidt, E.M.: IO and OI. I. J. Comput. System Sci. 15(3), 328–353 (1977)

[ES84] Engelfriet, J., Slutski, G.: Extended macro grammars and stack controlled machines. Journal of Computer and System sciences 29, 366–408 (1984)

[EV86] Engelfriet, J., Vogler, H.: Pushdown machines for the macro tree transducer. TCS 42, 251–368 (1986)

[Fis68] Fischer, M.J.: Grammars with macro-like productions. PhD thesis, Harvard University (1968). See also: In: Proc. 9th Symp. on Switching and Automata Theory, pp. 131–142 (1968)

[Fra05] Fratani, S.: Automates à piles de piles.. de piles. PhD thesis, Bordeaux 1 university (2005)

[FS06] Fratani, S., Sénizergues, G.: Iterated pushdown automata and sequences of rational numbers. Ann. Pure Appl. Logic 141(3), 363–411 (2006)

[Gre70] Greibach, S.: Full AFL's and nested iterated substitution. Information and Control, 7–35 (1970)

[Hon00] Honkala, J.: A short solution for the HDT0L sequence equivalence problem. Theoret. Comput. Sci. 244(1-2), 267–270 (2000)

[KNU02] Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: FoSSaCs 2002. LNCS, Springer, Heidelberg (2002)

[KRS97] Kari, L., Rozenberg, G., Salomaa, A.: L systems. In: Handbook of formal languages, vol. 1, pp. 253–328. Springer, Heidelberg (1997)

[Mar07] Marin, N.: Suites de mots et automates. Master thesis. Bordeaux 1 university, pp. 1–40 (2007)

[Mas74] Maslov, A.N.: The hierarchy of indexed languages of an arbitrary level. Soviet. Math. Dokl. 15, 1170–1174 (1974)

[Mas76] Maslov, A.N.: Multilevel stack automata. Problemy Peredachi Informatsii 12, 38–43 (1976)

[MW67] Mezei, J., Wright, J.B.: Algebraic automata and context-free sets. Information and Control 11, 3–29 (1967)

[PWZ96] Petkovšek, M., Wilf, H.S., Zeilberger, D.: $A = B$. A K Peters Ltd., Wellesley, MA (1996)

[Sén99] Sénizergues, G.: T(A) = T(B)? In: van Emde Boas, P., Wiedermann, J., Nielsen, M. (eds.) Proceedings ICALP'99, vol. 1644, pp. 665–675. Lecture Notes in Computer Science (1999). Detailed version in technical report 1209-99, pp. 1–61. Can be accessed at http://www.labri.u-bordeaux.fr/~ges

[Sta80] Stanley, R.P.: Differentiably finite power series. European Journal of Combinatorics 1, 175–188 (1980)

Timers and Proximities for Mobile Ambients

Bogdan Aman[2] and Gabriel Ciobanu[1,2]

[1] "A.I.Cuza" University, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
[2] Romanian, Academy, Institute of Computer Science
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Abstract. In this paper we extend mobile ambients with timers and proximities, and so we get a clear notion of location and mobility. Timers define timeouts for various resources, making them available only for a determined period of time; we add timers to ambients and capabilities. We present an example how the new model is working. The coordination of the ambients in time and space is given by assigning specific values to timers, and by a set of coordination rules.

1 Introduction

The open systems with mobile entities, and dynamically re-configurable evolving systems accept uncertainty and distribution, leading to difficult control schemes where decision-making are distributed widely and unexpectedly. Among the several challenges related to the coordination aspects, we can mention the mechanisms for delivery of adequate responses to time-critical demands, coordination methods that for real-time systems, a decentralized control take into account the timed resources. Different application contexts exhibit different needs with respect to coordination, and the choice of a coordination model is likely to have an important impact in the design of component-based applications.

A coordination model should integrate a number of (heterogeneous) components (processes, objects, agents) such that the resulting system can execute as a whole, resulting a flexible system even in an open distributed environment. A coordination model is a high-level interaction abstraction used to globally rule the behaviour of different components in a system. Such a coordination model provides a formal framework in which the interaction can be expressed, and the dynamics of an open system ensures synchronized actions, flexibility, and a sound behaviour. The coordination rules define how the actions are handled when the components interact. These rules can be defined in terms of a coordination language. The first comprehensive survey on coordination models and languages has been given in [8]. More recently, in [1,7], the authors survey the state of art in coordination models for agent systems.

We intend to present a model where time is an active controlling input in the execution of the model, and the coordination rules explicitly depend on time. Time must advance to the desired moment when an event is allowed to take place.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 33–43, 2007.

We do not use an absolute time in our approach, but a relative time given by timers. The global clock advances the time, and the interactions happen whenever the involved resources are available. Using such a time control, we can achieve the concurrency of the components, and we can select between different choices in the system evolution. Time as a common interaction requirement provides a natural and flexible synchronization technique able to integrate and regulate dynamically the possible evolutions of the components.

We consider the following example which can motive and illustrate the time-driven model. We assume a student finishing the lectures, moving out of the university building and looking for an available vehicle in order to move to a given place. In front of the university there is a tram stop, a bus stop, and certain taxi-cabs. The student has the possibility to use any of the three types of vehicles: bus, tram and cab to reach the target location. Since the three variants (vehicles) have different costs, the student establishes a priority among them: the bus has the highest priority, followed by the tram, and finally the cab. The bus and the tram are moving according to a predetermined scheduler which is known by the systems (this knowledge is given by the use of a global clock). Our scenario involves a bus, a tram and two cabs (a hired cab and an available cab). This example involves space, mobility, time, (bus and tram) capacity, hired and available resources at a certain moment. In order to avoid the situation in which a student exits the moving vehicle we impose some resource constraints over the city.

Our approach is aimed to provide a faithful syntactic description, and a flexible dynamic semantics for such an example. Time represents a primary notion in the new time-driven coordination model. Since space is dual to time, we think of a model where the notions of location and space can be explicitly described. We define a model as an extension with timers of the pure mobile ambients. Timers define timeouts for various resources, making them available only for a determined period of time; thus we add timers to ambients and capabilities.

2 Mobile Ambients with Time Constraints

Ambient calculus is a formalism for describing distributed and mobile computation in terms of ambients. In contrast with other formalisms for mobile processes such as the π-calculus [6] whose computational model is based on the notion of *communication*, the ambient calculus is based on the notion of *movement*. An ambient, which is a named location, is the unit of movement. Ambients mobility is controlled by the capabilities *in, out*, and *open*. In an ambient we have processes which may exchange messages.

We introduce timed Mobile Ambients (tMA) where capabilities and ambients are used as temporal resources. A timer Δt of each temporal resource makes the resource available only for a determined period of time t. We add timers to ambients and capabilities.

A novelty of this approach is that a location, represented by an ambient, can disappear. We denote by $n^{\Delta t}_{(l,h,r)}[P]^{\mu}$ the fact that an ambient n has assigned a timer Δt, a *capacity* l representing the number of its free resources, a *weight* h

representing the number of resources allocated in the parent ambient, a radius r of its proximity, while the tag μ is a neutral tag that indicates if an ambient is active (a) or passive (s). If $t > 0$ the ambient behaves exactly as in untimed mobile ambients. Since the timer Δt can expire ($t = 0$) we use a pair $(n^{\Delta t}_{(l,h,r)}[P]^{\mu}, k \lhd Q)$ to denote a timed ambient, where Q is a *safety process*. The prefix $k\lhd$ represents the number of resources needed by process Q to be executed. $k \lhd Q$ means that process Q cannot be executed unless it has k free resources available. If nothing happens in t units of time, the ambient n is dissolved, process P running inside the ambient is reduced to $\mathbf{0}$, and process $k \lhd Q \mid \Diamond_h$ is executed. By $\Diamond_h$ we denote that h resources become available in the parent ambient of the dissolved ambient n. If $Q = \mathbf{0}$ we can simply write $n^{\Delta t}_{(l,h,r)}[P]^{\mu}$ instead of $(n^{\Delta t}_{(l,h,r)}[P]^{\mu}, k \lhd Q)$. If we want to simulate the behaviour of untimed mobile ambients, then we use ∞ instead of Δt, and 0 instead of h, k, l and r.

The process $open^{\Delta t} n.(P, k \lhd Q)$ evolves to P whenever, in the period of time Δt, the process becomes sibling to an ambient n; otherwise evolves to $k \lhd Q$. The process $!\langle m\rangle^{\Delta t}.(P, k \lhd Q)$ evolves to P whenever, in the period of time Δt, the process becomes sibling to a process which is willing to capture the name m; otherwise evolves to $k \lhd Q$.

The syntax of the timed Mobile Ambients is defined in the following table.

Table 1. Syntax of tMA

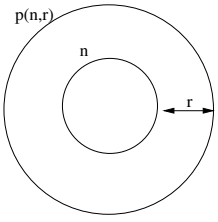
n, m, p	names		$P, Q ::=$		processes
a, s	tags			$\mathbf{0}$	inactivity
M	$::=$	capabilities		$M^{\Delta t}.(P, k \lhd Q)$	movement
	$in\ n$	can enter n		$(n^{\Delta t}_{(l,h,r)}[P]^{\mu}, k \lhd Q)$	ambient
	$out\ n$	can exit n		$P \mid Q$	composition
	$open\ n$	can open n		$(\nu n)P$	restriction
				$*(k \lhd P)$	replication
				$P + Q$	choice

Replication creates new processes, so we use $k \lhd P$ to avoid the execution of process P until it has k free resources. We define a weight function $weight$ which counts the resources needed by a process to be executed:

$$weight(P) = \begin{cases} h & \text{if } P = (n^{\Delta t}_{(l,h,r)}[R]^{\mu}, k \lhd Q) \\ weight(R) & \text{if } P = (\nu n)R \\ weight(R) + weight(Q) & \text{if } P = R \mid Q \\ max\{weight(R), weight(Q)\} & \text{if } P = R + Q \\ 0 & \text{otherwise} \end{cases}$$

The above function takes into account only the resources occupied by ambients; capabilities, replication and $k \lhd Q$ have no need of resources to be executed.

The proximity of an ambient n is defined by a function p which returns a set of points surrounding the ambient n with a given radius r. The radius r is established at the beginning of the computation for each ambient. By $p(n, r)$ we understand the proximity of ambient n of radius r, as in the next figure:



This proximity function has the following properties:

1. $0 \leq s < t$ implies $p(n,s) \subset p(n,t)$ for any ambient n and numbers s and t;
2. $m \subset p(n,s)$ implies $\exists t > 0$ such that $p(m,t) \subset p(n,s)$ for any ambients m, n and positive numbers s and t.

When we describe initially the ambients, we consider that all ambients are active, and we associate the tag a to them.

2.1 Semantics

The timed Mobile Ambients use discrete time. The passage of time is described by a(discrete) time-stepping function ϕ_Δ defined over the set $\mathcal{P}$ of tMA-processes. The possible actions are performed at every tick of a universal clock. ϕ_Δ affects the ambients and the capabilities which are not consumed. The consumed capabilities disappear together with their timers. If a capability or ambient has the timer equal to ∞, we use the equality $\infty - 1 = \infty$ when applying the time-stepping function ϕ_Δ. This function modifies a process accordingly with the global passage of time. Another property of the global time progress function ϕ_Δ is that the passive ambients can become active in the next unit of time in order to participate in other reductions.

Definition 1. *We define the time-stepping function* $\phi_\Delta : \mathcal{P} \to \mathcal{P}$ *by*

$$\phi_\Delta(P) = \begin{cases} M^{\Delta(t-1)}.(R, k \lhd Q) & \text{if } P = M^{\Delta t}.(R, k \lhd Q),\ t > 0 \\ k \lhd Q & \text{if } P = M^{\Delta t}.(R, k \lhd Q),\ t = 0 \\ \phi_\Delta(R) \mid \phi_\Delta(Q) & \text{if } P = R \mid Q \\ (\nu n)\phi_\Delta(R) & \text{if } P = (\nu n)R \\ (n^{\Delta(t-1)}_{(l,h,r)}[\phi_\Delta(R)]^a, k \lhd Q) & \text{if } P = (n^{\Delta t}_{(l,h,r)}[R]^\mu, k \lhd Q),\ t > 0 \\ k \lhd Q \mid \Diamond_h & \text{if } P = (n^{\Delta t}_{(l,h,r)}[R]^\mu, k \lhd Q),\ t = 0 \\ P & \text{if } P = *R \text{ or } P = \mathbf{0} \text{ or } P = k \lhd Q \\ \phi_\Delta(R) + \phi_\Delta(Q) & \text{if } P = R + Q \end{cases}$$

To see how this function is used, observe the reduction rules (Table 3).

The semantics of the timed mobile ambients is given by two relations: structural congruence relation and reduction relation. The *structural congruence relation* $P \equiv_p Q$ relates different syntactic representations of the same process; it is used to define the reduction relation. The *reduction relation* $P \to Q$ describes the evolution of processes.

Processes are grouped into equivalence classes by $\equiv_p$. This relation provides a way of re-arranging expressions such that the interacting parts can be brought together. The structural relation $\equiv_p$ over the timed mobile processes is the least relation satisfying the axioms and rules from the following Table:

Table 2. Structural congruence

(S-Refl)	$P \equiv_p P$ **(S-Sym)** $P \equiv_p Q$ implies $Q \equiv_p P$
(S-Trans)	$P \equiv_p R$, $R \equiv_p Q$ implies $P \equiv_p Q$
(S-Res)	$P \equiv_p Q$ implies $(\nu n)P \equiv_p (\nu n)Q$
(S-LPar)	$P \equiv_p Q$ implies $R \mid P \equiv_p R \mid Q$
(S-RPar)	$P \equiv_p Q$ implies $P \mid R \equiv_p Q \mid R$
(S-Repl)	$P \equiv_p Q$ implies $*(k \lhd P) \equiv_p *(k \lhd Q)$
(S-Amb)	$P \equiv_p Q$ and $R \equiv_p R'$ implies $(n^{\Delta t}_{(l,h,r)}[P], k \lhd R) \equiv_p (n^{\Delta t}_{(l,h,r)}[Q], k \lhd R')$
(S-Cap)	$P \equiv_p Q$ and $R \equiv_p R'$ implies $M^{\Delta t}.(P, k \lhd R) \equiv_p M^{\Delta t}.(Q, k \lhd R')$
(S-Par Com)	if $weight(P) = 0$ then $P \mid Q \equiv_p Q \mid P$
(S-Par Assoc)	$(P \mid Q) \mid R \equiv_p P \mid (Q \mid R)$
(S-Repl Par)	$*(k \lhd P) \equiv_p k \lhd P \mid *(k \lhd P)$
(S-Res Res)	$(\nu n)(\nu m)P \equiv_p (\nu m)(\nu n)P$ if $n \neq m$
(S-Res LPar)	$(\nu n)(P \mid Q) \equiv_p P \mid (\nu n)Q$ if $(n) \notin fn(P)$
(S-Res RPar)	$(\nu n)(P \mid Q) \equiv_p (\nu n)P \mid Q$ if $(n) \notin fn(Q)$
(S-Res Amb)	$(\nu n)(m^{\Delta t}_{(l,h,r)}[P], k \lhd Q) \equiv_p (m^{\Delta t}_{(l,h,r)}[(\nu n)P], k \lhd Q)$ if $n \neq m$
(S-Zero Par)	$P \mid \mathbf{0} \equiv_p P$ **(S-Zero Res)** $(\nu n)\mathbf{0} \equiv_p \mathbf{0}$ **(S-Zero Repl)** $*\mathbf{0} \equiv_p \mathbf{0}$

For the process $M^{\Delta t}.(P, Q)$ the timers of P are activated only after the consumption of $M^{\Delta t}$. To preserve the timers of P, we introduce a function which prevents the application of the time-stepping function for P.

We denote by $\not\rightarrow$ the fact that none of the rules from the following Table, except rule **(R-GTProgress)** can be applied. Because the safety process does not change when applying the reduction rules we shall omit it from the following table in order to simplify the syntax. The behaviour of processes is given by the reduction rules.

In the rules **(R-In)**, **(R-Out)**, **(R-Open)** ambient m can be *passive* or *active*, while in the rules **(R-In)**, **(R-Out)** ambient n is *active*. The difference between *passive* and *active* ambients is that the *passive* ambients can be used in several reductions in a unit of time, while the *active* ambients can be used in at most one reduction in a unit of time, by consuming their capabilities. In the rules **(R-In)**, **(R-Out)** the *active* ambient n becomes *passive*, forcing it to consume only one capability in one unit of time.

In timed mobile ambients, if one process evolves by one of the rules **(R-In)**, **(R-Out)**, **(R-Open)**, while another one does not perform any reduction, then one of the rules **(R-LPar)**, **(R-RPar)** should be applied. If more than one process evolve in parallel by applying one of the rules **(R-In)**, **(R-Out)**, **(R-Open)**, then the rule **(R-Par)** should be applied. When all the ambients become passive, the rule **(R-GTProgress)** is applied to simulate the global passage of time, changing all the p tags to a, and so permitting the ambients to participate in other reductions in the next unit of time.

Table 3. Reduction rules

(R-Free) $\dfrac{-}{n^{\Delta t}_{(l,h,r)}[R \mid \Diamond_k]^\mu \rightarrow n^{\Delta t}_{(l+k,h,r)}[\ R\ ]^\mu}$

(R-Alloc) $\dfrac{k \leq l}{n^{\Delta t}_{(l,h,r)}[k \lhd Q]^\mu \rightarrow n^{\Delta t}_{(l-k,h,r)}[Q]^\mu}$

(R-In) $\dfrac{h' \leq l'',\ n \in p(m,r'')}{n^{\Delta t'}_{(l',h',r')}[in^{\Delta t} m.(P,P') \mid Q]^a \mid m^{\Delta t''}_{(l'',h'',r'')}[R]^\mu \rightarrow m^{\Delta t''}_{(l''-h',h'',r'')}[n^{\Delta t'}_{(l',h',r')}[P \mid Q]^s \mid R]^\mu}$

(R-Out) $\dfrac{h'' \leq l,\ p(n,r'') \cap p(m,r') \neq \emptyset}{p^{\Delta t}_{(l,h,r)}[m^{\Delta t'}_{(l',h',r')}[n^{\Delta t''}_{(l'',h'',r'')}[out^{\Delta t} m.(P,P') \mid Q]^a \mid R]^\mu]^\mu \rightarrow p^{\Delta t}_{(l-h'',h,r)}[n^{\Delta t''}_{(l'',h'',r'')}[P \mid Q]^s \mid m^{\Delta t'}_{(l'+h'',h',r')}[R]^\mu]^\mu}$

(R-Open) $\dfrac{-}{m^{\Delta t'}_{(l',h',r')}[open^{\Delta t} n.\ (P,P') \mid n^{\Delta t''}_{(l'',h'',r'')}[Q]^\mu]^\mu \rightarrow m^{\Delta t'}_{(l'+l'',h',r')}[P \mid Q]^\mu}$

(R-Res) $\dfrac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q}$ **(R-LPar)** $\dfrac{P \rightarrow Q}{R \mid P \rightarrow R \mid Q}$

(R-Amb) $\dfrac{P \rightarrow Q}{n^{\Delta t}_{(l,h,r)}[P]^\mu \rightarrow n^{\Delta t}_{(l,h,r)}[Q]^\mu}$ **(R-RPar)** $\dfrac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}$

(R-Par) $\dfrac{P \rightarrow Q,\ P' \rightarrow Q'}{P \mid P' \rightarrow Q \mid Q'}$ **(R-Struct)** $\dfrac{P' \equiv_p P,\ P \rightarrow Q,\ Q \equiv_p Q'}{P' \rightarrow Q'}$

(R-Choice) $\dfrac{P \rightarrow P'}{P + Q \rightarrow P'}$ and $\dfrac{Q \rightarrow Q'}{P + Q \rightarrow Q'}$ **(R-GTProgress)** $\dfrac{P \not\rightarrow}{P \rightarrow \phi_\Delta(P)}$

Proposition 1. *If $P \equiv_p Q$ then $weight(P) = weight(Q)$.*

The resources are preserved through reduction only in a *closed* system, namely a system which is surrounded by an ambient which cannot be opened, and any ambient can pass through it. In an *open* system, the resources are not preserved through reduction because a process may acquire new resources from the environment, or transfer resources to the environment. Some resources may become restricted, and so unavailable for any other process, e.g. $(\nu n)(n^{\Delta t}_{(l,h,r)}[P]^\mu, k \lhd Q)$.

3 Interaction of the Timed Mobile Ambients

Let us consider that the example involves a bus, a tram and two cabs. Each of these entities is encoded in an ambient having a corresponding label. A *student* has the possibility to use any of the three transports *bus*, *tram* and *cab* to reach a given location. Because the three variants of transport have different costs, the student establishes a priority among them: the *bus* has the highest priority, followed by the *tram*, and finally the lowest priority is the *cab*. For an ambient A, $free\_resources(A)$ represents the (dynamically evolving) capacity l of A.

If *bus* is in the proximity of radius r of the *student* ($bus \subset p(student, r)$), and it has free resources ($free\_resource(bus) > 0$) then the *student* enters the *bus*. If the *bus* does not have free resources and the *tram* is in the proximity of radius r of the *student*, and it has free resources ($free\_resources(tram) > 0$) then the *student* enters the *tram*. If the *tram* does not have free resources and the *cab* is in the proximity of the *student*, and it has free resources ($free\_resources(cab) > 0$) then the *student* enters the *cab*. If the *tram* is not in the proximity of the *student*, the time needed for the *tram* to enter that proximity is greater that the time the *student* decided to wait for the *tram*, the *cab* is in the proximity of the *student* having free resources $free\_resources(cab) > 0$, then the *student* enters the *cab*.

If *bus* is not in the proximity of *student*, the time needed for the *bus* to enter that proximity is greater than the time the *student* decided to wait for the *bus*, the *tram* is in the proximity of radius r of the *student* and it has free resources ($free\_resource(tram) > 0$) then the *student* enters the *tram*. If the *tram* does not have free resources and the *cab* is in the proximity of radius r of the *student* and it has free resources ($free\_resources(cab) > 0$), then the *student* enter the *cab*. If the *tram* is not in the proximity of the *student*, the time needed for the *tram* to enter that proximity is greater that the time the *student* decided to wait for the *tram*, then the *student* searches for a *cab*; if the *cab* is in the proximity of the *student* and it has free resources $free\_resources(cab) > 0$, then the *student* enters the *cab*. If none of the above conditions holds, the time-stepping function ϕ_Δ is applied (simulating the passing of time), and then the above conditions are re-checked.

For this scenario the *bus* can be described as:

$$\begin{aligned} bus\_schedule &= in^{\Delta t_1} univ.out^{\Delta t_2} univ.in^{\Delta t_3} camp.out^{\Delta t_2} camp.bus\_schedule \\ bus &= bus^{\infty}_{(l\_bus, h\_bus, r\_bus)}[bus\_schedule]^{\mu} \qquad \text{where} \end{aligned}$$

- Δt_1 - time the *bus* needs to reach the *univ* starting from *camp*;
- Δt_2 - time the *bus* awaits at a stop;
- Δt_3 - time the *bus* needs to reach the *camp*;
- *l_bus*, *h_bus*, *r_bus* - the free, capacity and proximity radius of the *bus*.

Similarly, the *tram* can be described as follows:

$$\begin{aligned} tram\_schedule &= in^{\Delta t_4} univ.out^{\Delta t_5} univ.in^{\Delta t_6} camp.out^{\Delta t_5} camp.tram\_schedule \\ tram &= tram^{\infty}_{(l\_tram, h\_tram, r\_tram)}[tram\_schedule]^{\mu} \end{aligned}$$

The *cab* and the *client* could be described as in [9], but because our example contains only two locations the description could be simplified. The *cab* can be described as follows:

$$\begin{aligned} cab\_route &= out^{\Delta t_7} univ.in^{\Delta t_8} camp.cab\_route + out^{\Delta t_7} camp.in^{\Delta t_9} univ.cab\_route \\ cab &= cab^{\infty}_{(l\_cab, h\_cab, r\_cab)}[cab\_route]^{\mu} \end{aligned}$$

The description of *student* is more elaborated:

$$\begin{aligned} student &= student^{\infty}_{(1,1,r\_student)}[travel]^{\mu} \\ travel &= in^{\infty} univ.travel\_univ + in^{\infty} camp.travel\_camp \\ travel\_camp &= bus\_camp + tram\_camp + cab\_camp \\ travel\_univ &= bus\_univ + tram\_univ + cab\_univ \end{aligned}$$

$$
\begin{aligned}
cab\_camp &= in^{\Delta t_{15}} cab.(out^{\Delta t_9} cab.out^{\Delta t_{17}} univ.travel, cab\_camp)\\
cab\_univ &= in^{\Delta t_{14}} cab.(out^{\Delta t_8} cab.out^{\Delta t_{16}} camp.travel, cab\_univ)\\
tram\_camp &= in^{\Delta t_{13}} tram.(out^{\Delta t_4} tram.out^{\Delta t_{17}} univ.travel, tram\_camp)\\
tram\_univ &= in^{\Delta t_{12}} tram.(out^{\Delta t_6} tram.out^{\Delta t_{16}} camp.travel, tram\_univ)\\
bus\_camp &= in^{\Delta t_{11}} bus.(out^{\Delta t_1} bus.out^{\Delta t_{17}} univ.travel, bus\_camp)\\
bus\_univ &= in^{\Delta t_{10}} bus.(out^{\Delta t_3} bus.out^{\Delta t_{16}} camp.travel, bus\_univ)
\end{aligned}
$$

According to the above definitions, the *bus* and the *tram* move based on a predefined schedule relative to a global clock, no matter whether they contain a *student* inside or not, while the *cab* has no (time) restriction to move. Each *cab* has a cyclic movement between the two locations *univ* and *camp* even it has or does not have a *student* inside. The *student* can enter in the ambient found in his proximity, with no constraint imposed by other ambients. In the moment the *student* is at *univ* location, we can have one of the following scenarios:

- two of *bus*, *tram* and *cab*, or all are at the *univ* location and at least one of them has free resources; in this case the *student* enters one of them which has free resources, by choosing nondeterministic, with no regard to the cost of the trip;
- one of *bus*, *tram* and *cab* are at the *univ* location and has a free resource; in this case the *student* enters it with no regard to the cost of the trip, or the schedule of the others;
- one, two or all of *bus*, *tram* and *cab* are at the *univ* location, but no free resources are available; in this case the *student* awaits for an ambient with free resources, and moves to be in its proximity.

3.1 Coordination of Timed Mobile Ambients

The cost of the trip is not considered in all these scenarios. We can add priorities and interaction requirements acting over an initial assignment of the timers, capacities and proximities. Over the model tMA we define a coordination pair $(\mathcal{T}, \mathcal{CP})$. The first component $\mathcal{T}$ of this coordinating pair is a function assigning initial values to the timers, capacities and proximities. The second component $\mathcal{CR}$ is given by a set of rules. In our example the timer t_{10} should have initially a smaller value than the timer t_1, because the *student* is not willing to wait for a *bus* which has to cover a distance bigger or equal than the one from the two locations *univ* and *camp*. Having similar motivations, the timers t_{11}, t_{12} and t_{13} can be smaller that the initial value for t_3, t_4, respectively t_6. The rules given by CR influence the evolution of the system, and decide a certain behaviour according to the requirements expressed by the rules. For example, the *student* establishes a strategy for lowering its travel costs and gives a priority order: the *bus* has the bigger priority, followed by the *tram*, while the *cab* has the lower priority. We impose the following rules:

$$
\begin{gathered}
[l\_cab \neq h\_cab],\ [l\_cab = h\_cab],\ [t_{10} < t_1],\ [t_{11} < t_3],\\
[t_{10} < t_1 \wedge t_{12} < t_4],\ [t_{11} < t_3 \wedge t_{13} < t_6].
\end{gathered}
$$

These rules could be integrated in the syntax, and so the processes *cab* and *student* can be rewritten. Since the *cab* should make a trip only when it has

a *student* inside, and it should accept a *student* only after it completes the previous trip, we rewrite the syntax of the *cab* as

$$\begin{aligned} cab &= cab^{\infty}_{(l\_cab,h\_cab,r\_cab)}[\,[l\_cab = h\_cab]cab\_route] \\ cab\_route &= [l\_cab \neq h\_cab]out^{\Delta t_7} univ.in^{\Delta t_8} camp.[l\_cab = h\_cab]cab\_route \\ &\quad + [l\_cab \neq h\_cab]out^{\Delta t_7} camp.in^{\Delta t_9} univ.[l\_cab = h\_cab]cab\_route, \text{ where} \end{aligned}$$

- $l\_cab \neq h\_cab$ - denotes the fact that the *cab* has an *student* inside it;
- $l\_cab = h\_cab$ - denotes the fact that the *cab* is available.

The initial *student* has no strategy of lowering its travel cost by choosing an appropriate travel ambient. The coordinated *student* taking care of this aspect has the following syntax:

$$\begin{aligned} student &= student^{\infty}_{(1,1,r\_student)}[travel]^{\mu} \\ travel &= in^{\infty} univ.travel\_univ + in^{\infty} camp.travel\_camp \\ travel\_camp &= bus\_camp + tram\_camp + cab\_camp \\ travel\_univ &= bus\_univ + tram\_univ + cab\_univ \\ cab\_camp &= [t_{11} < t_3 \wedge t_{13} < t_6]in^{\Delta t_{15}} cab. \\ &\quad (out^{\Delta t_9} cab.out^{\Delta t_{17}} univ.travel, cab\_camp) \\ cab\_univ &= [t_{10} < t_1 \wedge t_{12} < t_4]in^{\Delta t_{14}} cab. \\ &\quad (out^{\Delta t_8} cab.out^{\Delta t_{16}} camp.travel, cab\_univ) \\ tram\_camp &= [t_{11} < t_3]in^{\Delta t_{13}} tram. \\ &\quad (out^{\Delta t_4} tram.out^{\Delta t_{17}} univ.travel, tram\_camp) \\ tram\_univ &= [t_{10} < t_1]in^{\Delta t_{12}} tram. \\ &\quad (out^{\Delta t_6} tram.out^{\Delta t_{16}} camp.travel, tram\_univ) \\ bus\_camp &= in^{\Delta t_{11}} bus.(out^{\Delta t_1} bus.out^{\Delta t_{17}} univ.travel, bus\_camp) \\ bus\_univ &= in^{\Delta t_{10}} bus.(out^{\Delta t_3} bus.out^{\Delta t_{16}} camp.travel, bus\_univ), \text{ where} \end{aligned}$$

- $t_{10} < t_1$ - compares the time (t_{10}) needed for the *bus* to reach *univ*, with the time (t_1) the *student* is willing to wait for *bus*;
- $t_{11} < t_3$ - compares the time (t_{11}) needed for the *bus* to reach *camp*, with the time (t_3) the *student* is willing to wait for *bus*;
- $t_{12} < t_4$ - compares the time (t_{12}) needed for the *tram* to reach *univ*, with the time (t_4) the *student* is willing to wait for *tram*;
- $t_{13} < t_5$ - compares the time (t_{13}) needed for the *tram* to reach *camp*, with the time (t_5) the *student* is willing to wait for *tram*.

3.2 Dynamic Aspects

The rules restrict the evolution of a system, acting when we have more than one interaction choice in a reduction step. In what follows we describe the possible evolutions of the system when the *student* is placed in the proximity of *univ*, after rewriting the *cab* and *student*. When *student* is at *univ* location, we have the following reduction:

$$\begin{aligned} &student^{\infty}_{(1,1,r\_student)}[travel]^{a} \mid univ^{\infty}_{(l\_univ,h\_univ,r\_univ)}[\,]^{\mu} \\ &\rightarrow univ^{\infty}_{(l\_univ-1,h\_univ,r\_univ)}[student^{\infty}_{(1,1,r\_student)}[travel\_univ]^{s}\,]^{\mu} \end{aligned}$$

If one the following scenarios holds:

- the *bus* having free resources and is in the proximity of *student*;
- the *bus* is not at *univ* location, $t_{10} > t_1$ (meaning that *student* waits for the *bus*) and the *bus* is arriving after t_1 units of time and has free resources;

the second reduction in the system is:

$$student^{\infty}_{(1,1,r\_student)}[travel\_univ]^a \mid bus^{\infty}_{(l\_bus,h\_bus,r\_bus)}[\,]^{\mu} \\ \rightarrow bus^{\infty}_{(l\_bus-1,h\_bus,r\_bus)}[student^{\infty}_{(1,1,r\_student)}[out^{\Delta t_3} bus.out^{\Delta t_{16}} camp.travel]^s\,]^{\mu}$$

But if the scenario is one of the following:

- in *univ* there is no *bus*; *tram*, having free resources, is inside *univ* in the proximity of *student*, and $t_{10} < t_1$ (which means that the *student* is not willing to wait for the *bus*);
- *tram* is not inside *univ*, $t_{12} > t_4$ (which means that the *student* is willing to wait for the *tram*), $t_{10} < t_1$ and the *tram* is arriving after t_4 units of time, and has free resources;

the next reduction in the system is going to be the following one:

$$student^{\infty}_{(1,1,r\_student)}[travel\_univ]^a \mid tram^{\infty}_{(l\_tram,h\_tram,r\_tram)}[\,]^{\mu} \\ \rightarrow tram^{\infty}_{(l\_tram-1,h\_tram,r\_tram)}[student^{\infty}_{(1,1,r\_student)}[out^{\Delta t_6} tram.out^{\Delta t_{16}} camp.travel]^s\,]^{\mu}$$

In case of the following situations:

- in *univ* there is no *bus* or *tram*; *cab*, having free resources, is inside *univ* is in the proximity of the *student*, $t_{10} < t_1$ (which means that the *student* is not willing to wait for the *bus*) and $t_{12} < t_4$ (which means that the *student* is not willing to wait for the *tram*);
- the *cab* is not inside *univ*, $t_{14} > t_9$ (which means that the *student* is willing to wait for the *cab*), $t_{10} < t_1$, $t_{12} < t_4$ and an available *cab* is arriving after t_9 units of time;

the system reduces as follows:

$$student^{\infty}_{(1,1,r\_student)}[travel\_univ]^a \mid cab^{\infty}_{(l\_cab,h\_cab,r\_cab)}[\,]^{\mu} \\ \rightarrow cab^{\infty}_{(l\_cab-1,h\_cab,r\_cab)}[student^{\infty}_{(1,1,r\_student)}[out^{\Delta t_8} cab.out^{\Delta t_{16}} camp.travel]^s\,]^{\mu}$$

After the *student* enters one of the *bus*, *tram* or *cab* by applying one of the three steps above, *student* is carried to the *camp* where *student* is discarded. Then *student* exits the *camp*, and we can have a similar scenario in order to move *student* from *camp* to *univ*.

4 Conclusion and Related Work

The new model is given by mobile ambients extended with timers associated to ambients and capabilities. We define the semantics of this model, and give some technical results. Using an easy-to-understand example, we describe the coordination steps given by indicating specific values to timers and capacities,

as well as to capacities and proximity radiuses, followed by defining a set of coordination rules.

The interaction and synchronization in time-driven coordination models are governed by time. We are working with relative time. Time is used both to restricting the interaction between components, and to enforce a limited availability for resources .

In [4] we extend the distributed π-calculus (which is able to model communications restricted by types) by introducing timers over channel names in order to define timeouts for communications. The resulting formalism is called timed distributed π-calculus ($(tD\pi)$. Over this formalism we define a coordination by timers for channel-based communications by assigning specific values to timers and defining a constant set of coordination rules [5]. A natural example motivating an extension from timed distributed π-calculus to timed mobile ambients is presented in [3].

References

1. Busi, N., Ciancarini, P., Gorrieri, R., Zavattaro, G.: Coordination Models: A Guided Tour. In: Coordination of Internet Agents: Models, Technologies, and Applications, pp. 6–24. Springer, Heidelberg (2001)
2. Cardelli, L., Gordon, A.: Mobile Ambients. In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
3. Ciobanu, G.: Interaction in time and space. In: Proceedings of Foundations of Interactive Computation. Electronic Notes in Theoretical Computer Science, pp. 45–61, 2007 (to appear)
4. Ciobanu, G., Prisacariu, C.: Timers for Distributed Systems. In: di Pierro, A., Wiklicky, H. (eds.) Quantitative Aspects of Programming Languages. Electronic Notes in Theoretical Computer Science, vol. 164(3), pp. 81–99 (2006)
5. Ciobanu, G., Prisacariu, C.: Coordination by Timers for Channel-Based Anonymous Communications. In: Foundations of Coordination Languages and Software Architectures (FOCLASA), pp. 1–19, 2006 (to appear in ENTCS)
6. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press, Cambridge (1999)
7. Papadopoulos, G.A.: Models and Technologies for the Coordination of Internet Agents: A Survey. In: Coordination of Internet Agents: Models, Technologies, and Applications, pp. 25–56. Springer, Heidelberg (2001)
8. Papadopoulos, G.A., Arbab, F.: Coordination Models and Languages. Advances in Computers 46, 329–400 (1998)
9. Teller, D., Zimmer, P., Hirschkoff, D.: Using Ambients to Control Resources. In: Brim, L., Jančar, P., Křetínský, M., Kucera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 288–303. Springer, Heidelberg (2002)

Pushing Random Walk Beyond Golden Ratio

Ehsan Amiri and Evgeny Skvortsov

School of Computing Science, Simon Fraser University, Burnaby, Canada
{eamiri,evgenys}@cs.sfu.ca

Abstract. We propose a simple modification of a well-known Random Walk algorithm for solving the Satisfiability problem and analyze its performance on random CNFs with a planted solution. We rigorously prove that the new algorithm solves the Full CNF with high probability, and for random CNFs with a planted solution of high density finds an assignment that differs from the planted in only ε-fraction of variables. In the experiments the algorithm solves random CNFs with a planted solution of any density.

1 Introduction

Random Walk(RW) is an algorithm for solving the Satisfiability problem originally proposed by Papadimitriou[7]. In this algorithm we start with a random initial assignment. At every step a random unsatisfied clause in the formula is selected and a value of a variable randomly picked from the clause is changed. It was proven by Papadimitriou that this algorithm finds a satisfying assignment of instances of 2-SAT in expected polynomial time.

The algorithm was also analyzed theoretically and tested for formulas with three variables per clause by Alekhnovich and Ben-Sasson[2]. It was proven that the algorithm solves in linear time random 3-Satisfiability formulas for densities lower than 1.6 (i.e. the clause/variable ratio is 1.6) and there is experimental evidence that the algorithm succeeds for densities up to 2.7. On the other hand it was proven that RW does not solve instances with planted solution of high densities. Moreover almost surely it does not find an assignment that would coincide with the planted solution on substantially more than the golden ratio conjugate ($\frac{2}{1+\sqrt{5}} \approx 0.62$) of all variables in linear time.

We add some kind of weighting of variables during Random Walk which makes it more powerful. Informally, the longer a variable keeps its value unchanged the greater its weight becomes, and it is harder to change a variable of higher weight. Our experiments show that Weighted Random Walk (WRW) finds solutions of random planted instances of 3-SAT of any fixed density with high probability. This is in contrast with Alekhnovich and Ben-Sasson's exponential lower bound for the running time of the standard random walk algorithm for solving random planted 3-SAT of density larger than a constant[2]. We rigorously prove that it solves the Full CNF (CNF consisting of all clauses that are satisfied by the planted solution) and for the random CNFs with planted solution of densities $\rho = \rho(n) \underset{n \longrightarrow \infty}{\longrightarrow} \infty$ for any $\varepsilon > 0$ almost surely it finds an assignment that differs

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 44–55, 2007.

from the planted solution on at most ε-fraction of all variables. The capability of the algorithm to solve random CNFs of constant density remains unproven.

Probabilistic analysis of algorithms for Satisfiability is a growing area. In particular some work on analysis of Local Search[8,5,3], DPLL[1] and spectral techniques[4] was done so far. There are known algorithms (see for example [4] and [8]) that are proven to be efficient in solving CNFs with a planted assignment, but the algorithms are more complicated than WRW, and only their capability of solving problems of high density is reported.

The remainder of the paper is organized as follows. In section 2 we give definitions and formulate known lemmata required for analysis, in section 3 theoretical upper bounds are given, in section 4 we present experimental results and we conclude in section 5.

2 Definitions and Main Concepts

2.1 Satisfiability and Weighted Random Walk

In a 3-SAT problem we are given a Boolean formula ϕ in 3-CNF, i.e. the formula is a conjunction of clauses. Each clause is a disjunction of three literals. Each literal is either some variable or a negation of some variable.

The goal is to find an assignment that satisfies all clauses. In a MAX-3-SAT problem the goal is to find an assignment that satisfies the greatest number of clauses. We don't make a distinction between assignments and boolean vectors of size equal to the number of variables and denote both by small Latin letters written in bold font, e.g. $\boldsymbol{x}$.

The models of a random CNF we use in this paper are a random CNF of fixed density and a random CNF with a planted solution of fixed density. Given density ρ and a boolean vector $\boldsymbol{r}$ of size n, to creat a random CNF with planted solution $\boldsymbol{r}$ we select (allowing repetition) uniformly at random ρn clauses among all $7\binom{n}{3}$ clauses that satisfy $\boldsymbol{r}$. The formula containing all $7\binom{n}{3}$ possible clauses is called the Full CNF with a planted solution. A random CNF of fixed density is obtained if we are not restricting ourselves to only clauses that satisfy $\boldsymbol{r}$.

In a random walk one starts with a random initial assignment and at every step a random unsatisfied clause in ϕ is selected and a random variable in the clause is flipped. The *Weighted Random Walk* starts with a random assignment and weights of all variables equal to 1. At the first stage of a step just as in regular RW a random unsatisfied clause in ϕ is selected, and for every variable in the clause we check the following. If its weight is 1, then it is flipped and its weight remains 1. If its weight is greater than 1, then the weight is decreased by 1. At the second stage two variables are randomly selected and their weights are increased by 1, if it does not make their weights greater than K, which is the maximum allowed. A pseudocode description of WRW is presented in Fig. 1.

```
INPUT: A CNF φ containing n variables, numbers T, K
OUTPUT: An assignment x
let x be a random vector.
let w(i) = 1, for i ∈ {1, ..., n}
for step from 1 to T do
    pick a random unsatisfied clause C in φ
    for all variables x_j in C do
        w(j) = w(j) - 1
        if w(j) = 0:
            let x_j = ¬x_j, w(j) = 1
    pick two random variables, and for each of them do
        if its weight is less than K then increase it by one
return x
```

Fig. 1. The Weighted Random Walk Algorithm

2.2 Probabilistic Analysis Tools

In this section we formulate some results from the area of probabilistic analysis that we will use to prove our theorems. We give them a form that is convenient for our purposes.

Chernoff Bounds. Let $B(p,n)$ be a random variable, that is a number of successes in n independent trials. If p is the probability of success in each trial, then $\mathbf{P}\left(\left|\frac{B(p,n)}{n} - p\right| \leq \varepsilon\right) \leq 2e^{-\frac{\varepsilon^2 n}{3p}}$.

Azuma's Inequality [6]. Let $X_0, X_1, \ldots$ be a sequence of random variables and c, Δ constants such that for each k, $|X_k - X_{k-1}| \leq c, \mathbf{E}(X_k - X_{k-1}) \geq \Delta$, then for all t and any λ we have

$$\mathbf{P}(X_t - X_0 \leq \lambda) \leq 2e^{-\frac{(t\Delta-\lambda)^2}{2tc^2}}.$$

Wormald's Theorem. In our analysis we use the theorem proven by Wormald [9] that allows one to replace probabilistic analysis of combinatorial algorithm with analysis of a deterministic system of differential equations.

We consider only discrete time random processes. Such a process is a probability space Ω denoted by $(Q_0, Q_1, \ldots)$, where each Q_i takes values in some set S. Consider a sequence Ω_n, $n = 1, 2, \ldots$, of random processes. The elements of Ω_n are sequences $(q_0(n), q_1(n), \ldots)$ where each $q_i(n) \in S$. For convenience the dependence of n will usually be dropped from the notation. Asymptotics, denoted by the notation o and O, are for $n \to \infty$, but uniform over all other variables. For a random X, we say $X = o(f(n))$ *always* if $\max\{x|\mathbf{P}(X = x) \neq 0\} = o(f(n))$. An event occurs *almost surely* (a.s.) if its probability in Ω_n is $1 - o(1)$. We denote by S^+ the set of all $h_t = (q_0, \ldots, q_t)$, each $q_t \in S$ for $t = 0, 1 \ldots$. By H_t we denote the *history* of the processes, that is the $n \times (t+1)$-matrix with entries $Q_i(j)$, $0 \leq i \leq t, 1 \leq j \leq n$.

A function $f(u_1, \ldots, u_j)$ satisfies a *Lipschitz condition* on $D \subseteq \mathbb{R}^j$ if a constant $L > 0$ exists with the property that

$$|f(u_1, \ldots, u_j) - f(v_1, \ldots, v_j)| \leq L \sum_{i=1}^{j} |u_j - v_i|$$

for all $(u_1, \ldots, u_j)$ and $(v_1, \ldots, v_j)$ in D.

Theorem 1 (Wormald, [9]). *Let k be fixed. For $1 \leq \ell \leq k$, let $y^{(\ell)} \colon S^+ \to \mathbb{R}$ and $f_\ell \colon \mathbb{R}^{k+1} \to \mathbb{R}$, such that for some constant C and all ℓ, $|y^{(\ell)}| < Cn$ for all $h_t \in S^+$ for all n. Suppose also that for some function $m = m(n)$:*

(i) for all ℓ and uniformly over all $t < m$, $\mathbf{P}\left(|Y_{t+1}^{(\ell)} - Y_t^{(\ell)}| > n^{1/5} \mid H_t\right) = o(n^{-3})$ always;

(ii) for all ℓ and uniformly over all $t < m$, $\mathbf{E}\left(Y_{t+1}^{(\ell)} - Y_t^{(\ell)} \mid H_t\right) = f_\ell(t/n, Y_t^{(1)}/n, \ldots, y_t^{(k)}/n) + o(1)$ always;

(iii) for each ℓ the function f_ℓ is continuous and satisfies a Lipschitz condition on D, where D is some bounded connected open set containing the intersection of $\{(t, z^{(1)}, \ldots, z^{(k)}) \mid t \geq 0\}$ with some neighborhood of $\{(0, z^{(1)}, \ldots, z^{(k)}) \mid \mathbf{P}\left(Y_0^{(\ell)} = z^{(\ell)} n, 1 \leq \ell \leq k\right) \neq 0$ for some $n\}$.

Then:

(a) For $(0, \hat{z}^{(1)}, \ldots, \hat{z}^{(k)}) \in D$ the system of differential equations $\frac{dz_\ell}{ds} = f_\ell(s, z_1, \ldots, z_k)$, $\ell = 1, \ldots, k$, has a unique solution in D for $z_\ell \colon \mathbb{R} \to \mathbb{R}$ passing through $z_\ell(0) = \hat{z}^{(\ell)}$, $1 \leq \ell \leq k$, and which extends to points arbitrarily close to the boundary of D.

(b) Almost surely $Y_t^{(\ell)} = nz_\ell(t/n) + o(n)$ uniformly for $0 \leq t \leq \min\{\sigma n, m\}$ and for each ℓ, where $z_\ell(s)$ is the solution in (a) with $\hat{z}^{(\ell)} = Y_0^{(\ell)}/n$, and $\sigma = \sigma(n)$ is the supremum of those s to which the solution can be extended.

3 Theoretical Lower Bounds for Success Probability

We first analyze the behavior of the algorithm given a formula ϕ that has all clauses that are satisfied by $\mathbf{1} = (1, \ldots, 1)$. Then we show that if the density is high enough then we get the same result.

We denote by $\mathcal{C}_{AAA}, \mathcal{C}_{AAB}, \mathcal{C}_{ABB}, \mathcal{C}_{BBB}$ sets of clauses containing three, two, one and none variables from A respectively.

3.1 Formula with All Clauses

In this section we analyze the performance of WRW on a 3-SAT formula ϕ that contains all clauses that satisfy $\mathbf{1}$, i.e. all clauses that have at least one positive literal.

Let $\boldsymbol{x}$ be any vector of variable values, $w(\cdot)$ be a weight function, and $\boldsymbol{x}', w'(\cdot)$ be a random vector and weight function which are obtained from $\boldsymbol{x}, w(\cdot)$ by

performing one step of the algorithm. Let A_i be a set of all variables that have value 1 and weight i, B_i a set of all variables that have value 0 and weight i, and let $a_i = \frac{|A_i|}{n}, b_i = \frac{|B_i|}{n}, a = \sum_{i=1}^{K} a_K, b = 1 - a$.

We consider a function $V(\boldsymbol{x}) = \sum_{i=1}^{K} i a_i - \sum_{i=1}^{K} (i-1) b_i$, which is obviously bounded by K. We will show that there exists a positive constant δ such that for any $\boldsymbol{x}$ we have

$$\mathbf{E}\left(V(\boldsymbol{x}') - V(\boldsymbol{x})\right) > \delta, \tag{1}$$

which by the Azuma's inequality implies that a.s. after $O(n)$ steps $V(\boldsymbol{x})$ becomes equal to K, and consequently the process stops.

Lemma 1. *Let us have an assignment $\boldsymbol{x}$ and X be some variable from A, Y be some variable from B. If C is an unsatisfied clause, chosen uniformly at random, then the probability that X occurs in C is $\frac{3(1-a^2)}{(1-a^3)n}$ and the probability that Y occurs in C is $\frac{3}{(1-a^3)n}$.*

Proof. Let C be a clause chosen uniformly at random (not necessary unsatisfied). Then

- $\mathbf{P}\left(C \in \mathcal{C}_{AAA}\right) = a^3, \mathbf{P}\left(X \in C | C \in \mathcal{C}_{AAA}\right) = \frac{3}{an}$,
  $\mathbf{P}\left(Y \in C | C \in \mathcal{C}_{AAA}\right) = 0$,
- $\mathbf{P}\left(C \in \mathcal{C}_{ABB}\right) = 3a^2 b, \mathbf{P}\left(X \in C | C \in \mathcal{C}_{AAB}\right) = \frac{2}{an}$,
  $\mathbf{P}\left(Y \in C | C \in \mathcal{C}_{AAB}\right) = \frac{1}{bn}$,
- $\mathbf{P}\left(C \in \mathcal{C}_{AAB}\right) = 3ab^2, \mathbf{P}\left(X \in C | C \in \mathcal{C}_{ABB}\right) = \frac{1}{an}$,
  $\mathbf{P}\left(Y \in C | C \in \mathcal{C}_{ABB}\right) = \frac{2}{bn}$,
- $\mathbf{P}\left(C \in \mathcal{C}_{BBB}\right) = b^3, \mathbf{P}\left(X \in C | C \in \mathcal{C}_{AAA}\right) = 0$,
  $\mathbf{P}\left(Y \in C | C \in \mathcal{C}_{BBB}\right) = \frac{3}{bn}$.

In the first case the clause will definitely be satisfied and in each of the latter three the probability that the clause is unsatisfied equals $\frac{1}{7}$. So we have $\mathbf{P}\left(\neg C(\boldsymbol{x})\right) = \frac{1}{7}(3a^2 b + 3ab^2 + b^3) = \frac{1}{7}(1 - a^3)$.

Now we compute

$$\begin{aligned}\mathbf{P}\left(X \in C \;\&\; \neg C(\boldsymbol{x})\right) = {} & \mathbf{P}\left(X \in C \;\&\; \neg C(\boldsymbol{x}) | C \in \mathcal{C}_{AAB}\right) \mathbf{P}\left(C \in \mathcal{C}_{AAB}\right) + \\ & \mathbf{P}\left(X \in C \;\&\; \neg C(\boldsymbol{x}) | C \in \mathcal{C}_{ABB}\right) \mathbf{P}\left(C \in \mathcal{C}_{ABB}\right) + \\ & \mathbf{P}\left(X \in C \;\&\; \neg C(\boldsymbol{x}) | C \in \mathcal{C}_{BBB}\right) \mathbf{P}\left(C \in \mathcal{C}_{BBB}\right). \end{aligned} \tag{2}$$

Events $X \in C$ and $\neg C(\boldsymbol{x})$ are independent under the conditions, so we have

$$\mathbf{P}\left(X \in C \;\&\; \neg C(\boldsymbol{x})\right) = \frac{1}{7}\Big(\frac{2}{an} 3a^2 b + \frac{1}{an} 3ab^2 + 0b^3\Big) = \tag{3}$$

$$= \frac{3}{7n}(2ab + b^2) = \frac{3}{7n}(1 - a^2) \tag{4}$$

We plug the obtained expression into the definition of conditional probability and get the desired expression $\mathbf{P}\left(X \in C | \neg C(\boldsymbol{x})\right) = \frac{3(1-a^2)}{(1-a^3)n}$.

The probability $\mathbf{P}\left(Y \in C | \neg C(\boldsymbol{x})\right)$ is computed similarly. □

Now let a_i, b_i correspond to $\boldsymbol{x}$ and a'_i, b'_i to $\boldsymbol{x}'$. We are interested in $\mathbf{E}\left(V(\boldsymbol{x}') - V(\boldsymbol{x})\right)$. We can express the change in V as

$$V(\boldsymbol{x}') - V(\boldsymbol{x}) = \sum_{i=1}^{K} i(a'_i - a_i) - \sum_{i=1}^{K}(i-1)(b'_i - b_i) \tag{5}$$

so to compute $\mathbf{E}\left(V(\boldsymbol{x}') - V(\boldsymbol{x})\right)$ we need to compute $\mathbf{E}\left(a'_i - a_i\right)$ and $\mathbf{E}\left(b'_i - b_i\right)$.

The numbers a_i and b_i are changed similarly. The set A_i changes because some variables leave it and some arrive into it. Now it is convenient to denote $c_i = a_i$ and $c_{-(i-1)} = b_i$. Let $q_{c_i \to c_{i\pm1}}$ be the number of variables that leave A_i and arrive into $A_{i\pm1}$. None of the variables can change its weight by more than one in one step, so we have

- $c'_i = c_i - q_{c_i \to c_{i-1}} - q_{c_i \to c_{i+1}} + q_{c_{i-1} \to c_i} + q_{c_{i+1} \to c_i}$, for all i, except $-K+1$ and K,
- $c'_K = c_K - q_{c_K \to c_{K-1}} + c_{b_{K-1} \to b_K}$, and similarly for c_{-K+1}.

Variables go from A_i to A_{i+1} when the weights of two variables are increased, so

$$\mathbf{E}\left(q_{a_i \to a_{i+1}}\right) = 2a_i. \tag{6}$$

Variables go from A_i to A_{i-1} and from A_1 to B_1 when three variables of an unsatisfied clause decrease weights/flip. Applying lemma 1 we get

$$\mathbf{E}\left(q_{a_i \to a_{i-1}}\right) = \frac{3(1-a^2)}{(1-a^3)n} a_i n = \frac{3(1-a^2)}{(1-a^3)} a_i, \mathbf{E}\left(q_{a_1 \to b_1}\right) = \frac{3(1-a^2)}{(1-a^3)} a_1. \tag{7}$$

Analogously we get

$$\mathbf{E}\left(q_{b_i \to b_{i-1}}\right) = \frac{3}{(1-a^3)} b_i, \mathbf{E}\left(q_{b_1 \to a_1}\right) = \frac{3}{(1-a^3)} b_1, \mathbf{E}\left(q_{b_i \to b_{i+1}}\right) = 2b_i. \tag{8}$$

Substituting the expressions for a'_i into (5) we obtain

$$V(\boldsymbol{x}') - V(\boldsymbol{x}) = \tag{9}$$
$$\underbrace{\sum_{i=0}^{K-1} q_{a_i \to a_{i+1}}}_{\Psi_1} \underbrace{- \sum_{i=1}^{K} q_{a_i \to a_{i-1}} - q_{a_1 \to b_1}}_{\Psi_2} \underbrace{- \sum_{i=0}^{K-1} q_{b_i \to b_{i+1}}}_{\Psi_3} \underbrace{+ \sum_{i=1}^{K} q_{b_i \to b_{i-1}} + q_{b_1 \to b_a}}_{\Psi_4}.$$

Using equations 6 - 8 we get $\mathbf{E}\left(\Psi_1\right) = 2a - 2a_K, \mathbf{E}\left(\Psi_2\right) = -\frac{3a(1-a^2)}{1-a^3}, \mathbf{E}\left(\Psi_3\right) = -2b + 2b_K, \mathbf{E}\left(\Psi_4\right) = \frac{3b}{1-a^3}$, thus

$$\mathbf{E}\left(V(\boldsymbol{x}') - V(\boldsymbol{x})\right) = 2a - 2a_K - \frac{3a(1-a^2)}{1-a^3} - 2b + 2b_K + \frac{3b}{1-a^3} = \tag{10}$$
$$\frac{4a^3 - a^2 - a + 1}{1 + a + a^2} - 2a_K + 2b_K. \tag{11}$$

Using a standard method for finding a local minima by analysis of the first derivative it can easily be shown that $\frac{4a^3-a^2-a+1}{1+a+a^2} > 0.42$, so if we could bound $2a_K$ by $c < 0.42$ then we would be able to conclude that $\mathbf{E}\,(V(\boldsymbol{x}') - V(\boldsymbol{x})) \geq 0.42 - c > 0$.

Next we argue that $a_K \leq \frac{1}{K}$. Thus, taking $K \geq 5$ we obtain inequality (1) with $\delta = 0.02$.

Lemma 2. *For any natural number K and for any $T = O(n)$ a.s. at any step of the WRW before step T we have $a_K \leq \frac{1}{K}$.*

Proof. We will use Wormald's theorem to prove that the system behaves close to solutions of a system of differential equations and then argue that the variable corresponding to a_K never becomes greater than $\frac{1}{K}$. Below we check conditions (i)-(iii).

(i) As at every step only one variable is flipped we have inequalities $max|a_l' - a_l| \leq 1, max|b_l' - b_l| \leq 1$ true with probability one.
(ii) This point follows from equations 6 - 8, when we set $f_{a_i}(a_0, \ldots, b_K) = \mathbf{E}\left(q_{a_{i+1}\to a_i}\right) + \mathbf{E}\left(q_{a_{i-1}\to a_{i+1}}\right) - \mathbf{E}\left(q_{a_i\to a_{i+1}}\right) - \mathbf{E}\left(q_{a_i\to a_{i-1}}\right)$ and use obtained expressions for all $\mathbf{E}\,(q_\diamond)$ for $i > 0$, and similarly for f_{b_i}, f_{a_1}.
(iii) The functions $f_\diamond$ are Lipschitz, because they have finite first derivative.

Thus we get the equations

$$\begin{cases} \frac{du_l}{dx} = f_{a_l}(u_1, \ldots, u_K) \\ \frac{dv_l}{dx} = f_{b_l}(u_1, \ldots, u_K) \end{cases}, \tag{12}$$

and initial conditions $u_0(0) = v_0(0) = \frac{1}{2}$, for $0 < i \leq K$, $u_i(0) = v_i(0) = 0$. Almost surely $a_l(t) = u_l(t/n) + o(1)$, $b_l(t) = v_l(t/n) + o(1)$.

Thus to finish the proof of the lemma we show that the solution of system (12) satisfies $u_K(x) < \frac{1}{K}$.

We use induction to show the following:

Claim. For any $0 \leq R < K$, if s is such that $\sum_{l=R+1}^{K} u_l(s)$ is maximum and equals $\alpha(K-R)$ then $u_R(s) > \alpha$, which in particular means that the maximum value of $\sum_{l=R}^{K} u_l(s)$ is greater than $\alpha(K-R+1)$.

Proof. We denote $\sum_{l=R+1}^{K} u_l(s)$ by $\mathcal{I}_{R+1}(s)$. First note that $\frac{\mathrm{d}\mathcal{I}_{R+1}(s)}{\mathrm{d}s} = q_{u_R\to u_R+1} - q_{u_{R+1}\to u_R}$, which follows directly from the definition of f, so if $\frac{\mathrm{d}\mathcal{I}_R(s)}{\mathrm{d}s} = 0$ then $u_R \geq u_{R+1}$. Indeed we have

$$q_{u_R\to u_R+1} - q_{u_R+1\to u_R} = 2u_R - \frac{3(1-u^2)}{(1-u^3)}u_{R+1}$$

and the expression $\frac{3(1-u^2)}{(1-u^3)}$ equals 2 if $u = 1$ and is smaller if $u < 1$. Thus $u_R < u_{R+1}$ would imply $\frac{\mathrm{d}\sum_{l=R+1}^{K} u_l(s)}{\mathrm{d}s} < 0$.

The induction proof will go from $R = K - 1$ to $R = 0$. The base of induction $R = K - 1$ follows from the fact that $\frac{du_K}{ds} = 0$ implies $u_{K-1} \geq u_K$.

Induction step:

Consider s_0 such that $\mathcal{I}_{R+1}(s_0)$ is maximum and equals $\alpha(K - R)$. We have $u_R(s_0) \geq u_{R+1}(s_0)$. For the sake of contradiction assume that $u_R(s_0) < \alpha_1$, which implies $u_{R+1}(s_0) < \alpha_1$. Then

$$\mathcal{I}_{R+2}(s_0) = \mathcal{I}_{R+1}(s_0) - u_{R+1}(s_0) > (K - R)\alpha_1 - \alpha_1,$$

which leads to a contradiction as $(K - R - 1)\alpha_1$ is the maximum value of $\mathcal{I}_{R+2}$. □

It follows from the Claim that if the maximum value of $\mathcal{I}_R(s)$ is $\alpha(K - R)$ then the maximum value of $\mathcal{I}_{R-1}(s)$ is at least $\alpha(K - R + 1)$. Thus if the maximum value of $u_K(s)$ is α then the maximum value of $\mathcal{I}_0(s)$ is at least $K\alpha$. As $\mathcal{I}_0(s)$ cannot be greater than 1 we get the inequality $u_K(s) < \frac{1}{K}$. Thus almost surely we have $a_K(t) < \frac{1}{K}$. □

So if $K \geq 5$ then there is a constant $c > 0$ such that at every step of the WRW the expectation of the amount of change of $V(\boldsymbol{x})$ is greater than c. The value of $V(\boldsymbol{x})$ cannot change by more than 5 at every step so by the Azuma's inequality we have

$$\mathbf{P}\left(V(\boldsymbol{x}^t) \leq nK\right) \leq 2e^{-\frac{(tc-nK)^2}{2 \cdot 25t}}.$$

The right hand side of the above equation starts to decrease rapidly when tc becomes greater than nK. This proves the following theorem.

Theorem 2. *If ϕ is a Full 3-CNF formula with a planted assignment $\boldsymbol{r}$ then almost surely WRW with $K \geq 5$ weights finds $\boldsymbol{r}$ in $O(nK)$ steps.*

3.2 Random Formula

In this subsection we prove

Theorem 3. *Let ϕ be a random 3-CNF with a planted solution $\boldsymbol{r}$ of density $\rho = \rho(n) \underset{n \to \infty}{\longrightarrow} \infty$, and $\varepsilon > 0$ be some constant. With high probability WRW with more than 5 weights finds a vector that differs from $\boldsymbol{r}$ in at most εn coordinates.*

Let ϕ be a random 3-CNF with a planted solution consisting of ones. For $A, B, A_1 \subseteq A$ we denote by $\mathcal{C}_{AA_1B}$ the set of all unsatisfied clauses that have one variable from A_1, one variable from $A \setminus A_1$ and one from B. By $\mathcal{C}^{\phi}_{AA_1B}$ we denote the set of clauses in ϕ that have this property. Analogously we define $\mathcal{C}_{AA_iB}, \mathcal{C}_{AAB_i}$, etc. We define the set of all unsatisfied clauses by $\mathcal{C}_u$ and all unsatisfied clauses in ϕ by $\mathcal{C}^{\phi}_u$.

For a formula with all clauses the expectation of the number of variables that at a given step go from A_1 to B_1 equals

$$\frac{|\mathcal{C}_{AA_1B}| + 2|\mathcal{C}_{A_1A_1B}|}{|\mathcal{C}^u|}, \tag{13}$$

while for formula ϕ it is

$$\frac{|\mathcal{C}^{\phi}_{AA_1B}| + 2|\mathcal{C}^{\phi}_{A_1A_1B}|}{|\mathcal{C}^{\phi}_{u}|}. \tag{14}$$

In the next lemma we show that a.s. $\frac{\mathcal{C}^{\phi}_{AA_1B}}{\rho n}$ is close to $6aa_1b$, which equals $\frac{\mathcal{C}_{AA_1B}}{\rho n}$. The same techniques can be used to show that other members of equation (14) divided by ρn are close to respective members of equation (13) divided by ρn. Under the conditions of theorem 3 we have $\frac{\mathcal{C}_u}{\rho n} = b^3 > \varepsilon^3$, thus the denominator of (13) is separated from zero, so expression (14) is close to (13).

Lemma 3. *Let $\rho(n)$ tend to infinity and b be a constant greater than 0. Then a.s. for any boolean assignment $\boldsymbol{x}$ and any subsets of variables $B, A, A_1 \subseteq A$, $\frac{|B|}{n} = b$, where $\boldsymbol{x}$ has all variables from A equal to 1 and all variables from B equal to 0, the following inequality holds: $\left|\frac{\mathcal{C}_{AA_1B}}{\rho n} - \frac{3}{7}a(a-a_1)b\right| \leq o(1)$.*

Proof. By simple counting it can be shown that if C is chosen uniformly at random then $\mathbf{P}\left(C \in \mathcal{C}_{AA_1B}\right) = \frac{3}{7}a_1(a-a_1)b$. Let $\mathcal{C}^{\phi}_{AA_1B}$ be the multiset of clauses in ϕ that belong to $\mathcal{C}_{AA_1B}$. In total ρn clauses are chosen to be in ϕ, so the expectation of the size of $\mathcal{C}^{\phi}_{AA_1B}$ equals $\frac{3}{7}a_1(a-a_1)b\rho n$. Using the Chernoff bound we get

$$\mathbf{P}\left(\left|\frac{|\mathcal{C}^{\phi}_{AA_1B}|}{\rho n} - \frac{3}{7}a_1(a-a_1)b\right| > \epsilon\right) < 2e^{-\frac{\epsilon^2\rho n}{\frac{3}{7}a_1(a-a_1)b}}.$$

We will say that ϕ is ϵ-*bad* if there exists an assignment, and subsets of variables $A, B, A_1 \subseteq A$ for which inequality $\left|\frac{|\mathcal{C}^{\phi}_{AA_1B}|}{\rho n} - \frac{3}{7}a_1(a-a_1)b\right| > \epsilon$ is true.

Now we put to use the fact that the probability of a union of events is less than or equal to the sum of the probabilities of the events to estimate the probability of ϕ being ϵ-bad.

There are 2^n boolean assignments, 2^n ways to select A and B, and at most 2^n to select $A_i \subseteq A$, so we have

$$\mathbf{P}\left(\phi \text{ is bad}\right) \leq 2e^{-\frac{\epsilon^2\rho n}{\frac{3}{7}a_1(a-a_1)b}}2^{3n} = e^{-n(\gamma_1\epsilon^2\rho-\gamma_2)}, \tag{15}$$

where γ_1, γ_2 are constants. We can choose $\epsilon = \rho^{-1/3} = o(1)$ so as $\rho \to \infty$ the function in the right hand side of the equation (15) is $o(1)$, which completes the proof. □

Thus for random CNFs with planted solution $\mathbf{1}$ and vectors with more than εn zeros, WRW acts as it does for the Full CNF, that is it tends to get closer and closer to $\mathbf{1}$. So with high probability an assignment with more than $(1-\varepsilon)n$ ones will be found.

3.3 Discussion

The obtained theoretical results provide intuition on the reasons of the algorithm's success. When standard Random Walk starts with a random assignment there are more occurrences of variables assigned zero in the unsatisfied clauses, so the number of zero variables decreases. But when the golden ratio conjugate is reached the numbers of occurrences of zeros and ones become equal and progress stops. For the same reasons, when WRW starts, the number of zero variables is decreased. Once there are fewer zero variables than one variables, the ones start benefiting from increasing weight of randomly picked variables. The problem one might expect here is that weights of some one variables grow infinitely (or up to a maximum allowed size), while other variables still stay zero. Lemma 2 shows that this is not the case with WRW: the set of one variables with maximum possible weight stays reasonably bounded, and thus the added weight is used in the 'struggle' between one and zero. Formally it is shown via use of a potential function V.

4 Experiments

In this section we describe experiments done with the WRW algorithm. Our experiments are done on random and random planted 3-SAT.

With regard to random 3-SAT, our experiments show that WRW works in linear time for formulas with density 3.9. We studied the running time of the algorithms on formulas with 10000 variables and then increased the number of variables in steps of 1000 until 50000. For each fixed density we ran the algorithm on 100 random instances. The result of this experiment shows linear running time of WRW for density 3.9 when $K = 4$. This is interesting when compared with the empirical evidence that standard random walk requires exponential time for densities higher than 2.7.

Other experiments that we report here are for random planted 3-SAT. In the first set of experiments we try to determine for which densities WRW can solve random planted 3-SAT in a reasonable timebound. It turns out that for any fixed density WRW works reasonably fast. Our experiment was done on formulas with 10000 variables. The density started from 3 and increased to 10 in jumps of 0.1. For each fixed density the algorithm ran on 100 random instances and we looked at the average running time of these 100 instances. The results of this experiment are summarized in fig. 2 in the next page. As it is seen, the hardest instances are those with density around 5. When the density is below 3, the formula has too many solutions and it is easy to find one. When the density is higher than 10, intuitively speaking, the formula contains a lot of information about the planted solution and this information guides the algorithm toward it.

The next set of experiments was aimed at figuring out the running time of WRW on random planted instances of a fixed density. Our observations in this part were surprising. For density 10, we ran the algorithm on instances with 10000 to 100000 variables. The running time was the highest for instances with 10000 variable and then it reduced and converged to a fixed value and remained

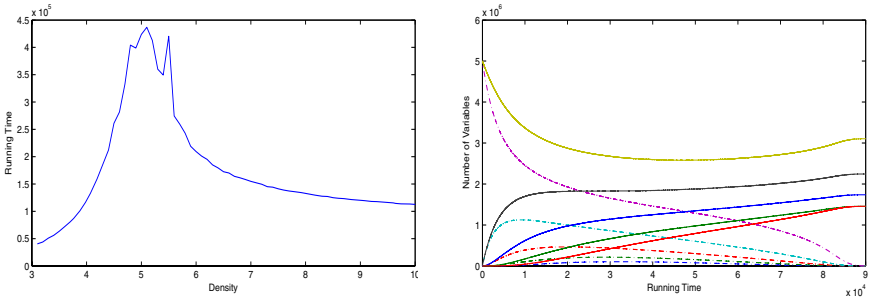


Fig. 2. Left: Running time vs. density. Right: Number of variables with different weights vs. time.

steady. We believe that this is because this number of variables is not large enough to allow us to see the asymptotic behavior of the algorithm. We observed a similar behavior when the density was set to 4.5.

The last set of experiments was done to check how the number of variables with each fixed weight changes during the course of the algorithm. For this experiment K is set to 5, so there are ten classes of variables. The experiment was done on formulas with 10000 variables and density 30. The result is summarized in the figure 2. Each curve shows c_l, i.e. the total number of variables with a specific weight in 1000 experiments. The solid lines correspond to $c_1, \ldots, c_K$, the dashed to $c_0, \ldots, c_{-K+1}$ starting with the upmost ones and going down. Since all experiments were finished before 90000 steps, when time approaches to 90000 in the graph, all lines become straight. Planted solution is chosen to be all one. We see that in fact for all $l(1 \leq l < K)$ we have $c_l > c_l + 1$, which is even stronger than the fact stated in the Claim used in the proof of Lemma 2. Another observation about the graphs one could predict using formulas (6), (7) and (8) is that when $a \longrightarrow 1$ we have the value b_i/b_{i+1} growing, while the value a_i/a_{i+1} decreases, which intuitively means that the weights are more and more evenly distributed over ones, while there are more zeros with small weights than with bigger weights.

5 Conclusion and Future Work

We proposed Weighted Random Walk as a modification of the Random Walk algorithm studied in [7,2]. The experiment shows that WRW finds a satisfying assignment for random formulas of density 3.9 and for instances of random planted 3-SAT of any density. We proved that for instances of random planted 3-SAT with density $\omega(1)$, WRW finds a solution that is equal to the planted solution except on a fraction ϵ of variables.

It remains open to show that WRW finds a satisfying assignment for instances of random planted 3-SAT with any density. Analysis of WRW for regular random formulas is also left open.

Acknowledgment. We would like to thank Andrei Bulatov, Valentine Kabanets, Javier Thaine and anonymous reviewers for helpful comments.

References

1. Achlioptas, D., Sorkin, G.B.: Optimal myopic algorithms for random 3-SAT. In: IEEE Symposium on Foundations of Computer Science, pp. 590–600 (2000)
2. Alekhnovich, M., Ben-Sasson, E.: Analysis of the random walk algorithm on random 3-CNFs Technical Report ECCC TR04-016 (2002)
3. Bulatov, A.A., Skvortsov, E.S.: Efficiency of local search. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 297–310. Springer, Heidelberg (2006)
4. Flaxman, A.: A spectral technique for random satisfiable 3-cnf formulas. In: SODA 2003, pp. 357–363 (2003)
5. Koutsoupias, E., Papadimitriou, C.: On the greedy algorithm for satisfiability. Information Processing Letters 43(1), 53–55 (1992)
6. Raghavan, P., Motwani, R.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
7. Papadimitriou, C.H.: On selecting a satisfying truth assignment. In: Proceedings of the 32nd Annual IEEE FOCS'91, pp. 163–169 (1991)
8. Feige, U., Vilenchik, D.: A local search algorithm for 3sat. Technical report, The Weizmann Institute of Science (2004)
9. Wormald, N.: Differential equations for random processes and random graphs. The Annals of Applied Probability 5(4), 1217–1235 (1995)

Reversible Machine Code and Its Abstract Processor Architecture

Holger Bock Axelsen, Robert Glück, and Tetsuo Yokoyama

DIKU, Dept. of Computer Science, University of Copenhagen
DK-2100 Copenhagen, Denmark
{funkstar,glueck,yokoyama}@diku.dk

Abstract. A reversible abstract machine architecture and its reversible machine code are presented and formalized. For machine code to be reversible, both the underlying control logic and each instruction must be reversible. A general class of machine instruction sets was proven to be reversible, building on our concept of reversible updates. The presentation is abstract and can serve as a guideline for a family of reversible processor designs. By example, we illustrate programming principles for the abstract machine architecture formalized in this paper.

1 Introduction

This paper presents the principles behind a reversible processor architecture. We are interested in the *von Neumann architecture*, a classic computer design for sequential computation with a single random access memory. One of the first reversible programmable processors built, *Pendulum*, is of this type [16,17,7]. We shall define an abstract machine and prove that all instruction sets for this machine that satisfy certain formal conditions, identified and presented in this paper, are reversible; *i.e.*, their semantics are forward and backward deterministic. A unique feature of the reversible abstract machine is a control logic that allows us to change the direction of execution by flipping the direction bit. It is noteworthy that any program written in reversible machine code is guaranteed to be reversible; no programming error can break the reversibility.

The purpose of this paper is to provide a clear specification of the interplay between the physical and software levels. Our goal is for this formalization to provide a better understanding of the essence of reversible processor architectures. From here, hardware designers may extend the model and work downward toward a physical realization (*e.g.*, at the circuit level) and software designers may work upward through the various abstraction layers (*e.g.*, assembler, high-level languages, compilers). Understanding this interface is important, as a challenge of reversible computing is that the *entire* computing system must be reversible, from the physical bottom to the abstract top.

We believe that reversible computation models have properties that are noteworthy in their own right and have interesting implications in other areas. For example, reversible computing holds the promise of reducing power

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 56–69, 2007.

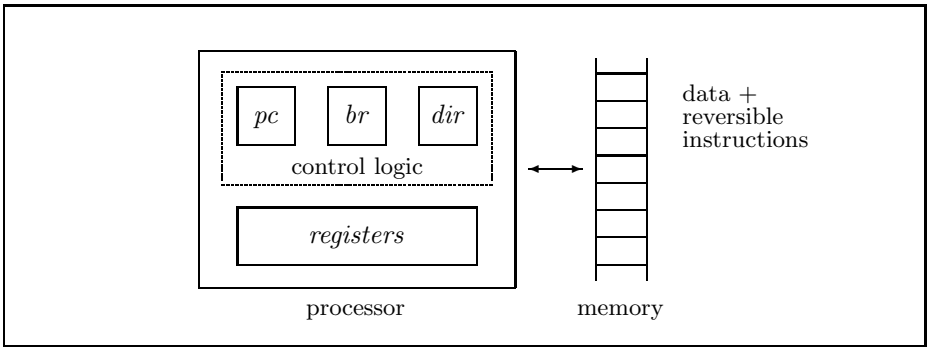


Fig. 1. Reversible abstract machine

consumption [13,2], and unconventional physical computation models, such as quantum computing, require that all computations are organized reversibly [6].

After presenting the architecture (Sect. 2), we describe the instruction set and state its formal properties (Sect. 3). We explain the principles of programming the processor (Sect. 4), and conclude with related work (Sect. 5) and future work (Sect. 6). An appendix (App. A) containing proof sketches is also included.

2 Reversible Abstract Machine

This section presents the principles behind an abstract reversible processor architecture. The design of the control logic is based on work by Vieri [16], Frank [7], Hall [12], and Cezzar [3]. Our contribution is a clear design and a formalization of a reversible abstract machine for which we will prove that any instruction set that satisfies certain conditions is reversible.

We are interested in a reversible version of the *von Neumann architecture*, a classic computer design with a processing unit (with registers) and random access memory, instead of more theoretical models, such as Turing machines. A standard abstract machine performs *one-directional* execution of machine code, while a reversible abstract machine allows *bidirectional* (forward and backward) execution of reversible machine code.

The challenge of reversibility for an abstract machine is two-fold. First, the *instruction execution* must be reversible. This places restrictions on the instructions possible in such a reversible architecture. Specifically, they must be injective when considered as functions on machine states. Second, the *control logic* must be reversible. With control logic, we refer to the part of the processing unit controlling the *program counter*, including any interaction from the instructions. To ensure reversibility, this must also be an injective function on machine states.

Control Logic. The most difficult of these challenges is good design for control. At any point of program execution, the computation direction of a reversible machine can be switched (forward, backward). In a standard machine model, we

constantly face the *orthogonality problem*. In general, we cannot know whether we have arrived at the current state by a jump or by sequential execution. Thus, we cannot reverse program execution and return to exactly the state that led to the current state. A classic solution to this problem is the generation of a *trace* [13,2,3]. For practical purposes, however, this is unsatisfactory, as the trace grows proportionally to the length of the computation.

A good design for a reversible processor has to take a *trace-free approach*. This can be accomplished in an architecture that has *three special-purpose registers* involved in the control logic (Fig. 1):

- A *program counter* (pc) for pointing to the current instruction.
- A *branch register* (br) for jumps.
- A *direction bit* (dir) for specifying execution direction.

Between instruction executions, we update the control state by the following two rules: (1) If the branch register is *zero*, add the direction bit to the program counter. The direction bit has the value 1 or -1, and thus the program counter is either incremented or decremented. This achieves sequential forward or backward execution of a program. (2) If the branch register is *non-zero*, add that (times the direction bit) to the program counter. A non-zero branch register indicates that a jump is to be performed. Instructions that require jumps will thus modify the branch register and *not* the program counter directly. The effect of this is to make the control logic reversible by solving the orthogonality problem: the branch register is preserved after modifying the program counter. This is sufficient to determine where a jump came from. We will come back to this important point after formalizing the state model and the execution semantics.

State Model. We model the set of machine states, $\Sigma = \mathcal{M} \times \mathcal{R} \times \mathcal{C}$, as follows. A machine state $\sigma \in \Sigma$ is a triple $\sigma = (M, R, C)$ such that

$$\begin{array}{ll} \text{Memory:} & \mathcal{M} \ni M : \mathbb{Z}_{32} \to \mathbb{Z}_{32} \\ \text{Registers:} & \mathcal{R} \ni R : \mathit{RegNames} \to \mathbb{Z}_{32} \\ \text{Control:} & \mathcal{C} \ni C : \mathbb{Z}_{32} \times \mathbb{Z}_{32} \times \{1, -1\} \end{array}$$

where $\mathbb{Z}_{32}$ is the set of 32-bit integers (or any set of n-bit integers) and *RegNames* is the set of register names reg_0 to reg_{31} (written `$0` to `$31` in machine code). In a state, M and R describe the contents of the *memory* and the *registers* (each is a function from addresses or register names to 32-bit integers), respectively, and tuple $C = (pc, br, dir)$ encapsulates the *control* registers. We do not model input/output facilities. It is assumed that program and data are entered into memory before the machine starts and that the results are read from memory after the machine stops (if it stops). Before loading program and data, all registers and the entire memory are zero-cleared.

Instructions. The design of the abstract machine is closely tied to the design of its reversible instruction set. The instructions fall into the two general classes of *data modification* and *control flow* instructions. As an example of a reversible data modification instruction, consider the addition instruction

```
ADD $3 $4 .
```

Reversible execution step:

$$\frac{\sigma \rightarrow_{ie} \sigma'' \quad \sigma'' \rightarrow_{pc} \sigma'}{\sigma \rightarrow_{1} \sigma'}$$

Instruction execution:

$$\frac{\sigma = (M, R, (pc, br, dir)) \quad \mathcal{I}(M(pc), dir) = i \quad \langle \sigma, i \rangle \rightarrow_{inst} \sigma'}{\sigma \rightarrow_{ie} \sigma'}$$

Control logic:

$$\frac{br = 0}{(M, R, (pc, br, dir)) \rightarrow_{pc} (M, R, (pc +_{32} dir, br, dir))}$$

$$\frac{br \neq 0}{(M, R, (pc, br, dir)) \rightarrow_{pc} (M, R, (pc +_{32} br \cdot dir, br, dir))}$$

Fig. 2. Semantics of a reversible execution step

The effect of executing this instruction is to add to the value in register `$3` the value in register `$4`. This instruction has an inverse interpretation: subtract the value of `$4` from `$3`. A reversible architecture must provide two interpretations of the same instruction depending on the direction bit: the *standard semantics* and the *inverse semantics*. A standard instruction usually has the effect of *overwriting* the destination register irreversibly: the original value of the register cannot be recovered from the resulting state. Such irreversible instructions are not allowed in a reversible architecture.

As outlined above, control flow instructions interact with the control state in non-trivial ways. The unconditional jump instruction

```
BRA 15
```

has the effect of adding $15 \cdot dir$ to the branch register br. It is important that branch instructions use relative *offsets* for jumps, and not absolute addresses. If $br \neq 0$, the control logic adds $br \cdot dir$ to the program counter pc, instead of irreversibly overwriting the value of pc with an absolute address to perform the jump. We illustrate programming principles for the abstract machine in Sect. 4.

3 Reversible Instruction Set

The small-step operational semantics shown in Fig. 2 describes the workings of the machine code after program and data have been entered into memory. We treat each instruction as atomic, regardless of the actual processor design. Each step updates the machine state $\sigma = (M, R, C)$. Below, we will explain the rules and state the main theorems. As an example of a reversible instruction set, we use the *Pendulum Instruction Set Architecture*, PISA [17,7]. The meanings of the instructions are defined by structural operational semantics, *cf.* [18].

Reversible Execution Step. An *execution step* is described by a relation $\rightarrow_1 \subseteq \Sigma \times \Sigma$. The single judgment form shows explicitly how an execution step consists of two parts: (i) *instruction execution* ($\rightarrow_{ie}$) and (ii) *control logic* ($\rightarrow_{pc}$). This separation is important when we consider the reversibility properties later, as the details are more subtle than one would initially assume.

Control Semantics. The relation $\rightarrow_{pc} \subseteq \Sigma \times \Sigma$ formalizes the control logic as described in Sect. 2. The program counter *pc* is updated, while all other parts of the state are unchanged. The choice between the two rules depends on the contents of branch register, *br*. Operation $a +_{32} b$ is defined as $(a + b) \bmod 2^{32}$.

Instruction Execution Semantics. The relation $\rightarrow_{ie} \subseteq \Sigma \times \Sigma$ is a single judgment form that defines instruction execution. We will need the direction bit *dir* as well as the current instruction $M(pc)$. The *instruction interpretation* function $\mathcal{I}$ has type $(\mathbb{Z}_{32} \times \{1, -1\}) \rightharpoonup \mathit{Inst}$, where *Inst* is a fitting (abstract) description of the instruction set. If the bit pattern does not define a legal instruction, its interpretation is undefined. An abstract instruction $i \in \mathit{Inst}$ will consist of an instruction name and up to three arguments, which can be either register names or immediate values (*i.e.*, integer constants contained in the instruction). Depending on *dir*, the current instruction $M(pc)$ is mapped into an abstract instruction i implementing its standard or its inverse semantics. This mapping is *local program inversion* performed on-the-fly at execution time [10]. It is important that the inversion of an instruction depends only on the local context of the instruction ('peephole') and, unlike other program inversion methods [11,14], does not require global analysis or transformation of the program. This is key to efficient reversible execution.

Local program inversion is the only feature of $\mathcal{I}$ in which we are interested—the actual bit patterns of the instructions do not concern us. Thus, we only assume that $\mathcal{I}(b, d) = i \Leftrightarrow \mathcal{I}(b, -d) = \mathit{inv}(i)$. The inverse instruction $\mathit{inv}(i)$ of an instruction i is specified by its effect on the state such that $\forall \sigma_{in}, \sigma_{out} \in \Sigma$:

$$\langle \sigma_{in}, i \rangle \rightarrow_{inst} \sigma_{out} \iff \langle \mathit{flip}(\sigma_{out}), \mathit{inv}(i) \rangle \rightarrow_{inst} \mathit{flip}(\sigma_{in})$$

where $\mathit{flip}((M, R, (pc, br, dir))) = (M, R, (pc, br, -dir))$ flips the direction bit. Suppose bit pattern b represents instruction `ADD $2 $3`, then $\mathcal{I}(b, 1) = \mathit{ADD}\ \mathit{reg}_2\ \mathit{reg}_3$ and $\mathcal{I}(b, -1) = \mathit{SUB}\ \mathit{reg}_2\ \mathit{reg}_3$. We use the self-inverse function *inv* defined by the following table. Unless indicated in this table, we have $\mathit{inv}(i) = i$. Instructions *RL*, *RLV*, *RR*, *RRV* rotate the bit pattern of a register reg_{sd} by a specified number of places to the left or right.

i				$\mathit{inv}(i)$			
ADD	reg_{sd}	reg_t		*SUB*	reg_{sd}	reg_t	
ADDI	reg_{sd}	*imm*		*ADDI*	reg_{sd}	$-\mathit{imm}$	
RL	reg_{sd}	*amt*		*RR*	reg_{sd}	*amt*	
RLV	reg_{sd}	reg_t		*RRV*	reg_{sd}	reg_t	

And-xor:

$$\frac{\sigma = (M, R, C) \qquad reg_d \neq reg_s \qquad reg_d \neq reg_t}{\langle \sigma, ANDX\ reg_d\ reg_s\ reg_t \rangle \rightarrow_{inst} (M, R[reg_d \mapsto R(reg_d) \oplus (R(reg_s) \wedge R(reg_t))], C)}$$

Branch-if-equal:

$$\frac{\sigma = (M, R, (pc, br, dir)) \qquad R(reg_a) = R(reg_b)}{\langle \sigma, BEQ\ reg_a\ reg_b\ off \rangle \rightarrow_{inst} (M, R, (pc, br +_{32} off \cdot dir, dir))}$$

$$\frac{\sigma = (M, R, C) \qquad R(reg_a) \neq R(reg_b)}{\langle \sigma, BEQ\ reg_a\ reg_b\ off \rangle \rightarrow_{inst} \sigma}$$

Unconditional-branch:

$$\frac{}{\langle (M, R, (pc, br, dir)), BRA\ off \rangle \rightarrow_{inst} (M, R, (pc, br +_{32} off \cdot dir, dir))}$$

Reverse-unconditional-branch:

$$\frac{}{\langle (M, R, (pc, br, dir)), RBRA\ off \rangle \rightarrow_{inst} (M, R, (pc, -(br +_{32} off \cdot dir), -dir))}$$

Swap-branch-register:

$$\frac{}{\langle (M, R, (pc, br, dir)), SWAPBR\ reg_b \rangle \rightarrow_{inst} (M, R[reg_b \mapsto br], (pc, R(reg_b), dir))}$$

Memory-register-exchange:

$$\frac{\sigma = (M, R, (pc, br, dir)) \qquad pc \neq R(reg_a)}{\langle \sigma, EXCH\ reg_d\ reg_a \rangle \rightarrow_{inst} (M[R(reg_a) \mapsto R(reg_d)], R[reg_d \mapsto M(R(reg_a))], C)}$$

Fig. 3. Excerpt of the instruction semantics of PISA ($\rightarrow_{inst}$)

Instruction Semantics. The relation $\rightarrow_{inst} \subseteq (\Sigma \times Inst) \times \Sigma$ is a judgment form for updating the state by executing an abstract instruction. Except for the program counter, all parts of a state may be reversibly updated by $\rightarrow_{inst}$. As will be explained below, the relation $\rightarrow_{inst}$ may be extended with instructions the execution semantics of which are reversible updates (with the restriction that pc and $M(pc)$ must not be changed). The concept of a reversible update is introduced below. An excerpt of the semantics for PISA is shown in Fig. 3. We shall now describe the semantics of the selected instructions.

Data Modification. And-xor ($ANDX$) updates the register reg_d with the result of $R(reg_s) \wedge R(reg_t)$. Memory M and control C are unchanged. The update $a \oplus b$ is defined as bitwise exclusive-or on the binary representation of values a and b. Similarly, $\wedge$ is bitwise and. Conditions $reg_d \neq reg_s$ and $reg_d \neq reg_t$ ensure that reg_d is not an operand of $\wedge$, which would be generally irreversible.

Branch Instructions. Branch-if-equal (BEQ) depends on the contents of registers reg_a and reg_b: if they are equal, $off \cdot dir$ is added to branch register br; otherwise, state σ is unchanged. Unconditional-branch (BRA) is similar to BEQ except that

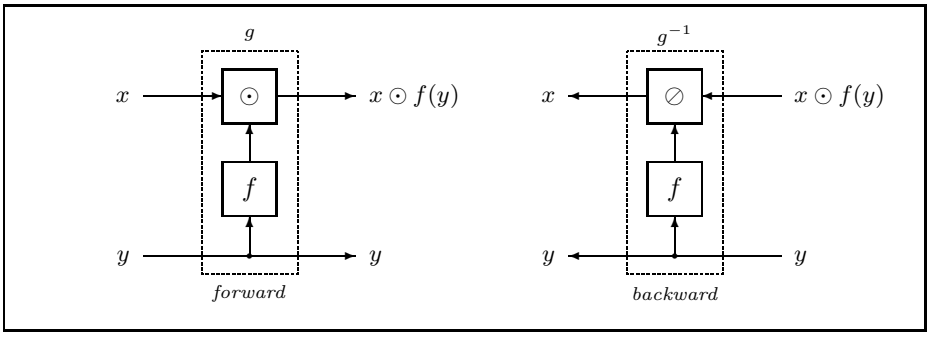


Fig. 4. Logically reversible update

br is unconditionally updated. Reverse-unconditional-branch ($RBRA$) flips the sign of the direction bit and the sign of the branch register after adding $\mathit{off} \cdot \mathit{dir}$ to it.[1] Swap-branch-register ($SWAPBR$) exchanges the contents of register reg_b and branch register br. The branch instructions modify the branch register br, but never the program counter pc, which is only updated by the control logic. This is the way in which instruction execution and control logic cooperate.

Memory Access. Memory access can be performed reversibly by allowing only an exchange ($EXCH$) between a register reg_d and a memory cell pointed to by reg_a (regardless of whether the memory cell holds an instruction or data). This combines the conventional irreversible load and store instructions that overwrite values into a single reversible instruction.

Reversible Updates. The conditions of an instruction *reversibly updating* values are as follows. We say that a partial function, written as an infix binary operator $\odot : (A \times B) \rightharpoonup C$, is *injective in its first argument*, if $\forall a, a' \in A, \forall b \in B$: if $a \odot b$ and $a' \odot b$ are defined then

$$a \odot b \;=\; a' \odot b \;\Rightarrow\; a = a' \;.$$

For any operator $\odot$ injective in its first argument there exists an operator $\oslash : (C \times B) \rightharpoonup A$ such that $\forall a \in A, \forall b \in B$:

$$((a \odot b) \oslash b) \;=\; a \;.$$

For example, $n + c = m + c$ implies $n = m$ and we have $(n + c) - c = n$ for any integer c. On the other hand, $n \cdot 0 = m \cdot 0$ does not imply $n = m$, and thus the operator $\cdot$ is not injective in its first argument (in fact, it is not injective in any argument). Bitwise exclusive-or, $\oplus$, has the useful property that $((a \oplus b) \oplus b) = a$.

[1] The operational meaning of $RBRA$ given in [7] is incorrect in that it either breaks the reversibility of the control logic, or does not work as intended in uncalls of subroutines. The semantics shown in Fig. 3 rectifies these problems.

Every injective operator is also injective in its first argument, but the converse need not hold. Note that $(a \odot b) \oslash b = a$ does not imply $(c \oslash b) \odot b = c$, so $\odot$ and $\oslash$ are not necessarily exchangeable.[2]

Given a partial function $f : D \rightharpoonup B$ and operator $\odot : (A \times B) \rightharpoonup C$ injective in its first argument, we call a partial function $g : (A \times D) \rightharpoonup (A \times D)$ such that

$$g(x, y) = (x \odot f(y), y) \ ,$$

a *reversible update wrt its first argument*. There always exists a left inverse

$$g^{-1}(x, y) = (x \oslash f(y), y) \ .$$

A reversible update g is *necessarily* injective. It is bijective if $A \times D$ is finite, and g is total. The second argument of g and g^{-1} is returned unchanged. Note that f does *not* need to be injective and that parameters x and y may be tuples $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$, $n, m \geq 0$. Clearly, the above can be stated for any argument, and not only for the first. Reversible updates are illustrated in Fig. 4. It is useful to know that a reversible update with $\oplus$ is self-inverse for any f:

$$g(x, y) = g^{-1}(x, y) = (x \oplus f(y), y)$$

Reversible updates are particularly well suited to model changes of a computation state where one part of the state, x, is updated using another part of the state, y, that is *not* changed. As an example, consider the instruction `ANDX` the effect of which is described by the reversible update

$$g_{\texttt{ANDX}}(x, (y_1, y_2)) = (x \oplus (y_1 \wedge y_2), (y_1, y_2)) \ .$$

A particular instruction, say `ANDX $2 $5 $7`, can thus be viewed as a reversible update of register `$2` by $g_{\texttt{ANDX}}(\texttt{\$2}, (\texttt{\$5}, \texttt{\$7}))$. It is self-inverse: $g_{\texttt{ANDX}} = g^{-1}_{\texttt{ANDX}}$.

It is essential for the reversibility of the entire machine architecture that *all* modifications of the machine state, be it by instructions or the control logic, are reversible updates regardless of whether they change memory M, registers R, or control C. For example, *BEQ* is a reversible update of branch register br and relation $\rightarrow_{pc}$ is a reversible update of program counter pc. This is a requirement for any machine architecture to be truly reversible.

The concept of reversible updates provides a *generic instruction format* for a reversible instruction set architecture (ISA). Reversible updates are particularly well suited for implementation in reversible hardware or reversible programming languages (*e.g.*, a self-interpreter for a reversible language [19]). As f is generally irreversible, a realization in reversible hardware usually requires the addition of *garbage bits* to the output and constant presets to the input. For reversible implementation, there is a direct parallel to the Bennett trick [2] in which a circuit f' computing $f(y)$ *with* garbage bits and constant presets can be inverted to yield y again and no garbage bits. Fig. 5 shows the general implementation strategy

[2] For example, with integer multiplication `*` and integer division `/` defined on $\mathbb{Z}\backslash\{0\}$, we have $(a * b) / b = a$, but $(c / b) * b \neq c$ if b does not divide c.

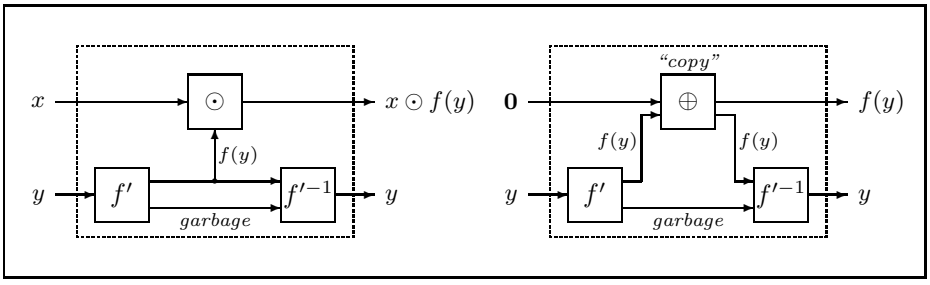


Fig. 5. Reversible implementation of reversible update

for reversible updates (constant presets omitted). Also shown is the case of a copy operator and $x = 0$, when the strategy defaults to being the Bennett trick. We assume that $copy(x, z) = (x \odot z, z)$ can be implemented without garbage bits. This assumption is not unreasonable as *copy* is injective, and in general we expect injective functions to be implementable without garbage generation. In the case of $\odot = \oplus$, *copy* is implementable by a Feynman gate [6].

Formal Properties. Now, we present the main theorems regarding the abstract machine and its instruction set. The first theorem states that the semantics of an execution step (Fig. 2) is forward and backward deterministic if every instruction in the machine's instruction set *Inst* satisfy certain conditions. The second theorem states that an execution step can be reversed in a particular simple way. The interested reader can find proof sketches in App. A.

Theorem 1 (Forward and Backward Determinism). *Assume that the semantics of every instruction $i \in Inst$ is a reversible update with the restriction that pc and $M(pc)$ must be preserved over $\rightarrow_{inst}$. Then, the semantics of an execution step, $\rightarrow_1$, is forward and backward deterministic:*

1. *Forward determinism:* $\forall \sigma, \sigma', \sigma'' \in \Sigma \,.\, \sigma \rightarrow_1 \sigma' \wedge \sigma \rightarrow_1 \sigma'' \;\Rightarrow\; \sigma' = \sigma''$
2. *Backward determinism:* $\forall \sigma, \sigma', \sigma'' \in \Sigma \,.\, \sigma' \rightarrow_1 \sigma \wedge \sigma'' \rightarrow_1 \sigma \;\Rightarrow\; \sigma' = \sigma''$.

This corresponds to saying that $\rightarrow_1$ is an injective function. Similar results apply to $\rightarrow_{pc}$ and $\rightarrow_{ie}$. We define the function $rev : \Sigma \rightarrow \Sigma$ by $rev = (\rightarrow_{pc}) \circ flip$ to reverse the execution direction and update the program counter.

Theorem 2 (Local Invertibility). *Assume Thm. 1 holds for instruction set Inst, and that $\forall i \in Inst \,.\, inv(i) \in Inst$. Individual execution steps are reversible:*

$$\forall \sigma, \sigma' \in \Sigma \;.\; \sigma \rightarrow_1 \sigma' \;\Rightarrow\; rev(\sigma') \rightarrow_1 rev(\sigma).$$

An instruction set for which these two theorems hold is *reversible*, specifically reversible on the abstract machine of Sect. 2.

Theorem 3. *PISA, as presented here and in [7, App.B], without the I/O instructions, is a reversible instruction set.*

Thm. 1 states what is usually meant by reversibility. The restrictions on the instruction set illustrate a central point of this paper: instruction execution and control logic should be kept separate. Even if an instruction is a reversible update of the program counter, recovering a unique previous program counter may no longer be possible. If an instruction modifies either pc or $M(pc)$, we can no longer uniquely identify which instruction was executed from the resulting state. This means backwards non-determinism and irreversibility.

Thm. 1 and 2 guarantee some surprising properties for programs written in reversible machine code. No matter what the program does, regardless of the programmer's intent, execution of the program is guaranteed to be reversible. Furthermore, if the state space is *finite*, programs either terminate or eventually return to the precise starting state, looping forever.

4 Programming the Reversible Processor

Rather than giving a general programming methodology, we explain the main points by examining an example program written in PISA (Fig. 6). The program shows some key aspects of reversible programming. The subroutine `Fall` implements a simple discrete physical simulation of an object that falls through a vacuum. The object has two associated variables, h and v, giving its height in meters and downward velocity in meters per second, respectively. The two variables are calculated for each second ($t = 1, 2, ...$) of free fall according to the following equations where g is the gravitational acceleration at sea level (approx. 10 m/s^2). Initially, $v_0 = 0$ and h_0 is the height from which the object is dropped.

$$v_t = v_{t-1} + g$$
$$h_t = h_{t-1} - v_t + \frac{g}{2}$$

The simulation is coded as a subroutine, callable from anywhere in the program. For simplicity, the subroutine updates two registers (`v`, `h`) instead of recording the values of each iteration in an array. We use symbolic names for registers (`v`, `h`,...) instead of (`$1`, `$2`,...). We will now explain the implementation of the subroutine (labels `27-36`) and then subroutine call and uncall (labels `10-12`, `18-21`). The subroutine simulates a free fall lasting t_{end} seconds. The program is loaded into memory at the locations indicated by the labels in Fig. 6.

The *paired branch* approach is used for reversible jumps [12,7]. Jumping from one branch instruction to another that points back at the calling branch clears the branch register and resumes normal sequential execution.

A *reversible unconditional jump* between labels `27` and `36` is implemented by the paired branch instructions `BRA 9` and `BRA -9`. To illustrate the reversibility of this control-flow structure, let control $C_0 = (36, 0, 1)$ in a state (M, R, C_0). The execution of `BRA -9` at label `36` with $\rightarrow_1$ leaves M and R unchanged, but changes C_0 to $C_1 = (27, -9, 1)$, which is a jump from label `36` to `27`. Next, the execution of `BRA 9` at label `27` changes C_1 to $C_2 = (28, 0, 1)$ and normal sequential execution is resumed. Now, reverse the execution direction: $rev((M, R, C_2)) = (M, R, C_3)$

```
Call Fall:                      Subroutine Fall:
 10:  ADDI h 1000                27:  BRA 9
 11:  ADDI tend 5                28:  SWAPBR rtn      ; br <=> rtn
 12:  BRA  16  ; call            29:  NEG    rtn      ; rtn=-rtn
                                 30:  BGTZ t 5        ; t > 0?
Uncall Fall:                     31:    ADDI v 10     ;   v += 10
 18:  ADDI v 40                  32:    ADDI h 5      ;   h += 5
 19:  ADDI t 4                   33:    SUB  h v      ;   h -= v
 20:  ADDI tend 4                34:    ADDI t 1      ;   t += 1
 21:  RBRA 7   ; uncall          35:  BNE t tend -5   ; t ≠ tend?
                                 36:  BRA -9
```

Fig. 6. Example program: free-falling object

where $C_3 = (27, 0, -1)$. Executing `BRA 9` gives $C_4 = (36, -9, -1)$, which is the reverse jump from label `27` to `36`, and then `BRA -9` gives $C_5 = (35, 0, -1)$, and we return to the initial state: $rev((M, R, C_5)) = (M, R, C_0)$. This shows how paired branch instructions implement reversible unconditional jumps.

A *reversible loop* structure implements the main part of the simulation. Conceptually, this control-flow structure has *two* conditionals to maintain [19]: a *loop-entry* and a *loop-exit* conditional. The *loop-entry* condition must be *true* when entering the loop and *false* after each subsequent execution of the loop body. Symmetrically, the *loop-exit* condition will have to be *false* in all except the last iteration when it becomes *true* and the loop is exited. We implement the loop using the paired branch approach (labels `30-35`): at loop entry we expect $t = 0$ and at loop-exit $t = t_{end}$. When the loop is repeated by jumping from `35` to `30`, `BNE` and `BGTZ` are both executed because $0 < t < t_{end}$. The instructions at `30` and `35` are negations of the *loop-entry* and *loop-exit* conditions, respectively. The paired offsets are `5` and `-5`. Branch instructions `BNE` and `BGTZ` are identical to `BEQ` (Fig. 3), except that they branch on $\neq$ and > 0 instead of $=$.

A *reversible subroutine* should be callable from different places in a program. For this purpose, the paired branch approach does not suffice as a "callee branch" can only point back to a single "caller branch". When a non-recursive subroutine is called, the "return address" in the guise of the branch register is swapped with a *zero*-cleared return register (here, `rtn`), using instruction `SWAPBR` and changing the sign using instruction `NEG` (labels `28-29`). After the swap $br = 0$, which means that normal sequential execution is resumed and the subroutine body is executed. The body is enclosed in paired branches, as discussed above, so at the end of the subroutine execution we reach the same `SWAPBR` instruction at which we entered the subroutine. The effect of this will be return to the caller. Recursive subroutines are implemented by maintaining a call stack with "return addresses" instead of a single return register `rtn`.

We can now solve the following simulation problem: an object is dropped from the top of a tower 1000 m in height. What is the height of the object after a free

fall of 5 s? This can be calculated (labels `10-12`) by initializing registers `h` and `tend` with the corresponding values and calling the subroutine, implemented by an unconditional jump `BRA` to the procedure entry (label `28`). Recall that all registers are initially zero-cleared so register `v` already has the required value.

With the same subroutine, we can also solve the following *inverse simulation problem*: after a free fall of 4 s, an object reaches the ground with a downward velocity of 40 m/s. How tall is the tower? To solve this problem, we change the execution direction and run the forward simulation *backwards* (labels `18-21`). The `RBRA` instruction (Fig. 3) changes the direction bit and runs the subroutine backwards, *i.e.* it *uncalls* the subroutine. Sharing the forward and backward code is possible because the abstract machine allows changing the execution direction. Note that forward and backward computation of the subroutine take the same number of execution steps, and thus there is no penalty for running the forward code backward, or vice versa.

An abstract machine may have a forward and backward deterministic execution relation, which means that no information is lost and the machine can be implemented on reversible hardware. However, as long as there is no facility to change the machine's execution direction while running, we cannot share the code of a subroutine and its inverse. In this case, it will be necessary to implement two subroutines—one for forward and another for backward computation.

5 Related Work

A great deal of the previous work on reversible machine code suffered from deficiencies. In [3], a history trace was used for reversibility as suggested earlier [13]; in [12] data modification was reversible, but the control logic was unclear and quite probably irreversible; and in [8] the control logic and certain control flow instructions were irreversible. We stress that even if each instruction is reversible, this is *not* sufficient to ensure reversibility of the complete architecture.

In contrast, PISA is the newest and most complete design of a reversible architecture [16]. The description of the instruction set is informal [7], but our formalization has been shown to be reversible as described in this paper. This makes PISA the only truly reversible practical programmable architecture. Reversible logic gates [9,6] and reversible logic circuits [4,5] have been studied.

A high-level imperative language for reversible programming, Janus, has recently been formalized and confirmed to be reversible [19]. One of the earliest works considering reversible subroutines was [15].

Reversible languages, such as Janus and PISA, allow efficient standard and inverse computation. A general method for inverse computation is the Universal Resolving algorithm [1], which also allows inverse computation of programs that do not implement injective functions, but is less efficient due to the search space.

6 Conclusions and Future Work

In this paper, we have formally described an abstract machine suitable for reversible computing. We clarified that for machine code to be fully reversible both

the underlying control logic as well as each instruction must be reversible, and that forward and backward code can be shared if the machine allows a program to switch between standard (forward) and inverse (backward) computation. We have shown a general class of instruction sets that are reversible, building on our concept of reversible updates. We have shown that PISA, as a member of this class, is reversible. Finally, we gave programming principles for the abstract machine architecture formalized in this paper.

We posit that reversible computing is sufficiently different from irreversible computing to warrant consideration as a separate paradigm. As such, we suggest a "principles first" approach to reversible machine architectures. Designing and implementing hardware for a minimal instruction set based on the principles outlined in this paper could serve as a first effort in this direction. Work on a translator from the structured high-level reversible programming language Janus [19] to PISA is currently in progress.

References

1. Abramov, S.M., Glück, R.: The universal resolving algorithm and its correctness: inverse computation in a functional language. Science of Computer Programming 43(2-3), 193–229 (2002)
2. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
3. Cezzar, R.: Design of a processor architecture capable of forward and reverse execution. In: Proceeding of IEEE SOUTHEASTCON'91, vol. 2, pp. 885–890 (1991)
4. De Vos, A., Van Rentergem, Y., De Keyser, K.: The decomposition of an arbitrary reversible logic circuit. Journal of Physics A: Mathematical and General 39(18), 5015–5035 (2006)
5. Desoete, B., De Vos, A.: A reversible carry-look-ahead adder using control gates. Integration, the VLSI Journal 33(1), 89–104 (2002)
6. Feynman, R.P.: Reversible computation and the thermodynamics of computing (chapter 5). In: Feynman Lectures on Computation, pp. 137–184. Addison-Wesley, Reading, MA, USA (1996)
7. Frank, M.P.: Reversibility for Efficient Computing. PhD thesis, MIT (1999)
8. Frank, M.P., Rixner, S.: Tick: A simple reversible processor (6.371 project report). Online term paper, MIT EECS Department (1996)
9. Fredkin, E., Toffoli, T.: Conservative logic. Intl. J. Theor. Phy. 21, 219–253 (1982)
10. Glück, R., Kawabe, M.: A program inverter for a functional language with equality and constructors. In: Ohori, A. (ed.) APLAS 2003. LNCS, vol. 2895, pp. 246–264. Springer, Heidelberg (2003)
11. Glück, R., Kawabe, M.: A method for automatic program inversion based on LR(0) parsing. Fundamenta Informaticae 66(4), 367–395 (2005)
12. Hall, J.S.: A reversible instruction set architecture and algorithms. In: Workshop on Physics and Computation. Proceedings, pp. 128–134. IEEE Press, New York (1994)
13. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
14. Mogensen, T.Æ.: Semi-inversion of guarded equations. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 189–204. Springer, Heidelberg (2005)

15. Reilly, E.D., Federighi, F.D.: On reversible subroutines and computers that run backwards. Communications of the ACM 8(9), 557–558 (1965)
16. Vieri, C.J.: Reversible Computer Engineering and Architecture. PhD thesis, MIT (1999)
17. Vieri, C.J., Ammer, M.J., Frank, M.P., Magoulis, N., Knight, T.: A fully reversible asymptotically zero energy processor. In: Proceedings of the ISCA workshop (1998)
18. Winskel, G.: The Formal Semantics of Programming Languages. MIT Press, Cambridge (1993)
19. Yokoyama, T., Glück, R.: A reversible programming language and its invertible self-interpreter. In: Partial Evaluation and Program Manipulation. Proceedings, pp. 144–153. ACM Press, New York (2007)

A Appendix: Proof Sketches

Proof Sketch for Thm. 1. Each instruction is a reversible update of some part of the state (*e.g.*, a register), and can therefore be considered an injective function on the state. Thus, $\rightarrow_{inst}$ is injective for each individual instruction. The restriction that *pc* is preserved ensures that the *program counter* of the previous state is unique, as $\rightarrow_{pc}$ is injective by simple case analysis. The restriction that $M(pc)$ is preserved ensures that the previous *instruction* executed is also unique, so the injectivity of $\rightarrow_{inst}$ for this instruction then gives the injectivity of $\rightarrow_{ie}$. The composition of two injective functions is itself injective, so $\rightarrow_1$ is injective.

Proof Sketch for Thm. 2. It can easily be shown that

$$(\rightarrow_{pc})^{-1} = \mathit{flip} \circ (\rightarrow_{pc}) \circ \mathit{flip} \quad \text{and} \quad (\rightarrow_{ie})^{-1} = \mathit{flip} \circ (\rightarrow_{ie}) \circ \mathit{flip}.$$

Note that *flip* and *rev* are self-inverse functions $\Sigma \rightarrow \Sigma$. In the equation sequence below, the term to be reduced/expanded in each successive equation is underlined to make it easier to understand. This is equivalent to the claim of Thm. 2.

$$\begin{aligned}
\underline{(\rightarrow_1)^{-1}} &= (\rightarrow_{ie})^{-1} \circ \underline{(\rightarrow_{pc})^{-1}} \;=\; (\rightarrow_{ie})^{-1} \circ \mathit{flip} \circ \underline{(\rightarrow_{pc}) \circ \mathit{flip}} \\
&= \underline{(\rightarrow_{ie})^{-1} \circ \mathit{flip} \circ \mathit{rev}} \;=\; \underline{\mathit{id}} \circ (\rightarrow_{ie})^{-1} \circ \mathit{flip} \circ \mathit{rev} \\
&= (\rightarrow_{pc}) \circ \underline{(\rightarrow_{pc})^{-1}} \circ (\rightarrow_{ie})^{-1} \circ \mathit{flip} \circ \mathit{rev} \\
&= \underline{(\rightarrow_{pc}) \circ \mathit{flip}} \circ (\rightarrow_{pc}) \circ \mathit{flip} \circ (\rightarrow_{ie})^{-1} \circ \mathit{flip} \circ \mathit{rev} \\
&= \mathit{rev} \circ (\rightarrow_{pc}) \circ \mathit{flip} \circ \underline{(\rightarrow_{ie})^{-1}} \circ \mathit{flip} \circ \mathit{rev} \\
&= \mathit{rev} \circ (\rightarrow_{pc}) \circ \underline{\mathit{flip} \circ \mathit{flip}} \circ (\rightarrow_{ie}) \circ \underline{\mathit{flip} \circ \mathit{flip}} \circ \mathit{rev} \\
&= \mathit{rev} \circ \underline{(\rightarrow_{pc}) \circ (\rightarrow_{ie})} \circ \mathit{rev} \;=\; \mathit{rev} \circ (\rightarrow_1) \circ \mathit{rev}
\end{aligned}$$

Proof Sketch for Thm. 3. By case analysis on $\rightarrow_{inst}$, each instruction in PISA (without the four I/O instructions *READ*, *SHOW*, *EMIT*, *TAKE*) can be shown to be a reversible update. No instruction in PISA can modify *pc* and memory access is specifically restricted from modifying the current instruction, $M(pc)$. Thus, Thm. 1 holds for PISA. The inverse of each instruction is easily specified by the reversible update as instruction in PISA, and so Thm. 2 holds.

A Fast Algorithm for Path 2-Packing Problem

Maxim A. Babenko*

Dept. of Mechanics and Mathematics, Moscow State University,
Vorob'yovy Gory, 119899 Moscow, Russia
mab@shade.msu.ru

Abstract. Let $G = (V_G, E_G)$ be an undirected graph, $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a collection of disjoint subsets of nodes. Nodes in $T_1 \cup \ldots \cup T_k$ are called *terminals*, other nodes are called *inner*. By a $\mathcal{T}$-path P we mean an undirected path such that P connects terminals from distinct sets in $\mathcal{T}$ and all internal nodes of P are inner. We study the problem of finding a maximum cardinality collection $\mathcal{P}$ of $\mathcal{T}$-paths such that at most two paths in $\mathcal{P}$ pass through any node $v \in V_G$. Our algorithm is purely combinatorial and achieves the time bound of $O(mn^2)$, where $n := |V_G|$, $m := |E_G|$.

1 Introduction

For an undirected graph G we write V_G, E_G to denote the set of nodes and the set of edges of G, respectively. Let G be an undirected graph, $T \subseteq V_G$ be a distinguished set of nodes (called *terminals*). All nodes in $V_G - T$ are called *inner*. Furthermore, suppose that T is partitioned into a collection of sets $\mathcal{T} = \{T_1, \ldots, T_k\}$. By a *$\mathcal{T}$-path* we mean a path P such that: (i) the endpoints of P are terminals belonging to distinct sets T_α and T_β ($1 \le \alpha \ne \beta \le k$); (ii) all internal nodes of P are inner. In particular, the definition implies that the endpoints of P are distinct. Let $\mathcal{P}(G, \mathcal{T})$ denote the family of all $\mathcal{T}$-paths in G. A *packing* (of $\mathcal{T}$-paths) is a function $f\colon \mathcal{P}(G, \mathcal{T}) \to \mathbb{R}_+$. The *size* $||f||$ of a packing f is the sum $\sum_P f(P)$. Put

$$\zeta^f := \sum \left(f(P) \cdot \chi^{V_P} : P \in \mathcal{P}(G, \mathcal{T}) \right)$$

thus forming a function $\zeta^f\colon V_G \to \mathbb{R}_+$. Here V_P, E_P denote the (multi-) sets of nodes and edges of a path P, respectively; χ^U denotes the indicator of a (multi-) set U. A packing of the maximum size (in a certain class) is called *maximum*.

Given a *node capacities* vector $c\colon V_G \to \mathbb{Z}_+$, we call f a *c-packing* if $\zeta^f(v) \le c(v)$ holds for all $v \in V_G$. Consider the following problem:

(P) *Find a maximum integer c-packing $f\colon \mathcal{P}(G, \mathcal{T}) \to \mathbb{Z}_+$.*

For the case $c(v) \equiv 1$, this problem was earlier studied in [3] (see also [1,5,6]). It is known that (P) reduces to the linear matroid matching problem and, hence,

* Supported by RFBR grants 03-01-00475, 05-01-02803, and 06-01-00122.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 70–81, 2007.

is polynomially solvable. However, the actual time bound that comes from the general algorithm is unsatisfactory. Recently, an $O(n^6)$-algorithm for (P) (in case $c(v) \equiv 1$) was proposed, see [1] (hereinafter n denotes the number of nodes in a graph and m — the number of edges). Another algorithmic approach (applicable to a somewhat more general case of group-labelled graphs) can be found in [4].

In this work we consider the case $c(v) \equiv 2$, which seems to be more tractable. The paper is organized as follows. Section 2 discusses some useful properties of 2-packings. The general algorithmic scheme is presented in Section 3. The complexity of the suggested procedure is estimated in Section 4. Section 5 summarizes the results and describes several open questions.

2 Basic Properties of 2-Packings

In case all node capacities are even, (P) exhibits the following nice property: the optimum does not change if we allow arbitrary real-valued packings f, see e.g. [4]. This simple fact allows to employ the standard linear programming techniques, linear duality theory in particular.

Let us generalize (P) as follows. Consider a collection $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of subsets of V_G such that: (i) the sets in $\mathcal{Q}$ are pairwise disjoint; (ii) $T_i \subseteq Q_i$ holds for all $1 \leq i \leq k$. The sets Q_i are called *islands* and $\mathcal{Q}$ is called an *island collection*. Hereinafter, $\delta(A)$ (resp. $\gamma(A)$) denotes the set of edges (or arcs) e such that exactly one (resp. both) endpoints of e are in A. A $\mathcal{T}$-path P is said to be $\mathcal{Q}$-*admissible* if $|E_P \cap \delta(Q_i)| \leq 1$ holds for all $1 \leq i \leq k$. A 2-packing f is said to be a $(\mathcal{Q}, 2)$-*packing* if P is a $\mathcal{Q}$-admissible path whenever $f(P) > 0$.

Consider a new problem as follows:

(QP) *Find a maximum integer* $(\mathcal{Q}, 2)$-*packing* $f\colon \mathcal{P}(G, \mathcal{T}) \to \mathbb{Z}_+$.

A function $w\colon V_G \to \mathbb{R}_+$ is called a $(\mathcal{Q}, 2)$-*covering* if $w(V_P) \geq 2$ holds for all $\mathcal{Q}$-admissible $\mathcal{T}$-paths P. (We assume that any real-valued function $h\colon U \to \mathbb{R}$ is extended to 2^U: $h(A) := \sum_{a \in A} h(a)$ for all $A \subseteq U$.) The *size* $||w||$ of a covering c is $c(V_G)$. A covering of the minimum size (in a certain class) is called *minimum*.

Viewing (QP) as a linear program (obtained by dropping the integrality requirement) one can write the dual problem and the complementary slackness conditions as follows:

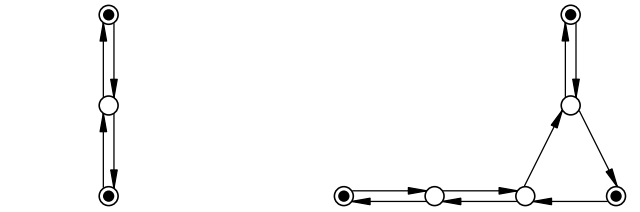
(QC) *Find a minimum* $(\mathcal{Q}, 2)$-*covering* $w\colon V_G \to \mathbb{R}_+$.

(CS1) *If* $f(P) > 0$ *for some* $P \in \mathcal{P}(G, \mathcal{T})$ *then* $w(V_P) = 2$.

(CS2) *If* $w(v) > 0$ *for some* $v \in V_G$ *then* $\zeta^f(v) = 2$.

Clearly, any integer packing $f\colon \mathcal{P}(G, \mathcal{T}) \to \mathbb{Z}_+$ may be given by a multiset $\mathcal{P}$ of $\mathcal{T}$-paths with $|\mathcal{P}| = ||f||$. We shall use f and $\mathcal{P}$ notations interchangeably and will make no distinction between these two forms.

One may consider two types of packings with a very simple structure. Firstly, let P be a $\mathcal{T}$-path. Hereinafter we denote the reverse path by P^{-1}. Clearly, $\{P, P^{-1}\}$ is 2-packing; we call it a *double path* generated by P, see Fig. 1(a).



(a) A double path (b) An odd star

Fig. 1. Elements of a canonical 2-packing

The other class is obtained as follows. Fix a certain subset of terminals $A = \{t_0, \ldots, t_{s-1}\}$, a collection $\mathcal{S} = \{P_0, \ldots, P_{s-1}\}$, where P_i is a $\mathcal{T}$-path from t_i to t_{i+1} (indices are assumed to be taken modulo s), and a sequence of non-negative integers $L = (l_0, \ldots, l_{s-1})$. For each $0 \le i < s$ the nodes of P_i, except for the first $l_i + 1$ and the last $l_{i+1} + 1$ nodes, are called *proper*. By a *q-prefix* of a path P we mean a path obtained by taking the first q edges of P. Suppose that $(A, \mathcal{S}, L)$ obeys the following properties: (i) for each $0 \le i < s$ the l_i-prefix of P_i coincides with the l_i-prefix of P_{i-1}^{-1}; (ii) all proper nodes are distinct; (iii) for all $0 \le i < s$ the l_i-prefixes of P_i are node-disjoint. Then clearly $\mathcal{S}$ is a 2-packing; we call it a *star*. In case s is odd, $\mathcal{S}$ is called an *odd* star. Refer to Fig. 1(b) for an example.

We call a 2-packing $\mathcal{P}$ *canonical* if $\mathcal{P}$ is a disjoint union of double paths and odd stars. A maximum 2-packing constructed by our algorithm is canonical. Hence, we shall prove the following statement:

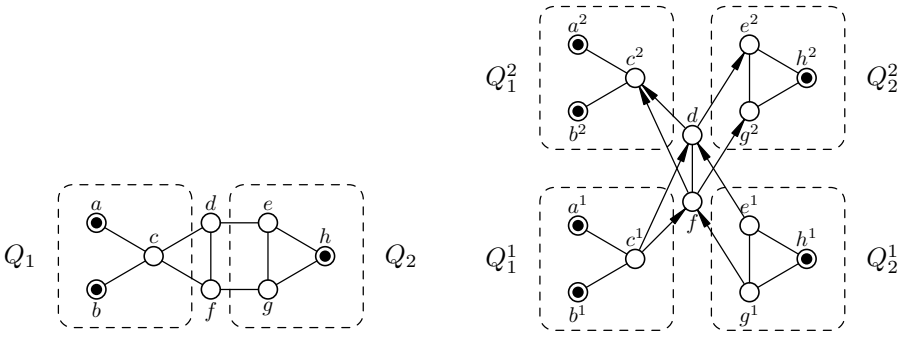
Theorem 1. *For any undirected graph G and a collection of disjoint terminal sets $\mathcal{T}$ there exists, and can be found in $O(mn^2)$ time, a maximum 2-packing $\mathcal{P}$ of $\mathcal{T}$-paths that is canonical.*

A (possibly empty) path L is called a *half $\mathcal{T}$-path* if L starts in a node from T and all intermediate nodes of L are in $V_G - T$. (In contrast to a $\mathcal{T}$-path, a half $\mathcal{T}$-path may end in any node. Moreover, the first and the last nodes of L may coincide.)

Let $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ denote an island collection; put $Q := Q_1 \cup \ldots \cup Q_k$ and $Z := V_G - Q$. The notion of $\mathcal{Q}$-admissibility is extended to half $\mathcal{T}$-paths in a natural manner. Note that any $\mathcal{Q}$-admissible half $\mathcal{T}$-path is exactly of one of the following three kinds:

(H1) a path that is fully contained in one of the islands $Q_\alpha \in \mathcal{Q}$;
(H2) a path that starts in an island $Q_\alpha \in \mathcal{Q}$, crosses the cut $\delta(Q_\alpha)$, and ends in Z;
(H3) a path that starts in an island $Q_\alpha \in \mathcal{Q}$, crosses the cut $\delta(Q_\alpha)$, traverses some (possibly empty) set of nodes in Z, crosses some other cut $\delta(Q_\beta)$ for $\alpha \ne \beta$, and ends in Q_β.

The next lemma constitutes the core of our island-extension approach.



(a) Graph G and a pair of islands Q_1 and Q_2

(b) Graph $G(\mathcal{Q})$ (undirected edges denote pairs of oppositely directed arcs)

Fig. 2. Constructing $G(\mathcal{Q})$

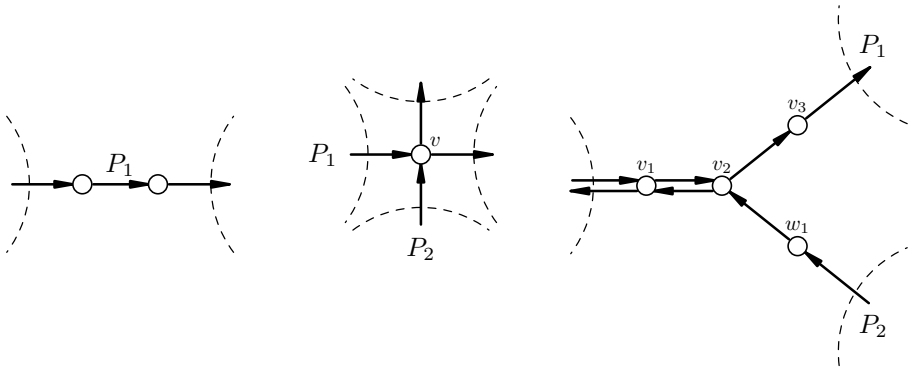
Lemma 1. *Let $\mathcal{P}$ be a $(\mathcal{Q},2)$-packing, $Q_\alpha \in \mathcal{Q}$ be an island, $A \subseteq Z$ be a set of nodes. Put $Q'_\alpha := Q_\alpha \cup A$, $\mathcal{Q}' := \mathcal{Q} - \{Q_\alpha\} \cup \{Q'_\alpha\}$. Suppose that $\mathcal{P}$ is a $(\mathcal{Q}',2)$-packing. Also, suppose that for each $x \in A$ there is a $(\mathcal{Q}',2)$-packing $\mathcal{P}_x$ and a half $\mathcal{T}$-path L_x such that: (i) $|\mathcal{P}_x| = |\mathcal{P}|$; (ii) L_x is contained in Q'_α and ends in x; (iii) for each $y \in V_{L_x} - \{x\}$ one has $\zeta^{\mathcal{P}_x}(y) \le 1$. Now if $\mathcal{P}$ is a maximum $(\mathcal{Q}',2)$-packing then $\mathcal{P}$ is a maximum $(\mathcal{Q},2)$-packing.*

Proof. Let w denote a minimum $(\mathcal{Q}',2)$-covering. One has $|\mathcal{P}| = ||w||$. We claim that w is also a $(\mathcal{Q},2)$-covering, hence $\mathcal{P}$ is a maximum $(\mathcal{Q},2)$-packing. Suppose the contrary: there exists a $\mathcal{Q}$-admissible $\mathcal{T}$-path P in G such that $w(V_P) \le 1$. Path P cannot be $\mathcal{Q}'$-admissible. Therefore, $|E_P \cap \delta(Q'_\alpha)| \ge 2$. Clearly there is a part P_1 of P that obeys the following properties: (i) P_1 starts in $T_\beta \in \mathcal{T}$ for some $\beta \ne \alpha$; (ii) P_1 ends in a node $x \in A$; (iii) $V_{P_1} \cap Q'_\alpha = \{x\}$. Then, $w(V_{P_1}) \le w(V_P) \le 1$. Extend P_1 to a $\mathcal{Q}'$-admissible $\mathcal{T}$-path P_2 by appending L_x^{-1} to its end. By (CS2) for w and $\mathcal{P}_x$ one has $w(y) = 0$ for all $y \in V_{L_x} - \{x\}$. Therefore, $w(V_{P_2}) = w(V_{P_1}) = 1$ contradicting with the $(\mathcal{Q}',2)$-covering condition for w.

Corollary 1. *Let $\mathcal{P}$ be a $(\mathcal{Q},2)$-packing, let L be a half $\mathcal{T}$-path of type (H2) that starts in $T_\alpha \in \mathcal{T}$. Let $\zeta^{\mathcal{P}}(x) \le 1$ for all $x \in V_L$, $\zeta^{\mathcal{P}}(x) = 0$ for all $x \in V_L \cap Z$. Put $Q'_\alpha := Q_\alpha \cup V_L$, $\mathcal{Q}' := \mathcal{Q} - \{Q_\alpha\} \cup \{Q'_\alpha\}$. Then $\mathcal{P}$ is a $(\mathcal{Q}',2)$-packing. Moreover, if $\mathcal{P}$ is a maximum $(\mathcal{Q}',2)$-packing, then $\mathcal{P}$ is a maximum $(\mathcal{Q},2)$-packing.*

Proof. Since $\zeta^{\mathcal{P}}(x) = 0$ for all $x \in A := V_L \cap Z$, no path in $\mathcal{P}$ passes through any node $x \in A$, so it is clear that $\mathcal{P}$ is a $(\mathcal{Q}',2)$-packing. Putting $\mathcal{P}_x := \mathcal{P}$ for all $x \in A$ and applying Lemma 1 one gets the required result.

We also introduce an auxiliary digraph $G(\mathcal{Q})$ defined as follows. Each node $v \in Q$ is split into a pair v^1, v^2 of nodes in $G(\mathcal{Q})$. Each node $v \in Z$ corresponds to a unique node in $G(\mathcal{Q})$; we identify the latter node in $G(\mathcal{Q})$ with v. Each edge $\{u,v\} \in \gamma(Z)$ generates a pair of arcs $(u,v),(v,u)$ in $G(\mathcal{Q})$. Each edge $\{u,v\} \in \gamma(Q_i)$, $1 \le i \le k$ generates four arcs $(u^j,v^j),(v^j,u^j)$ in $G(\mathcal{Q})$, $j = 1,2$.



(a) Case (Z2) (b) Case (Z3) (c) Case (Z4)

Fig. 3. Possible types of Z-nodes

Each edge $\{u, v\} \in E_G$ with $u \in Q_i$, $1 \le i \le k$, $v \in Z$ generates arcs (u^1, v) and (v, u^2) in $G(\mathcal{Q})$. Finally, each edge $\{u, v\} \in E_G$ with $u \in Q_i$, $v \in Q_j$, $1 \le i \ne j \le k$ generates arcs (u^1, v^2), (v^1, u^2) in $G(\mathcal{Q})$. Nodes v_1, v_2 in $G(\mathcal{Q})$ (corresponding to $v \in Q$) are endowed with capacity 1; nodes $v \in Z$ in $G(\mathcal{Q})$ are endowed with capacity 2. An example is depicted in Fig. 2.

Put $A^i := \{v^i \mid v \in A\}$ for any set $A \subseteq Q$. One can extend the notion of $\mathcal{T}$-path to $G(\mathcal{Q})$: by such a path we mean a directed path connecting a node in T^1_α with a node in T^2_β for some α, β $(1 \le \alpha \ne \beta \le k)$. By merging copies of nodes and arcs, each directed T^1–T^2 path $\widehat{P}$ in $G(\mathcal{Q})$ can be transformed into a path P in G with both endpoints in T. The resulting path P is a $\mathcal{Q}$-admissible $\mathcal{T}$-path in G iff $\widehat{P}$ is a $\mathcal{T}$-path in $G(\mathcal{Q})$. Vice versa, any $\mathcal{Q}$-admissible $\mathcal{T}$-path P in G can be lifted to give a $\mathcal{T}$-path $\widehat{P}$ in $G(\mathcal{Q})$.

3 Algorithm

The algorithm for constructing a maximum 2-packing consists of *iterations*. It maintains a certain current 2-packing $\mathcal{P}_0$. Each iteration aims to increase $|\mathcal{P}_0|$. Initially $\mathcal{P}_0 := \emptyset$; clearly, $O(n)$ iterations are possible. An iteration consists of a sequence of *phases*. An island collection $\mathcal{Q}_0 = \{Q_1, \ldots, Q_k\}$ is maintained such that $\mathcal{P}_0$ is a $(\mathcal{Q}_0, 2)$-packing. At the beginning of an iteration $Q_i := T_i$ for all $1 \le i \le k$. A phase aims to increase either $|\mathcal{P}_0|$ (in which case a new iteration starts) or $\sum_i |Q_i|$. Clearly, only $O(n)$ phases are possible during any iteration. The following invariant holds:

(J1) If $\mathcal{P}_0$ is a maximum $(\mathcal{Q}_0, 2)$-packing then $\mathcal{P}_0$ is a maximum 2-packing.

Canonicity property of a packing is too strong to be maintained directly during phases. Instead we introduce a relaxed version of it. Let $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ be an island collection, let $\mathcal{P}$ be a $(\mathcal{Q}, 2)$-packing. As earlier, put $Q := Q_1 \cup \ldots \cup Q_k$

and $Z := V_G - Q$. Then $\mathcal{P}$ is said to be *weakly $\mathcal{Q}$-canonical* if for each node $v \in Z$ exactly one of the following four cases (Z1)–(Z4) applies:

(Z1) No path from $\mathcal{P}$ passes through v.
(Z2) A unique path $P \in \mathcal{P}$ passes through v. Also, P has no intersections in Z with other paths from $\mathcal{P}$. See Fig. 3(a).
(Z3) There are exactly two paths $P_1, P_2 \in \mathcal{P}$ that pass through v. Each of P_1, P_2 intersects Z by exactly one node, namely v. Paths P_1, P_2 connect four distinct islands in $\mathcal{Q}$. See Fig. 3(b).
(Z4) There is a path $P_1 \in \mathcal{P}$ that passes through v. Let $v_1, \ldots, v_s$ be the sequence of nodes passed by P_1 in Z (in this order). There is a unique path $P_2 \in \mathcal{P}$ that has intersection with P_1 in Z. Namely, P_2 passes the nodes $w_1, \ldots, w_t, v_j, v_{j-1}, \ldots, v_1$ in Z (in this order). Here $t \geq 0$, $1 \leq j \leq s$. The nodes $v_1, \ldots, v_s, w_1, \ldots, w_t$ are distinct. Path P_1 starts in the same island where P_2 ends. No other path in $\mathcal{P}$ (except for P_1, P_2) passes through nodes $v_1, \ldots, v_s, w_1, \ldots, w_t$. Node v_j is called the *fork node* (of paths P_1 and P_2). See Fig. 3(c).

Clearly, any canonical $(\mathcal{Q}, 2)$-packing is weakly $\mathcal{Q}$-canonical (moreover, case (Z3) never applies to it).

Proofs of the next two lemmas are based on certain decomposition properties of 2-packings and due to the lack of space will appear in the full version of the paper.

Lemma 2. *Let $\mathcal{Q}$ be an island collection and $\mathcal{P}$ be a $(\mathcal{Q}, 2)$-packing that is weakly $\mathcal{Q}$-canonical. Then there exists, and can be found in $O(m)$ time, a canonical $(\mathcal{Q}, 2)$-packing $\mathcal{P}'$ with $|\mathcal{P}'| = |\mathcal{P}|$.*

Lemma 3. *Let $\mathcal{Q}$ be an island collection and $\mathcal{P}$ be a $(\mathcal{Q}, 2)$-packing that is weakly $\mathcal{Q}$-canonical. Let L be a half $\mathcal{T}$-path of type (H2) that starts in an island $Q_\alpha \in \mathcal{Q}$ and ends in a node $v \in Z$ of type (Z2). Denote by P the unique path in $\mathcal{P}$ passing through v. Let P connect some islands $Q_\beta \in \mathcal{Q}$ and $Q_\gamma \in \mathcal{Q}$, $\alpha \neq \beta$, $\alpha \neq \gamma$. Finally, suppose that $\zeta^{\mathcal{P}}(x) \leq 1$ for all $x \in V_L$. Then there exists, and can be found in $O(m)$ time, a canonical $(\mathcal{Q}, 2)$-packing $\mathcal{P}'$ with $|\mathcal{P}'| = |\mathcal{P}| + 1$.*

At the beginning of each phase the algorithm has a certain weakly $\mathcal{Q}_0$-canonical $(\mathcal{Q}_0, 2)$-packing. With the help of Lemma 2 it is turned into a canonical $(\mathcal{Q}_0, 2)$-packing in $O(m)$ time. Denote the latter packing by $\mathcal{P}_0$. The algorithm lifts each path $P \in \mathcal{P}_0$ to a (uniquely determined) $\mathcal{T}$-path $\widehat{P}$ in $G(\mathcal{Q}_0)$ thus forming a collection $\widehat{\mathcal{P}}_0$ of $\mathcal{T}$-paths in $G(\mathcal{Q}_0)$. Adding up the indicators of paths in $\widehat{\mathcal{P}}_0$ one gets a function $\varphi_0 \colon A_{G(\mathcal{Q}_0)} \to Z_+$ that is a feasible (w.r.t. node capacities) integer flow in $G(\mathcal{Q}_0)$ of value $|\mathcal{P}_0|$.

The algorithm tries to increase the size of $\widehat{\mathcal{P}}_0$ by applying the standard augmenting approach. More precisely, for an integer S–T flow $\varphi \colon A_H \to \mathbb{Z}_+$ in a digraph H endowed with integer node capacities $c \colon V_H \to \mathbb{Z}_+$ an *augmenting sequence* P (w.r.t. φ) is a sequence

$$P = (v_0, a_1, v_1, a_2, \ldots, a_l, v_l), \tag{1}$$

where $v_i \in V_H$, $a_i \in A_H$, and the following properties hold:

1. a_i connects nodes v_{i-1} and v_i $(1 \le i \le l)$;
2. if a_i leaves v_i then $\varphi(a_i) > 0$ $(1 \le i \le l)$;
3. if a_i leaves v_{i-1} and a_{i+1} leaves v_i then $\varphi(v_i) < c(v_i)$ $(1 \le i < l)$.
4. $v_0 \in S$, $v_l \in T$;
5. a_1 leaves v_0 and a_l enters v_l;
6. $\varphi(v_1) < c(v_1)$, $\varphi(v_l) < c(v_l)$.

(Here $\varphi(v)$ denotes the *flow through* v, i.e. the maximum of two values: the total incoming flow at v and total outcoming flow at v.) The arcs a_i in (1) leaving v_{i-1} are called *forward*, others (those leaving v_i) are called *backward*. The definition implies that the first and the last arcs of P are forward.

The algorithm checks for the existence of an augmenting sequence w.r.t. φ_0. In case no augmenting sequence is found, by a simple max-flow min-cut argument one can see that there exists a set B of nodes of $G(\mathcal{Q}_0)$ such that: (i) every directed T^1–T^2 path contains at least one node from B; (ii) the total capacity of nodes in B is $|\mathcal{P}_0|$. Construct an integer weight function $w\colon V_G \to \mathbb{Z}_+$ as follows: (i) for every $v \in Z$ put $w(v) := 2$ if $v \in B$, $w(v) := 0$ otherwise; (ii) for every $v \in Q$ put $w(v) := |B \cap \{v^1, v^2\}|$.

Lemma 4. *w is a minimum $(\mathcal{Q}_0, 2)$-covering.*

Proof. First prove that w is indeed a $(\mathcal{Q}_0, 2)$-covering. Suppose towards contradiction that there is a $\mathcal{Q}_0$-admissible $\mathcal{T}$-path P in G such that $w(V_P) \le 1$. Lift P to a directed T^1–T^2 path $\widehat{P}$ in $G(\mathcal{Q}_0)$. Since $B \cap V_{\widehat{P}} \ne \emptyset$, it follows that $w(V_P) \ge 1$, therefore $w(V_P) = 1$. Hence, all Z-nodes of $\widehat{P}$ do not belong to B and there is exactly one Q-node of $\widehat{P}$ belonging to B. Consider the path P^{-1}. Its image $\widehat{P}^{-1}$ in $G(\mathcal{Q}_0)$ does not contain nodes from B — a contradiction. The minimality of w follows from the equality $||w|| = |\mathcal{P}_0|$ and the fact that (QC) is dual to (QP).

Let R be an augmenting sequence. We follow along R while maintaining a current node q in $G(\mathcal{Q}_0)$, a collection $\widehat{\mathcal{P}}$ of paths in $G(\mathcal{Q}_0)$, and a path $\widehat{L}$ in $G(\mathcal{Q}_0)$ ending in q. Combine $\widehat{L}$ (viewed as a sequence of nodes and arcs) with the suffix of R starting at q and denote the resulting sequence by R_q. The following invariants (in addition to (J1)) hold:

(J2) The images P (in G) of paths $\widehat{P} \in \widehat{\mathcal{P}}$ form a weakly $\mathcal{Q}_0$-canonical $(\mathcal{Q}_0, 2)$-packing (denoted by $\mathcal{P}$).
(J3) R_q is an augmenting sequence w.r.t. the flow corresponding to $\widehat{\mathcal{P}}$.
(J4) The image of $\widehat{L}$ (in G) is a $\mathcal{Q}_0$-admissible half $\mathcal{T}$-path (denoted by L).

Initially $\widehat{\mathcal{P}} := \widehat{\mathcal{P}}_0$, q is the first node of R, and $\widehat{L}$ is the empty path starting and ending at q. A generic way move along R works as follows. Examine the type of arc a that follows q in R. If a is a forward arc, then execute a *forward step*: add arc a to the end of $\widehat{L}$. Otherwise, there exists a path $\widehat{P} \in \widehat{\mathcal{P}}$ containing a.

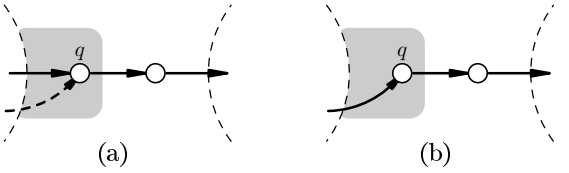


Fig. 4. Augmentation step in case (Z2): $\alpha = \beta$

Let $\widehat{P}$ be split by q into the parts $\widehat{P}_1$ and $\widehat{P}_2$ (that is, $\widehat{P} = \widehat{P}_1 \circ \widehat{P}_2$). Perform a *backward* step replacing $\widehat{\mathcal{P}}$ by $\widehat{\mathcal{P}} - \left\{\widehat{P}\right\} \cup \left\{\widehat{L} \circ \widehat{P}_2\right\}$ and putting $\widehat{L}$ to be $\widehat{P}_1$ with the last arc a removed. Once the end of R is reached, L is a $\mathcal{Q}_0$-admissible $\mathcal{T}$-path that can be added to $\mathcal{P}$, thus completing the iteration.

The above changes always preserve (J3). Property (J1) is preserved as long as $\mathcal{Q}_0$ is unchanged. The remaining of the section is devoted to explaining of how to maintain (J2) and (J4).

Firstly, if before an augmentation step L is of type (H1) or (H3), then (J2) and (J4) will hold after this step.

Secondly, suppose that L is of type (H2) before an augmentation. Let $Q_\alpha \in \mathcal{Q}$ denote the island where L starts. One may assume that $V_L \cap Z = \{q\}$. Indeed, otherwise let r denote the first node of L belonging to Z, $r \neq q$. Then either $\zeta^{\mathcal{P}}(r) = 0$ or $\zeta^{\mathcal{P}}(r) = 1$, so either (Z1) or (Z2) case applies to the appropriate prefix of R, see below. In both of these cases the current phase completes.

We proceed by case splitting according to the definition of weak canonicity.

Case (Z1). No path in $\mathcal{P}$ passes through $q \in Z$. Then by Corollary 1 one can add node q to Q_α; the current phase completes. (Note that extension of islands cannot violate weak $\mathcal{Q}_0$-canonicity of the current packing, so the next phase gets a weakly $\mathcal{Q}_0$-canonical $(\mathcal{Q}_0, 2)$-packing $\mathcal{P}$, as required.)

Case (Z2). A unique path $P \in \mathcal{P}$ passes through q. Also, P has no intersections in Z with other paths from $\mathcal{P}$. Let P go from $Q_\beta \in \mathcal{Q}_0$ to $Q_\gamma \in \mathcal{Q}_0$ for $\beta \neq \gamma$. In case $\alpha \neq \beta$ and $\alpha \neq \gamma$, apply Lemma 3 and complete the current iteration. If $\alpha = \beta$, then by Lemma 1 one can extend the island Q_β by adding the nodes of P up to q (including q), see Fig. 4. Otherwise one can similarly extend the island Q_γ by adding the nodes of P starting from q (including q). This completes the phase.

Case (Z3). Let $P_1, P_2 \in \mathcal{P}$ be the paths that pass through q. Since P_1 and P_2 connect four distinct islands, property (J2) can always be preserved (possibly with the help of path switching at q), The updated current path is of type (H1); node q is of type (Z3) or (Z4); property (J4) remains valid.

Case (Z4). Denote the corresponding fork node (see the definition of (Z4)) by f. Two subcases are possible depending of whether $\zeta^{\mathcal{P}}(q) = 1$ or $\zeta^{\mathcal{P}}(q) = 2$.

Subcase (Z4-1). Let $\zeta^{\mathcal{P}}(q) = 1$, denote by P_1 the unique path in $\mathcal{P}$ passing through q. Split P at q into the parts P_{11} and P_{12} ($P_1 = P_{11} \circ P_{12}$). One may assume that f belongs to P_{11}. Let P_1 go from an island Q_β to Q_γ. In case $\alpha \neq \beta$,

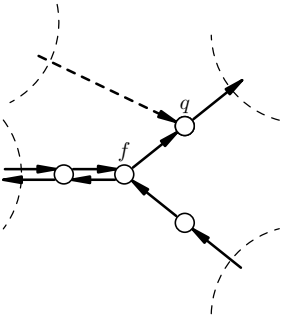


Fig. 5. Augmentation step in case (Z4-1): $\alpha \neq \beta$, $\alpha \neq \gamma$

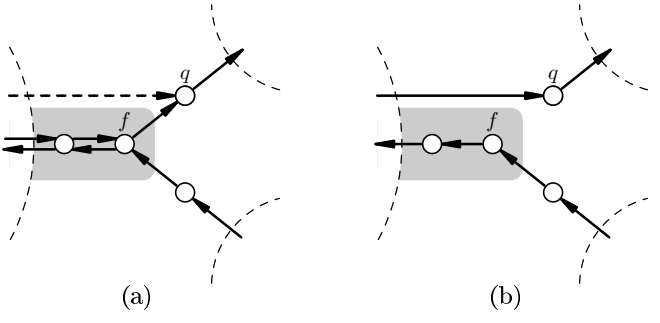


Fig. 6. Augmentation step in case (Z4-1): $\alpha = \beta$

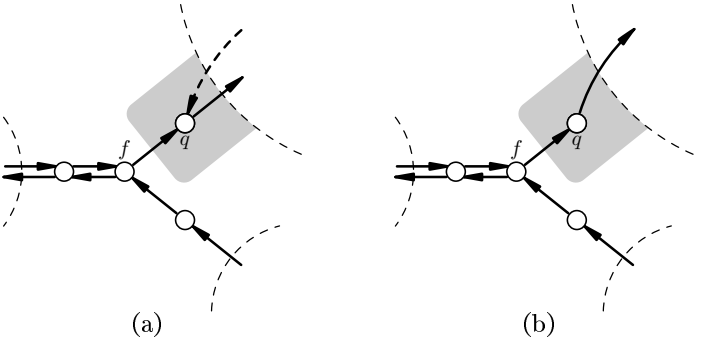


Fig. 7. Augmentation step in case (Z4-1): $\alpha = \gamma$

$\alpha \neq \gamma$, Lemma 3 applies and the current iteration completes, see Fig. 5. If $\alpha = \beta$, then by Lemma 1 the island Q_β may be extended by adding the nodes of P_1 up to the fork node f (including f), see Fig. 6. Finally, suppose $\alpha = \gamma$. Then again by Lemma 1 the island Q_γ may be extended by adding the nodes of P_1 starting from q (including q), see Fig. 7.

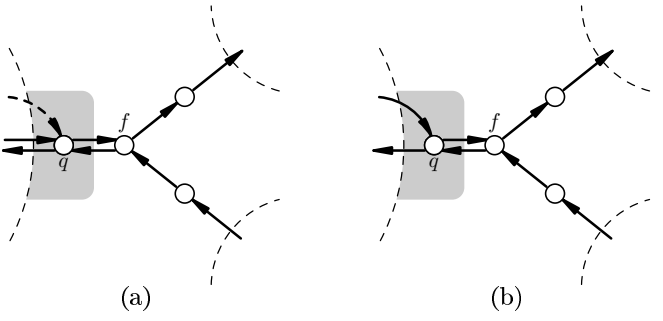


Fig. 8. Augmentation step in case (Z4-2): $\alpha = \beta$

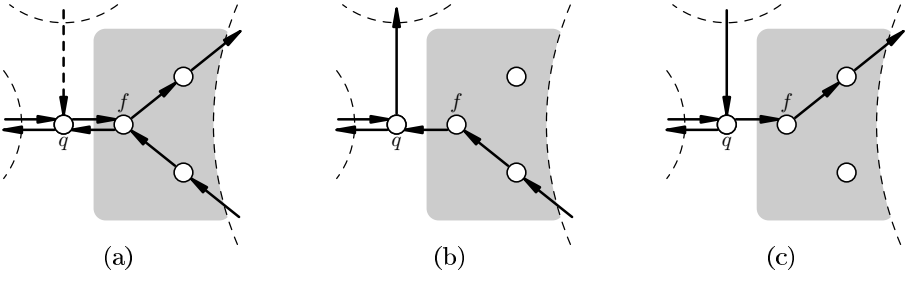


Fig. 9. Augmentation step in case (Z4-2): $q \neq f$, $\alpha \neq \beta$, $\gamma = \delta$

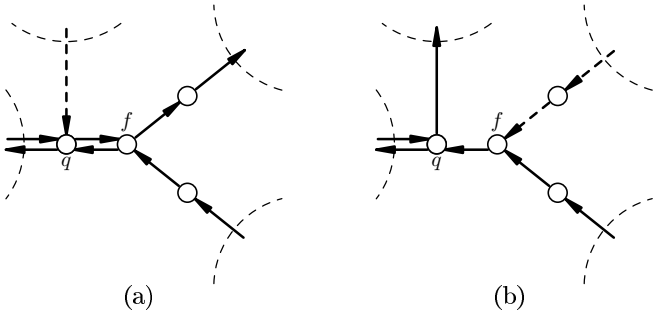


Fig. 10. Augmentation step in case (Z4-2): $q \neq f$, $\alpha \neq \beta$, $\gamma \neq \delta$

Subcase (Z4-2). Suppose $\zeta^{\mathcal{P}}(q) = 2$. Denote by P_1 and P_2 the paths in $\mathcal{P}$ that pass through q. Let P_1 go from Q_β to Q_γ and P_2 go from Q_δ to Q_β. If $\alpha = \beta$, then by Lemma 1 the island Q_β is extended by adding the nodes of P_1 up to q (including q), see Fig. 8. Otherwise $\alpha \neq \beta$. Suppose $q \neq f$. If $\gamma = \delta$, then by Lemma 1 the island $Q_\gamma = Q_\delta$ may be extended by adding the nodes of P_1 starting

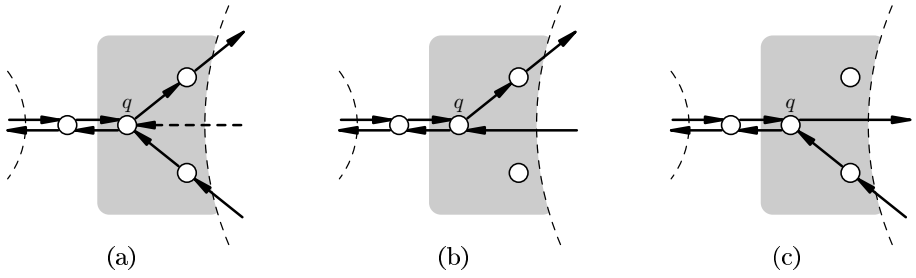


Fig. 11. Augmentation step in case (Z4-2): $q = f$, $\alpha \neq \beta$, $\gamma = \delta = \alpha$

from f (including f) and the nodes of P_2 up to f (including f), see Fig. 9. If $\gamma \neq \delta$, then let q and f split P_1 into the parts P_{11}, P_{12}, P_{13} ($P_1 = P_{11} \circ P_{12} \circ P_{13}$). Replace P_1 by $P_{11} \circ L^{-1}$. Now Lemma 3 applies with $L := P_{13}^{-1}$, so the iteration is complete, see Fig. 10. Finally, let us consider the case $q = f$. If $\gamma = \delta = \alpha$, then by Lemma 1 the island Q_α may be extended by adding the nodes of P_1 from f (including f) and the nodes of P_2 up to f (including f), see Fig. 11. Otherwise ($\gamma \neq \delta$ or $\gamma \neq \alpha$ or $\delta \neq \alpha$), the augmentation step always succeeds (possibly with the help of path switching at f). One can assume that $V_{P_1} \cap Z = V_{P_2} \cap Z = \{f\}$, since otherwise after replacing the corresponding "long" part of a path by L (possibly reversed) one may execute an island extension and, hence, complete the phase. After the augmentation step the node f is of an allowed type, namely (Z3) or (Z4), and the updated path L is of type (H1); properties (J2) and (J4) are preserved.

4 Running Time

Now let us estimate the running of the procedure described in Section 3. As earlier, put $n := |V_G|$, $m := |E_G|$. It was already shown that the algorithm performs up to $O(n)$ iterations each consisting of up to $O(n)$ phases. It remains to bound the complexity of a single phase. To store packings we represent each path by a linked list of edges (or arcs). Packing canonization procedure is carried out in $O(m)$ time. To find an augmenting sequence R (or figure out that no such sequence exists) one can traverse the residual graph in $O(m)$ time.

During each augmentation the algorithm may need to find out the islands containing the endpoints of a certain path P in the current packing. Generally this cannot be carried out in $O(1)$. But in all these cases the algorithm either completes a phase (so the straightforward tracing in $O(n)$ works) or the requested islands may be found by walking $O(1)$ distance from q along P.

Therefore, each augmentation step takes $O(1)$ time, except for the last one, which may take $O(n)$ time. Totally the running time is $O(n^2(m+n)) = O(mn^2)$, as claimed.

5 Conclusions

There are several ways the above results may be improved and generalized. Firstly, the approach seems to be directly applicable to the framework of group-labelled graphs (see [1,4]) thus providing a simple combinatorial algorithm for constructing a maximum 2-packing of non-returning T-paths.

Secondly, one may consider a generalized version of (P) where $c(v)$ are even for all $v \in V_G$. Obviously, this problem may be solved by the above algorithm in pseudo-polynomial time with the help of node-splitting transformation. The capacity-scaling approach should also work here achieving the time bound that is polynomial in n, m, and $\log U$ (where U denotes the maximum capacity). The exact complexity for this problem is still to be settled.

Finally, the case of singleton terminal classes ($|T_i| = 1$ for all $1 \leq i \leq k$) may be considered. It turns out that our approach is not efficient here. Rather, a simple augmenting procedure (which resembles Edmonds' ideas) can be used to achieve $O(mn)$ time bound. Due to the lack of space, we omit the details. One may ask if some sort of the blocking augmentation strategy may help to reduce the time bound even further.

Acknowledgements

The author would like to express his gratitude to Prof. Alexander V. Karzanov for the collaboration and help. The author is also thankful to the anonymous referees for pointing out several inaccuracies in the initial version of the paper.

References

1. Chudnovsky, M., Geelen, J., Gerards, B., Goddyn, L., Lohman, M., Seymour, P.: Packing non-zero A-paths in group-labelled graphs. Combinatorica 26(5), 521–532 (2006)
2. Lovász, L.: Matroid matching and some applications. J. Combinatorial Theory, Ser. B 28, 208–236 (1980)
3. Mader, W.: Uber die maximalzahl kantendisjunkter H-wege. Archiv der Mathematik (Basel) 31, 382–402 (1978)
4. Pap, G.: Packing non-returning A-paths algorithmically. In: 2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05), volume AE of DMTCS Proceedings, Discrete Mathematics and Theoretical Computer Science, pp. 139–144 (2005)
5. Schrijver, A.: A short proof of Mader's S-paths theorem. Journal of Combinatorial Theory, Ser. B 82, 319–321 (2001)
6. Schrijver, A.: Combinatorial Optimization. Springer, Heidelberg (2003)

Decidability of Parameterized Probabilistic Information Flow

Danièle Beauquier[1], Marie Duflot[1], and Yury Lifshits[2,⋆]

[1] University Paris 12, France
[2] Steklov Institute of Mathematics, St.Petersburg, Russia

Abstract. In this paper, we consider the decidability of two problems related to information flow in a system with respect to some property. A flow occurs in a system if the conditional probability of the property under some partial observation differs from the a priori probability of that property. For systems modelled as finite Markov chains we prove that the two following problems are decidable: does a system has information flow for a given regular property? is it true that the system has no information flow for any (sequential) property?

1 Introduction

In this paper we study the following *Security of Information Flow* problem: verify that no partial observation of a system behavior does leak an information that should be hidden.

Statement of the problem and our results. We use the framework of [14] and its formalization from [3]. In our trace-based approach, we assume a set of observable low-level events L and a set of (not directly observable) high-level events H. The question is whether observing a certain low-level trace can give information about the occurrence of high-level events in a probabilistic sense, yielding quantitative information about high-level activity. More precisely we propose a *parameterized* view of information flow. We define information flow with respect to a *property* (a set of system traces) which is deemed important for the system under scrutiny. This property is our *parameter* of the problem. The system has information flow with respect to the given property if there exist two low-level observations for which the chosen property has different probabilities of occurrence. In this case, the quantitative, probabilistic knowledge about the given property is sensitive to the observation which can be made, and so there is information flow in the system with respect to this property. It is worth mentioning that this probabilistic definition of information flow is related to Shannon's original definition of information, based on probabilities. In our previous paper [3] the formalization of information flow was presented for the first time together with necessary and sufficient conditions for having no information flow *for all*

⋆ Partially supported by the grants Russian Foundation for Basic Research N 06-01-00584-a INTAS YSF N 1000014-6233.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 82–91, 2007.

properties in a given system. Here, in order to get decidability results we restrict ourselves to systems modelled by finite Markov chains (with labelled edges) and to regular properties. It has a clear practical motivation. A field of application can be for example verification of security for parallel programming. Interleaving of actions of different threads is generally managed in a probabilistic way, and can be modelled as a Markov chain. As for security properties, many of them are regular.

Our first result states that it is decidable in polytime whether a system has information flow for a given property. The key ingredient of the algorithm is a trick from linear algebra, reformulating the notion of information flow as orthogonality of the set of vectors corresponding to all possible observations to some checking vector.

Our second result states that it is decidable in exponential time whether a given system has no information flow *for all properties*. We consider two subcases: no information flow for any property and no information flow for sequential properties (those that do not consider the explicit value of low level actions).

Plan of the paper. Section 2 introduces the model under study and some related notations and definitions. In sections 3, 4 we present our decidability results just mentioned above together with an example of application in the domain of concurrent programs that illustrates how the interleaving of low-level actions can give probabilistic information on what happens at the high-level. Section 5 concludes.

Related work. There is an important body of work in studying definitions related to information flow, for an overview see, e.g., [10]. We restrict the comparaison to the two features of our formalization: our notion of information flow is *parameterized* and it has *probabilistic* nature.

As far as we are aware of, only the paper of J. Halpern and K. O'Neill [7] parameterizes information flow by giving a definition of secrecy in multi-agent systems. They use a modal logic of knowledge in a state-based model as compared to our approach which is trace-based. Their framework generalizes several existing approaches and can be extended to probabilistic security. Their parametrization stems from defining formulas (knowledge) of what must be kept secret.

The other probabilistic approaches are more restrictive. McLean [10] introduces the *flow model* which distinguishes mere correlation from actual causal influence. Gray [5] introduces probabilistic *interference* in a context of finite state machines and gives a more general information-theoretic framework (as compared to [10]) including probabilistic channel capacity [6]. Sabelfeld and Sands [13] define probabilistic *noninterference* in the context of schedulers for multithreaded programs, based on the concept of probabilistic bisimulation. Lowe [9] treats quantitative information flow distinguishing probabilistic aspects from *nondeterminism*. A probabilistic process-algebraic approach is given in [1], focused on *noninterference*, generalizing the possibilistic variant and allowing formal reasoning about the amount of information flow. All these works are aimed at the definition of the models and do not deal with algorithmic problems.

Very few authors studied verification problems related to information flow. Among probabilistic approaches we can cite [4] that uses a process algebra formalism to study bisimulation-based security properties. Concerning probabilistic models, [12] gives a decidability result for "nondeducibility on composition" for probabilistic timed automata, and Gray [5] gives a sufficient condition for information flow security which *seems* decidable.

2 Probabilistic Event Systems

Notations. Given a finite alphabet A, A^* (resp. A^ω) denotes the set of finite (resp. infinite) sequences (or traces) over this alphabet. The set A^∞ is the union of A^* and A^ω. The empty sequence is denoted ε.
Given a sub-alphabet $A' \subset A$ and a trace λ, $\lambda_{|A'}$ denotes the projection of λ onto this sub-alphabet.

Let $u, v \in (A^*)^n$, $u = (x_1, x_2, \ldots, x_n), v = (y_1, y_2, \ldots, y_n)$. We denote by $u \otimes v$ the *simple interleaving* of u and v defined as $u \otimes v = x_1 y_1 x_2 y_2 \ldots x_n y_n$. If $U, V \subset (A^*)^n$, we denote by $U \otimes V$ the set: $U \otimes V = \{u \otimes v | u \in U, v \in V\}$. If $U, V \subset (A^*)^\omega$, the definition of $U \otimes V$ is extended in a standard way.

Probabilistic Event Systems. The behaviour of a probabilistic event system is modelled by its set Tr of traces which are finite or infinite sequences of atomic events from a set E. A particular atomic event τ is distinguished which represents the halting of the system. For example, if λ is a sequence of atomic events, it is useful to distinguish between "λ has occurred but the system is still in action", and "λ has occurred and the system stopped". The last case is modelled by the event $\lambda\tau$. In order to unify the presentation it is convenient to use only infinite sequences and thus we use $\lambda\tau^\omega$ instead of $\lambda\tau$. Then, from now on, Tr is a set of infinite sequences which either do not contain any occurrence of τ or of the form $\lambda\tau^\omega$ where λ does not contain any occurrence of τ.

In order to deal with information flow issues, the set of atomic events E is divided into two disjoint sets, the set H of high-level (*i.e.* secret) atomic events and the set L of low-level (*i.e.* public) ones.

The set of traces Tr is equipped with a probability measure μ over the σ-algebra generated by the cylinders αE^ω, such that Tr is μ-measurable. The measure $\mu(X)$ of a measurable set X is denoted as $Pr_\mu(X)$, or shortly $Pr(X)$. Thus if we consider the infinite tree $\mathcal{T}_S$ built from Tr with edges labelled by atomic events, each edge of the tree is equipped with a non-zero probability. (We assume that every prefix of a trace in Tr has a non-zero probability).

As usual, we introduce the following notation for conditional probabilities: if P and Q are two measurable events and $Pr(Q) \neq 0$, the conditional probability $Pr(P|Q)$ is equal to $Pr(P \cap Q)/Pr(Q)$. Since we are interested only in traces of the system $\mathcal{S}$ we will deal only with the conditional probabilities relative to Tr. Thus, for each measurable event X we denote by $Pr_\mathcal{S}(X)$ the probability $Pr(X|\mathcal{S})$ ($Pr(\mathcal{S})$ is supposed to be positive).

Definition 1. *A* probabilistic event system *is a tuple* (E, H, L, Tr, μ) *where* $E = H \cup L$, $H \cap L = \emptyset$ *and* H *(resp.* L*) is the set of high-level (resp. low-level) actions,* μ *is a probabilistic measure on* E^ω *and* $Tr \subset E^\omega$*, the set of traces of the system, is* μ*-measurable.*

We assume that only low-level actions are observable on the low-level, i.e., for a trace λ the projection $\lambda_{|L}$ is observable by low-level users. More precisely, every finite prefix of $\lambda_{|L}$ is observable. Thus, from the observation of $u \in L^*$, the low-level user who is supposed to know the entire system can construct the *bunch* $B_S(u) = \{\lambda \in Tr \mid u \text{ is a prefix of } \lambda_{|L}\}$ and possibly deduce some information on what happened or what will happen. When there is no ambiguity, we will write $B(u)$ instead of $B_S(u)$.

A *property* is a subset of E^ω. From now on we consider only μ-measurable properties.

Definition 2. *A system* S *is* without information flow *for a property* P *if for every* $u, v \in L^*$ *such that* $B(u)$ *and* $B(v)$ *are non-empty,* $Pr_S(P|B(u)) = Pr_S(P|B(v))$.

The above definition means that, whatever the low level user observes, he does not get additional information on the probability of P to hold.

A particular case of interest is when only the presence of low level events is important for P, not their value. Such a property is called *sequential* and defined below.

Definition 3. *A property* P *is* sequential *if there exists* $P' \subseteq (H \cup \{l\})^\omega$ *such that* $P = \phi^{-1}(P')$ *where* ϕ *is a morphism which is identity on* H *and for each* $l_i \in L, \phi(l_i) = l$.

3 Decidability of Information Flow for a Given Property

We will now state some conditions under which one can decide whether a probabilistic event system has information flow under some property.

The most common probabilistic systems described in a finite way are Markov chains, and the simplest properties are regular ones, i.e. recognized by a deterministic Muller automaton. We recall below the definition of Markov chains [8] (with a small change) and Muller automata [11].

Definition 4. *We call* Markov chain with labelled edges *a system* $\mathcal{A} = (\Sigma, i, A, T)$ *where* S *is a finite set of states,* $i \in S$ *is the initial state,* A *is a finite alphabet,* $T : S \times A \times S \mapsto [0, 1]$ *is a function such that* $\forall s \in S, \sum_{s' \in S, a \in A} T(s, a, s') = 1$ *and for each* $(s, a) \in S \times A$ *there is at most one* s' *such that* $T(s, a, s') > 0$.[1]

This system is slightly different from a classical Markov chain for which $T : S \times S \mapsto [0, 1]$. Here there can be more than one edge between two states (if they

[1] This last condition means that the underlying automaton (without the probabilities) is deterministic.

have different labels). In order to get decidability results we suppose that T has its values in the set $\mathbb{Q}$ of rational numbers.

Let $\mathcal{P}_q$ be the set of paths from state q. The set $\mathcal{P}_i$ of infinite paths from the initial state i is equipped with a probabilistic measure μ in a standard way. A trace is the infinite sequence of labels of an infinite path.

Let Tr be the set of traces. The probability measure μ' on Tr is defined as follows: for every basic cylinder uA^ω, $\mu'(uA^\omega)$ is the mesure $\mu(w\mathcal{P}_q)$ where w is the path from i labelled with u and q is the last state of w.

Thus if A is partitioned into two sets of high-level and low-level actions, H and L, the Markov chain defines a probabilistic event system (A, H, L, Tr, μ').

Definition 5. *A* Muller automaton *is a tuple $\mathcal{M} = (Q, A, q_0, \Delta, \mathcal{F})$, where Q is the finite set of states, q_0 is the initial state, Δ is the set of transitions and $\mathcal{F}$ is the set of accepting subsets. An infinite word w is accepted by the automaton if the set of infinitely visited states along some*[2] *path with label w belongs to $\mathcal{F}$.*

It is a well known result that deterministic (complete) Muller automata have the same expressive power as nondeterministic ones [11].

Now that we have defined the type of systems we consider, we can state the main result.

Theorem 1. *Given a system $\mathcal{S}$ described by a Markov chain with labelled edges, and a regular property on infinite traces P given as a deterministic Muller automaton, we can decide in polytime whether the system $\mathcal{S}$ has information flow for the property P.*

Proof. The full proof cannot fit in the page limitation but is available in the full paper [2]. We can however give here a sketch.

To prove the result, we state the problem in terms of matrices. We want the probability $Pr_S(P|B(u))$ to be a constant (not dependant on u). In other words we want to exhibit some constant c such that $\frac{Pr(P \cap B(u))}{Pr(B(u))} = c$ for every u, which is equivalent to: $Pr(P \cap B(u)) = c \times Pr(B(u))$.

1. first we compute a composition of the Markov chain and Muller automaton of the property, giving a new (labelled) Markov chain
2. from this chain we compute, for each low level event a "one step matrix" M_l giving the transition probabilities for H^*l,
3. next we show that the information flow problem is equivalent to the orthogonality of a linear hull (generated by the initial vector and iterated multiplication by the one step matrices) and a given "check vector" related to the probability, from each state of the product markov chain, to satisfy the property P,
4. and to conclude we prove that the hull mentioned above is computable. □

Example 1. Let us consider the problem of information flow for concurrent programs. The question is whether observing some values for its low variables we can conclude anything about high ones.

[2] Such a path is unique in the case of a deterministic and complete automaton.

Consider the following multi-threaded program O inspired by [15]:

- Thread α:
  $h_0 := h_1$;
  $l_0 := 1$;
- Thread β:
  $h_0 := h_2$;
  $l_1 := 1$;

The low variables are l_0, l_1 initially equal to zero, h_0, h_1, h_2 are high variables. The content of h_1 and h_2 are different. Suppose that the two threads are scheduled probabilistically, with equal probabilities at each step for each thread to be run. The corresponding Markov chain is given in figure 2.

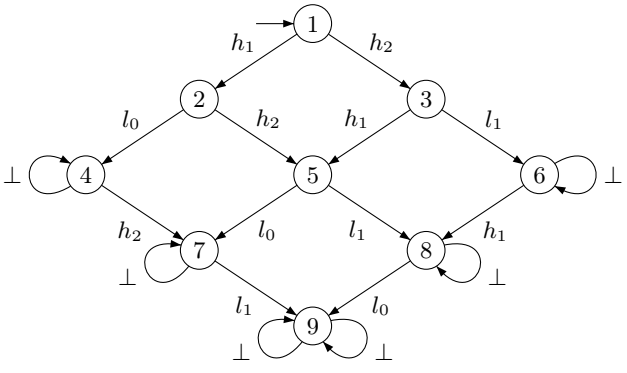


Fig. 1. The Markov chain associated to the program (each edge has a probability 1/2)

Each state contains the current state of threads (*i.e.* the set of instructions still to execute). For $i = 1, 2$, the label h_i means that the instruction $h_1 := h_i$ is executed, and the label l_i corresponds to the execution of $l_i := 1$. The actions of thread α (resp. β) correspond to the left (resp. right) edges.

For example state 5 corresponds to ($\alpha :\ l_0 := 1;\ \beta :\ l_1 := 1;$) and state 6 corresponds to ($\alpha : h_0 := h_1;\ l_0 := 1;$).

Suppose we are interested in the value of h_0 at the end of the program, more precisely in the property $P :\ h_0 = h_1$ after the execution of O. We can represent P as the language $(L \cup H \cup \bot)^* h_1 (L \cup \bot)^\omega$. Indeed this regular expression says that the last update of h_0 is $h_0 := h_1$. Notice that this property is sequential.

Using our method, from the Markov chain given previously and the two-state deterministic Muller automaton corresponding to property P, we get :

1. a composed Markov chain with 13 states,
2. two "one step matrices" M_{l_0} and M_{l_1},
3. we generate the hull reachable from the initial state using the above one step matrices (the dimension of the hull is 5 in this case),
4. then we see that this linear hull is not orthogonal to the check vector.

We can then conclude that the program has information flow for property P. In this particular case it means that, seeing the order in which low level variables are assigned, the low level user can gain (probabilistic) information on the order in which high level variables are assigned.

4 Decidability of General Information Flow

Definition 6. *A system is* without (sequential) information flow *if it is without information flow for every (sequential) property.*

In [3] such systems were characterized, *i.e.* necessary and sufficient conditions were given to ensure the absence of such information flow. We will show in this section that it is possible to decide whether a system is without (sequential) information flow when the considered system is a Markov chain with labelled edges.
The plan of this section is as follows:

- we first recall some definitions and notations necessary to state the criteria,
- then we recall the theorems from [3],
- and we conclude by proving that all these criteria are decidable for the models considered

In the following, low level actions are denoted $a, b, ...$, sequences of low-level actions $u, v, ...$, sequences of high-level actions $\alpha, \beta, ...$ and traces $\lambda, \lambda', ...$

Let $\mathcal{S} = (E, H, L, Tr, \mu)$ be a system, $\mathcal{T}$ be the associated probabilistic tree and $Pref(Tr)$ denote the set of finite prefixes of traces of Tr We define:

$H_n(Tr) = \{(\alpha_1, ..., \alpha_n) \in (H^*)^n \mid \exists\ a_1, ..., a_n \in L\ \alpha_1 a_1 ... \alpha_n a_n \in Pref(Tr)\}.$
$L_n(Tr) = \{(a_1, ..., a_n) \in L^n \mid \exists\ \alpha_1, \ldots, \alpha_n \in H^*\ \alpha_1 a_1 ... \alpha_n a_n \in Pref(Tr)\}$
$Tr_n = \{\alpha_1 a_1 \ldots \alpha_n a_n \in Pref(Tr) \mid \alpha_i \in H^*,\ a_i \in L\}.$

For the decidability proof given below, we need to introduce some technical terms related to the probabilistic tree $\mathcal{T}$.

We are interested in the set of sequences of high-level actions (including the empty word) which can occur starting from a node x. To make this set of sequences more explicit we build for each such node x a probabilistic tree $\mathcal{T}_x$ in the following way: we keep only the high edges reachable in $\mathcal{T}$ from x, and for each node y (including x) accessible from x by a high path with at least one low edge starting from y, we add a node y' and an edge (y, y') labelled by ε and with a probability equal to the sum p of the probabilities of low edges starting from y in $\mathcal{T}$. The tree $\mathcal{T}_x$ is a probabilistic tree which has the following meaning: the probability of a path in $\mathcal{T}_x$ starting from x labelled by α (without ε labels) is exactly the probability that the sequence of high-level actions α occurs from x; the probability of a path in $\mathcal{T}_x$ starting from x labelled by α and ending in a leaf is the probability that from x the sequence of actions α followed by a low-level action occurs.

A tuple (x, x', y, y') of nodes of the tree $\mathcal{T}$ is *H, L-compatible* if there exist $(\alpha_1, ..., \alpha_n), (\beta_1, ..., \beta_n) \in H_n(Tr)$, and $(a_1, ..., a_n), (b_1, ..., b_n) \in L_n(Tr)$ such that the paths from the root to x, x', y, y' are labelled respectively by $\alpha_1 a_1 ... \alpha_n a_n$, $\alpha_1 b_1 ... \alpha_n b_n$, $\beta_1 a_1 ... \beta_n a_n$ and $\beta_1 b_1 ... \beta_n b_n$.

Let $p_1, ..., p_n$, $q_1, ..., q_n$ be the probabilities of edges labelled by $a_1, ..., a_n$ on the path from the root to x (resp. y). Let $p'_1, ..., p'_n$, $q'_1, ..., q'_n$ be the probabilities of edges labelled by $b_1, ..., b_n$ on the path from the root to x' (resp. y').

An H, L-compatible tuple (x, x', y, y') is *perfect* if for every $i = 1, ..., n$ we have $p_i/q_i = p'_i/q'_i$.

Let us now rephrase theorems from [3].

Theorem 2. *A probabilistic system such that $Tr \not\subset H^\omega$ is*

1. *without information flow iff its projection on L is reduced to a single trace.*
2. *without sequential information flow iff*
   (1) $\forall n > 0\ Tr_n = H_n(Tr) \otimes L_n(Tr)$.
   (2a) Every H, L-compatible tuple (x, x', y, y') of the tree $\mathcal{T}$ is perfect and
   (2b) the probabilistic trees $\mathcal{T}_x$ (resp.$\mathcal{T}_y$) and $\mathcal{T}_{x'}$ (resp.$\mathcal{T}_{y'}$) are isomorphic.
   (3) For every $n > 0$ $(L_n(Tr) \neq \emptyset \rightarrow\ Pr_S(Tr \cap (H^*L)^{n-1}H^\omega) = 0)$.

We can now state our decidability result:

Theorem 3. *Given a system $\mathcal{S}$ described by a Markov chain $\mathcal{A}$ with labelled edges, one can decide in exponential time whether $\mathcal{S}$ has (sequential) information flow.*

Proof. The proof of this theorem is given in the full paper [2]. Here are the main ideas.

First for non sequential information flow the criterion is clearly decidable.

For point (1) of sequential information flow we compute the deterministic finite automata recognizing respectively $\cup_{n\in\mathbb{N}} Tr_n$ and $\cup_{n\in\mathbb{N}}(H_n(Tr) \otimes L_n(Tr))$. Checking the equality of the two languages is decidable.

For point (2) we compute an automaton whose states are the quadruples of H, L-compatible states (states corresponding to some H, L-compatible nodes) and verify on the fly that the ratio on probabilities are preserved, and that the corresponding probabilistic trees are isomorphic.

For point (3) we compute ergodic sets containing only high-level actions and, if one exists, reason on the number of low-level actions necessary to reach this ergodic set. □

Here is an example of system without sequential information flow. In particular the ratio of probabilities of events l_1 and l_2 from state 2 to state 4 and from state 3 to state 5 is the same.

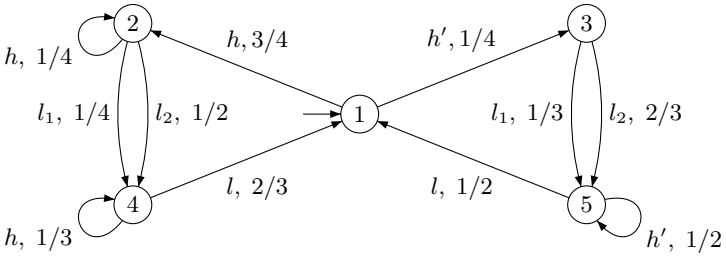


Fig. 2. A Markov chain without sequential information flow

5 Conclusion and Further Work

This paper presents two decidability results for information flow when the system is a Markov chain with labelled edges and properties are regular. First we show

that it is decidable whether a system has information flow for a specific regular property. Then we consider the decidability of absence of information flow for a class of properties and prove that the criteria given in [3] to ensure that the system has no information flow for two classes of properties, are decidable.

As it was noticed in the introduction, very few papers are devoted to algorithmic questions related to information flow. To our knowledge, our results are the first decidability ones for a probabilistic and parameterized model. This opens a way to quantitative evaluation of information flow.

Interesting open questions include: (1) computing quantitative estimations of information flow, (2) generalizing our algorithms to more expressive formalisms of system/property descriptions and (3) implementing and experimental testing of our algorithms in some applied domain.

Acknowledgements. We are grateful to Anatol Slissenko for the numerous and fruitful discussions about this paper.

References

1. Aldini, A., Bravetti, M., Gorrieri, R.: A process-algebraic approach for the analysis of probabilistic noninterference. Journal of Computer Security 12, 191–246 (2004)
2. Duflot, M., Beauquier, D., Lifshits, Y.: Decidability of parameterized probabilistic information flow. Technical report, LACL - Univ. Paris 12 (2007), http://www.univ-paris12.fr/lacl/duflot/publis.html
3. Duflot, M., Beauquier, D., Minea, M.: A probabilistic property-specific approach to information flow. In: Gorodetsky, V., Kotenko, I., Skormin, V.A. (eds.) MMM-ACNS 2005. LNCS, vol. 3685, pp. 206–220. Springer, Heidelberg (2005)
4. Focardi, R., Gorrieri, R.: The compositional security checker: A tool for the verification of information flow security properties. Software Engineering 23(9), 550–571 (1997)
5. Gray III, J.W.: Probabilistic interference. In: Proc. IEEE Symp. on Security and Privacy, pp. 170–179 (May 1990)
6. Gray III, J.W.: Toward a mathematical foundation for information flow security. In: Proc. 1991 IEEE Symp. on Security and Privacy, pp. 21–35. IEEE Comp. Soc. Press, Los Alamitos (1991)
7. Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. In: Proc. IEEE Computer Security Foundations Workshop, p. 32. IEEE Computer Society Press, Los Alamitos (2002)
8. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. D Van Nostad Co. Inc., Princeton, N.J. (1960)
9. Lowe, G.: Quantifying information flow. In: Proc. IEEE Computer Security Foundations Workshop, pp. 18–31. IEEE Computer Society Press, Los Alamitos (2002)
10. McLean, J.: Security models and information flow. In: Proc. IEEE Symp. on Security and Privacy, pp. 180–187. IEEE Computer Society Press, Los Alamitos (1990)
11. Perrin, D., Pin, J.E.: Infinite words. In: Pure and Applied Mathematics, vol. 141. Elsevier, Amsterdam (2004)
12. Maggiolo-Schettini, A., Lanotte, R., Troina, A.: Information flow analysis for probabilistic timed automata. In: Workshop on Formal Aspects in Security and Trust (FAST). IFIP International Federation for Information Processing, vol. 12, pp. 1–15 (2004)

13. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: Proc. IEEE Computer Security Foundations Workshop, pp. 200–214. IEEE Computer Society Press, Los Alamitos (2000)
14. Slissenko, A.: Probability and time in measuring security. In: Ţiplea, F.L., Clarke, E., Minea, M., Ţiplea, F.L. (eds.) Proc. of NATO Adv. Research Work.: Verification of Infinite-State Systems with Applications to Security (VISSAS'05). NATO Security through Science Series. D: Information and Communication Security, vol. 1, pp. 169–183. IOS Press, Amsterdam (2006)
15. Volpano, D., Smith, G.: Probabilistic noninterference in a concurrent language. Journal of Computer Security 7, 231–253 (1999)

Inverting Onto Functions and Polynomial Hierarchy

Harry Buhrman[1], Lance Fortnow[2], Michal Koucký[3], John D. Rogers[4], and Nikolay Vereshchagin[5]

[1] CWI, Amsterdam
buhrman@cwi.nl
[2] University of Chicago
fortnow@cs.uchicago.edu
[3] Institute of Mathematics of Czech Academy of Sciences
mkoucky@cs.mcgill.ca
[4] DePaul University, Chicago
jrogers@cs.depaul.edu
[5] Lomonosov Moscow State University
ver@mech.math.msu.su

Abstract. The class **TFNP**, defined by Megiddo and Papadimitriou, consists of multivalued functions with values that are polynomially verifiable and guaranteed to exist. Do we have evidence that such functions are hard, for example, if **TFNP** is computable in polynomial-time does this imply the polynomial-time hierarchy collapses? (By computing a multivalued function in deterministic polynomial-time we mean on every input producing one of the possible values of that function on that input.)

We give a relativized negative answer to this question by exhibiting an oracle under which **TFNP** functions are easy to compute but the polynomial-time hierarchy is infinite. We also show that relative to this same oracle, $\mathbf{P} \neq \mathbf{UP}$ and $\mathbf{TFNP}^{\mathbf{NP}}$ functions are not computable in polynomial-time with an **NP** oracle.

1 Introduction

Megiddo and Papadimtriou [MP91] defined the class **TFNP**, the class of multivalued functions with values that are polynomially verifiable and guaranteed to exist. A function from **TFNP** is specified by a polynomial time computable binary relation $R(x, y)$ and a polynomial p such that for every string x there is a string y of length at most $p(|x|)$ such that $R(x, y)$ holds. It maps x to any y of length at most $p(|x|)$ such that $R(x, y)$. This class of functions includes Factoring, Nash Equilibrium, finding solutions of Sperner's Lemma, and finding collisions of hash functions.

Fenner, Fortnow, Naik and Rogers [FFNR03] consider the hypothesis, which they called "Q", that for every function in **TFNP** there is a polynomial-time procedure that will output a value of that function. That is, Proposition Q states

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 92–103, 2007.

that for every R and p defining a **TFNP**-function there is a polynomial time computable function f such that $R(x, f(x))$ holds for all x.

Fenner et. al. showed that Q is equivalent to a number of different hypotheses including

- Given an **NP** machine M with $L(M) = \Sigma^*$, there is a polynomial-time computable function f such that $f(x)$ is an accepting computation of $M(x)$.
- Given an honest onto polynomial-time computable function g there is a polynomial-time computable function f such that $g(f(x)) = x$. (A function $g(x)$ is called honest if there is a polynomial $p(n)$ such that $|x| \leq p(|g(x)|)$ for all x.)
- For all polynomial-time computable subsets S of **SAT** there is a polynomial-time computable function f such that for all ϕ in S, $f(\phi)$ is a satisfying assignment to ϕ.
- For all **NP** machines M such that $L(M) = \mathbf{SAT}$, there is a polynomial-time computable function f such that for every ϕ in **SAT** and accepting path p of $M(\phi)$, $f(\phi, p)$ is a satisfying assignment of ϕ.

Proposition Q implies that all disjoint **coNP**-sets are **P**-separable (which implies that $\mathbf{NP} \cap \mathbf{coNP} = \mathbf{P}$) and is implied by $\mathbf{P} = \mathbf{NP}$. Fenner et. al. ask whether we can draw any stronger complexity collapses from Q, in particular if Q implies that the polynomial-time hierarchy collapses. We give a relativized negative answer to this question by exhibiting an oracle relative to which Q holds and the polynomial-time hierarchy is infinite. Our proof uses a new "Kolmogorov generic" oracle.

The proposition Q naturally generalizes to other levels of the polynomial hierarchy. Namely, define the class of $\mathbf{TF}\Sigma_k^p$ as follows. A $\mathbf{TF}\Sigma_k^p$-function is specified by a binary relation $R(x, y)$ computable in polynomial time with an oracle from Σ_{k-1}^p and a polynomial p such that for every string x there is a string y of length at most $p(|x|)$ such that $R(x, y)$ holds. For $k \geq 1$ we will label Σ_k^pQ the statement

> for every R and p defining a $\mathbf{TF}\Sigma_k^p$-function there is a function f computable in polynomial time with an oracle from Σ_{k-1}^p such that $R(x, f(x))$ holds for all x.

For $k = 1$ we obtain the class **TFNP** and the Proposition Q.

Proposition $\mathbf{TF}\Sigma_k^p$ implies that $\mathbf{P}^{\Sigma_{k-1}^p} = \Sigma_k^p \cap \Pi_k^p$ and is implied by $\Sigma_{k-1}^p = \Sigma_k^p$. A natural question is whether, similar to the implication

$$\Sigma_{k-1}^p = \Sigma_k^p \Longrightarrow \Sigma_k^p = \Sigma_{k+1}^p,$$

Proposition $\mathbf{TF}\Sigma_k^p$ implies Proposition $\mathbf{TF}\Sigma_{k+1}^p$. We give a relativized negative answer to this question in the case $k = 1$: for any Kolmogorov generic G the Proposition $\mathbf{TF}\Sigma_2^{p,G}$ does not hold. Besides, we show that for any Kolmogorov generic G, $\mathbf{P}^G \neq \mathbf{UP}^G$.

2 Definitions and Preliminaries

Let Σ denote the alphabet $\{0,1\}$. The set of all finite-length binary strings is denoted Σ^*.

2.1 Complexity Classes

Our model of computation is the oracle Turing machine, both deterministic (DTM) and nondeterministic (NTM). Unless otherwise noted, all machines in this paper run in polynomial time. We assume that the reader is familiar with the complexity classes **P**, **NP**, **UP**, **PSPACE**, Σ_k^p, and Π_k^p for $k \geq 0$. The class Δ_k^p is defined as $\mathbf{P}^{\Sigma_{k-1}^p}$, and $\mathbf{PH} = \bigcup_k \Sigma_k^p$ stands for the Polynomial hierarchy. The class $\mathbf{F}\Delta_k^p$ is defined as the class of all functions from Σ^* to Σ^* that are computable in polynomial time with an oracle from Σ_{k-1}^p.

We say that disjoint sets B, C are **P**-separable if there is a set $D \in \mathbf{P}$ such that $B \subset D$ and $C \subset \Sigma^* - D$.

Proposition Q and its generalizations Σ_k^pQ are defined in the Introduction. It is easy to see that Σ_k^pQ is equivalent to the following statement:

> For every nondeterministic polynomial-time Turing machine M with oracle from Σ_{k-1}^p that accepts Σ^*, there is a function f in $\mathbf{F}\Delta_k^p$ such that, for all x, $f(x)$ is an accepting computation of $M(x)$.

It is easy to see the following:

Proposition 1. *If $\Sigma_{k-1}^p = \Sigma_k^p$ then $\Sigma_k^p Q$ is true. If $\Sigma_k^p Q$ is true then $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$.*

2.2 Kolmogorov Complexity and Randomness

An excellent introduction to Kolmogorov complexity can be found in the textbook by Li and Vitányi [LV97]. We will state here the definitions and results relevant to our work. Roughly speaking, the Kolmogorov complexity of a binary string x is defined as the minimal length of a program that generates x; the conditional complexity $C(x|y)$ of x conditional to y is the minimal length of a program that produces x with y as input.

A *conditional description method* is a partial computable function Φ (that is, a Turing machine) mapping pairs of binary strings to binary strings. A string p is called a *description of x conditional to y* with respect to Φ if $\Phi(p, y) = x$. The complexity of x conditional to y with respect to Φ is defined as the minimal length of a description of x conditional to y with respect to Φ:

$$C_\Phi(x|y) = \min\{|p| \mid \Phi(p, y) = x\}.$$

A conditional description method Ψ is called *universal* if for all other conditional description methods Φ there is a constant k such that

$$C_\Psi(x|y) \leq C_\Phi(x|y) + k$$

for all x, y. The Solomonoff–Kolmogorov theorem [Sol64, Kol65] states that universal methods exist. We fix a universal Ψ and define *conditional Kolmogorov complexity* $C(x|y)$ as $C_\Psi(x|y)$. We call this Ψ the reference universal Turing machine. The (unconditional) Kolmogorov complexity $C(x)$ is defined as the Kolmogorov complexity of x conditional to the empty string. Comparing the universal function Ψ with the function $\Phi(p, y) = \Psi(p, \text{empty string})$ we see that the conditional Kolmogorov complexity does not exceed the unconditional one:

$$C(x|y) \ \leq \ C(x) + O(1).$$

Comparing the universal function Ψ with the function $\Phi(p, y) = p$ we see that the Kolmogorov complexity does not exceed the length:

$$C(x) \leq |x| + k \tag{1}$$

for some k and all x. For most strings this inequality is close to an equality: the number of strings x of length n with

$$C(x) < n - m$$

is less than 2^{n-m}. Indeed, the total number of descriptions of length less than $n - m$ is equal to

$$1 + 2 + \cdots + 2^{n-m-1} = 2^{n-m} - 1.$$

In particular, for every n there is a string x of length n and complexity at least n. Such strings are called *incompressible, or random*.

Let $f(x, y)$ be a computable function mapping strings to strings. To describe the string $f(x, y)$ it is enough to concatenate x and y. Thus we obtain:

$$C(f(x, y)) \ \leq \ 2|x| + |y| + k. \tag{2}$$

where k depends on f and on the reference universal machine but not on x, y. The factor of 2 is needed, as we have to separate x from y. To this end we write the former in a self-delimiting form. As a self-delimiting encoding of a string u we take the string $\bar{u}$ obtained from u by doubling all its bits and appending the pattern 01. For instance, $\overline{001} = 00001101$. A similar inequality holds for computable functions of more than 2 strings:

$$C(f(x_1, x_2, \ldots, x_n)) \ \leq \ |x_1| + 2|x_2| + \cdots + 2|x_n| + O(1). \tag{3}$$

2.3 Generic Oracles

An oracle is a subset of Σ^*. The oracles we use are *generic* oracles. What does this mean? To explain this we need to recall some definitions from category theory.

A *condition* on an oracle A is a finite set of *requirements* having the form $x \in A$ and $y \notin A$, where $x, y \in \Sigma^*$. We say that an oracle A *satisfies* a condition α if all the requirements in α are satisfied by A. Let U be a family of oracles and let U_α denote the set of all $A \in U$ satisfying α. An *interval* in U is a non-empty

subset of U having the form U_α. A subset S of U is called *dense in* U if every non-empty interval I in U has a sub-interval J in U included in S. Countable intersections of dense sets are called *large* subsets of U.

Let P be a property of oracles, that is, a set of oracles. We say that P *holds for a generic oracle* if the set P is large in the set of all oracles. We say that P *holds for a generic oracle in* U if the set $P \cap U$ is large in U. Assume that U has the following property: the intersection of every infinite descending chain

$$I_1 \supset I_2 \supset I_3 \supset \dots$$

of intervals in U is non-empty. Then by the usual diagonalization we can show that every large subset of U is non-empty. Using a metaphor, we can say that "generic oracles exist." Our usage of the term "generic" oracle is similar to the usage of the term "random" oracle. Indeed, we say that a property P holds for a random oracle if P is a measure 1 set.

Note that if a property P_0 holds for a generic oracle in U and P_1 holds for a generic oracle in U then so does $P_0 \cap P_1$. Therefore if we want to prove that Proposition Q holds but that the polynomial hierarchy is infinite relative to a generic oracle in U we can prove these things separately. The same applies to countable families of properties. If each P_i holds for a generic oracle in U then the property $\bigcap_i P_i$ also holds for a generic oracle in U. For example, if we want to prove that $\mathbf{P} \neq \mathbf{NP}$ relative to a generic oracle in U we can define a relativized language L that is in $\mathbf{NP}$ for generic oracle in U and then define a set of requirements R_i, where R_i is the statement "DTM number i does not accept L." Then it is enough to prove, for every i, that R_i holds relative to a generic oracle in U. To this end it suffices to prove that the set $U \cap R_i$ is dense in U: every interval I in U has a subinterval J in U such that $J \subset R_i$.

Good introductions and several applications of the approach we are using here may be found in the papers [BGS75, FFKL03, FR03, MV96].

Previous Results

The method we are using was essentially designed in [BGS75]. Let U be a family of oracles. Fix a **PSPACE**-complete set H and consider oracles of the form

$$A \oplus H = \{0x \mid x \in A\} \cup \{1x \mid x \in H\},$$

where $A \in U$. We will denote the set of all such oracles by $U \oplus$PSPACE-complete. Consider "tower" numbers, that is, natural numbers $1, 2, 2^2, 2^{2^2}, \dots$. The next tower number from n is 2^n. Let

$$U_0 = \{A \mid A \cap \Sigma^n = \emptyset \text{ for all non-tower } n\}.$$

Most of the oracle constructions use ($U \oplus$PSPACE-complete)-genericity where U is a subfamily of U_0. We survey three such particular families U, which are relevant to the results of this paper:

- Relative a generic oracle in $U_0 \oplus$ PSPACE-complete (Cohen-generic, according to [FR03]) the following is true: $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, Q is false [IN88], there are disjoint $\mathbf{P}$-inseparable $\mathbf{coNP}$-sets, $\mathbf{P} = \mathbf{UP}$ [FR03], and, moreover, the polynomial hierarchy $\mathbf{PH}$ is infinite (this is a direct corollary of lower bounds for constant depth circuits from [Hås89]).
- Let $U_1 = \{A \in U_0 \mid |A \cap \Sigma^n| \leq 1 \text{ for all tower } n\}$. Relative to a generic oracle in $U_1 \oplus$ PSPACE-complete ($\mathbf{UP}$-generic, according to [FR03]), $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, all disjoint $\mathbf{coNP}$-sets are $\mathbf{P}$-separable [BGS75], Q is true and $\mathbf{P} \neq \mathbf{UP} = \mathbf{PH}$ [FR03].
- $U_2 = \{A \in U_0 \mid |A \cap \Sigma^n| = 1 \text{ for all tower } n\}$. Relative to a generic oracle in $U_2 \oplus$ PSPACE-complete we have $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP} = \mathbf{UP} = \mathbf{PH}$ [MV96].

2.4 Kolmogorov-Generic Oracles

Neither of genericity notions known from the literature is suitable to construct an oracle such that Q is true but $\mathbf{PH}$ is infinite. For instance, this is easily seen for the three genericity notions surveyed above. So we have to define a new one, which we call *Kolmogorov generic*.

For each n fix a binary string Z_n of length $n2^n$ that is incompressible, that is, $C(Z_n) \geq |Z_n| - O(1)$. Divide Z_n into substrings $z_1, \ldots, z_{2^n}$, each of length n. Let Y_n be the set $\{\langle i, z_i \rangle | i \in \{0,1\}^n\}$. (Here we identify i with the integer binary represented by i. The pair $\langle u, v \rangle$ is encoded be the string $\bar{u}v$, where $\bar{u}$ stands for the self-delimiting encoding of u defined in Section 2.2.) Let U be the set of all subsets of $\bigcup_n Y_n$ where the union is over all tower n.

When proving that a certain property holds for a Kolmogorov generic oracle $A = G \oplus H \in U \oplus$ PSPACE-complete we use the fact that every two different lengths of strings in G are exponentially far apart. When discussing a particular polynomial-time computation, we only have to worry about strings at exactly one length in the oracle. Longer strings cannot be queried by the computation and so cannot affect it. Shorter strings can all be queried and found by the computation.

3 Results

Theorem 1. *Relative to a generic oracle in $U \oplus$ PSPACE-complete (a Kolmogorov-generic oracle), Proposition Q is true.*

Proof. We first assume that $\mathbf{P} = \mathbf{PSPACE}$ and prove that Proposition Q is true under a generic oracle $G \in U$.

As explained above it suffices for every polynomial-time oracle NTM M, to prove that relative to a Kolmogorov-generic oracle,

$$\begin{aligned} &M \text{ accepts } \Sigma^* \Rightarrow \text{there is a polynomial time machine} \\ &\text{finding for each input an accepting computation of } M. \end{aligned} \tag{4}$$

Fix M. WLOG M on an input x runs in time $|x|^k + k$, for some constant k independent of its oracle. Indeed, for each oracle nondeterministic Turing machine M (not necessarily polynomial time) and natural k we can construct an

NTM that acts as M supplied with a clock that prevents it to run more than in $|x|^k + k$ steps. If M^A runs in polynomial time then for some k the machine M^A supplied with the clock $|x|^k + k$ is equivalent to M^A.

We will show that the set of oracles satisfying (4) is dense. Let $I = U_\alpha$ be an interval in U. We need to construct a sub-interval J of I such that (4) is true for all $G \in J$. Consider two cases.

(1) There is a sub-interval of I such that for all A in that sub-interval, M^A does not accept Σ^*. Then let J be equal to that sub-interval of I.

(2) There is no such sub-interval. Consider the following polynomial-time deterministic algorithm A that, given an input x of length at least two, finds an accepting path of the computation $M^G(x)$. Let n be the largest tower number smaller or equal to $4|x|^{2k}$. The algorithm A will try to collect enough information about the oracle G so that it could find an accepting path of $M^G(x)$. The algorithm A starts by asking the value of G on all the strings in Y_i for $i \leq \log n$. This can be done in time polynomial in $|x|$.

After that it will iteratively build a set Q of strings from $G \cap Y_n$ starting from an empty set Q. Using the assumption that $\mathbf{P} = \mathbf{PSPACE}$ and the information about G collected so far, the algorithm finds the lexicographically first accepting path of M^G on x under the assumption that $G \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\} = Q$. (Note, M on x cannot query any string in Y_m, for $m > n$.) Such path does exist, as otherwise, the sub-interval J of I, consisting of all G' with $G' \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\} = Q$ and $G' \cap Y_i = G \cap Y_i$ for all $i \leq \log n$ would qualify case (1).

If this path is indeed an accepting path of the computation $M^G(x)$, A is done. If not then there is a string $w \in (G \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\}) \setminus Q$ that is queried along this path. Clearly such w is from Y_n. The algorithm picks the first such w, adds it to the set Q and iterates the process. Clearly, A eventually finds a correct accepting path of $M^G(x)$. We claim that A will find it within polynomially many iterations.

Observe, given M, x, $G \cap Y_{\leq \log n}$ each string in Q can be described by $k \log |x|$ bits by its order number among the queries of M on x on the accepting path described above. The set $G \cap Y_{\leq \log n}$ has at most $n + \log n + \log\log n + \ldots$ strings, each of length at most $\log n$. Thus $G \cap Y_{\leq \log n}$ can be described in at most $O(n \log n)$ bits. Hence if Q reaches size ℓ, we can describe Q by $\ell k \log |x| + O(n \log n) + 2|x| + O(1)$ bits (by Equation (3)).

Recall that all of the strings in Y_n are derived from Z_n. Because of the way Y_n is defined any set A of ℓ strings from Y_n has Kolmogorov complexity at least $\ell n/2 - O(1)$.

Indeed, each element of Y_n is a pair $\langle i, y\rangle$. Let p denote the concatenation of all y's from all pairs $\langle i, y\rangle$ outside A arranged according to the order on i's. The length of p is $n(2^n - \ell)$. The initial string Z_n can be obtained from p by inserting the second components of pairs from A, their first components specify the places where to insert. Thus given p and the shortest description q of A we can find Z_n, and Equation (2) implies

$$n2^n - O(1) \leq C(Z_n) \leq |p| + 2|q| + O(1) = n(2^n - \ell) + 2C(A) + O(1).$$

Since $2+2k\log|x| < n \leq 4|x|^{2k}$, the Kolmogorov complexity of ℓ strings from Y_n is at least $\ell k \log|x| + \ell - O(1)$. Thus Q cannot grow bigger than $O(n\log n) + 2|x| = O(|x|^{2k}\log|x|)$.

We can remove the hypothesis that $\mathbf{P} = \mathbf{PSPACE}$ by first relativizing to an oracle making $\mathbf{P} = \mathbf{PSPACE}$. It is known that relative to every **PSPACE**-complete set H we have $\mathbf{P} = \mathbf{PSPACE}$. Thus, relative to H, Q-property holds relative to a generic oracle in U. As H is computable, the Kolmogorov complexity relativized by H coincides with the unrelativized Kolmogorov complexity (up to an additive constant), and relativization does not change the notion of Kolmogorov genericity. In other words, Property Q holds relative a generic oracle in $U \oplus$ PSPACE-complete. □

Theorem 2. *Relative to a generic oracle in $U \oplus$ PSPACE-complete (a Kolmogorov-generic oracle), for all $k \geq 0$ we have $\Sigma_k^p \neq \Sigma_{k+1}^p$.*

Proof. Meyer and Stockmeyer [MS72] show that if $\Sigma_k^p = \Sigma_{k+1}^p$ then $\Sigma_k^p = \Sigma_j^p$ for all $j \geq k$. So it is sufficient for us to show that $\Sigma_{k-2}^p \neq \Sigma_{k+1}^p$ for all $k \geq 3$ relative to G.

We use the Sipser [Sip83] functions as defined by Håstad [Hås89]. The function f_k^m is represented by a depth k alternating circuit tree with an OR gate at the top with fan-in $\sqrt{m/\log m}$, bottom fan-in $\sqrt{km\log m/2}$ and all other fan-ins are m. Each variable occurs just once at each leaf.

Theorem 3 (Håstad). *Depth $k-1$ circuits computing f_k^m are of size at least $2^{\Omega(\sqrt{m/(k\log m)})}$.*

Pick a tower n. Set $m = 2^{n/k}$. The number of variables of f_k^m is $m^{k-1}\sqrt{k/2} < 2^n$ for large n. For each of the variables of this formula assign a unique $i \in \{0,1\}^n$ so we can in polynomial-time find i from the variable and vice-versa.

Now consider the language $L_k(G)$ such that input 1^n is in $L(G)$ if f_k^m is true if we set the variables corresponding to i to one if $\langle i, z_i\rangle$ is in G and zero otherwise.

We will show relative to a Kolmogorov generic oracle $G \oplus H$, $L_k(G) \in \Sigma_{k+1}^{p,G\oplus H} - \Sigma_{k-2}^{p,G\oplus H}$.

First notice that $L_k(G) \in \Sigma_{k+1}^{p,G\oplus H}$ for all $G \in U$: Consider an alternating Turing machine that uses k-alternations to simulate the circuit. To determine whether a variable corresponding to i is true the machine makes the **NP** query "is there a z such that $\langle i, z\rangle$ is in G." This gives us a $\Sigma_k^{\mathbf{NP},G} = \Sigma_{k+1}^{p,G}$ machine accepting $L_k(G)$.

Let M be an alternating Σ_{k-2}^p oracle Turing machine that runs in time n^j. Let $I = U_\alpha$ be an interval in U. We need to construct a subinterval J of I such that $M^{G\oplus H}$ does not accept $L(G)$ for all $G \in J$. Along the lines of Sipser [Sip83] we can convert the computation to a circuit of depth $k-1$ and size $2^{O(n^j)}$ whose input variables correspond to queries to G. Hardwire the queries not of the form $\langle i, z_i\rangle$ to zero and we have a circuit whose variables are the same as those in f_k^m in the definition of $L_k(G)$ on 1^n. By Theorem 3 for sufficiently large n this circuit cannot compute f_k^m so there must be some setting of the variables where the

circuit and f_k^m have different outputs. Add to the condition α the requirement $\langle i, z_i \rangle \in G$ if variable i is assigned 1 in this setting and the requirement $\langle i, z_i \rangle \notin G$ otherwise. For all $G \in U$ satisfying the resulting condition, $M^{G \oplus H}(1^n)$ accepts iff 1^n is not in $L(G)$. □

We can also show that one-way functions exist relative to G.

Theorem 4. *Relative to a Kolmogorov generic oracle $G \oplus H$, $\mathbf{P} \neq \mathbf{UP}$.*

Proof. Define the relativized language L^X as $\{\langle i, 0^n \rangle : (\exists z)|z| = n \ \& \ \langle i, z \rangle \in X\}$. For a string z of length n, there is at most one string of the form $\langle i, z \rangle$ in G so the language is in $\mathbf{UP}^G$. A simple argument demonstrates that L^G is not in $\mathbf{P}^{G \oplus H}$. □

Can the proof that Q holds relative to a Kolmogorov-generic be lifted to show that Σ_k^pQ holds and we get the collapse of the Δ_k^p and $\Sigma_k^p \cap \Pi_k^p$? The answer is no for $k = 2$ and the proof of this shows that this is true for a broad class of finite extension oracles.

To show that Σ_2^pQ fails relative to a Kolmogorov-generic oracle G, let f be a function from Σ^* to Σ^* where for every x of length n

$$f(x) = y_1 \dots y_{n-1}$$

and

$$y_j = 1 \iff (\exists u, z)\ |u| = n,\ |z| = 2n + \log n,\ \langle xju, z \rangle \in G.$$

No matter what strings are in G, the pigeonhole principle tells us that, for all n, there will always be a *collision*, that is, two different strings x_1 and x_2 of length n such that $f(x_1) = f(x_2)$.

Let M be a $\Sigma_2^{p,G}$ machine that on any input of length n guesses two different strings of length n in its existential step and then accepts iff those strings collide on f. It is clear from the definition of f that M can find these collisions and that it accepts Σ^*. A $\mathbf{P}^{\mathbf{NP}^G}$ machine that finds an accepting path of M could be modified to output the two colliding strings on that path so, without loss of generality, we will assume it does just that.

Theorem 5. *Relative to a Kolmogorov generic oracle $G \oplus H$, no $\mathbf{P}^{\mathbf{NP}}$ machine can find an accepting path of the computation $M(x)$ for every x.*

Proof. Let $\langle R, N \rangle$ be an arbitrary pair consisting of an oracle polynomial time DTM R and an oracle polynomial time NTM N. We will show that the set of all oracles G such that R with oracle $N^{G \oplus H}$ does not find any collision of f^G is dense in U.

WLOG we can assume that there are polynomial upper bounds of the running time of R and N that do not depend on their oracles. Let p_R and p_N stand for those polynomials, respectively.

Let I_α be an interval in U. We will show that for some n there is an interval $I_\beta \subset I_\alpha$ such that for all $G \in I_\beta$, $R^{N^{G \oplus H}}(0^n)$ does not find two strings that collide on f^G.

Pick a large n that is bigger than the maximal length of strings in the domain of α and such that $2n + \log n$ is a tower number. (We call the set of all y such that α contains a condition $y \in G$ or $y \notin G$ the domain of α and use the notation $\operatorname{dom}\alpha$.)

Note that the outcome of $R^{N^{G\oplus H}}$ on input 0^n depends only on membership in G of strings of length at most $p_N(p_R(n))$. First we add to α the requirements $y \notin G$ for all strings $y \notin Y_{2n+\log n} \cup \operatorname{dom}\alpha$ of length at most $p_N(p_R(n))$ and denote by β_0 the resulting condition. The condition β is obtained from β_0 in at most $p_R(n)$ iterations. In ith iteration we define a condition β_i obtained from β_{i-1} by adding some requirements of the form $y \in G$ and $y \notin G$ for $y \in Y_{2n+\log n}$.

Let us explain this in more detail. For $x \in \Sigma^n$ and $j = 1, \ldots, n-1$ let

$$B_{jx} = \{\langle xju, z_{xju}\rangle \mid \ u \in \Sigma^n\}.$$

We call the set $B_x = \bigcup_{j=1}^{n-1} B_{jx}$ the *bag* corresponding to x. The value $f^G(x)$ depends only on $B_x \cap G$. More specifically, jth bit of $f^G(x)$ is 0 if the set $B_{jx} \cap G$ is empty.

On each iteration we choose a set $D \subset \Sigma^n$ of cardinality at most $p_N(p_R(n))$ and *set* oracle's value on the set $\bigcup_{x\in D} B_x$. This means that for every y in this set we include in β_i either the condition $y \notin G$, or the condition $y \in G$. The notation D_i will refer to the set of all strings x such that oracle's value is set on B_x during iterations $s = 1, \ldots, i$. We will keep the following statement invariant: f^G is injective on D_i for all $G \in I_{\beta_i}$.

Additionally, on ith iteration we choose the *desired* answer a_i of $N^{G\oplus H}$ to the i-th query to $N^{G\oplus H}$ in the run of R on input 0^n.

On ith iteration we run R on input 0^n assuming the answers $a_1, \ldots, a_{i-1}$ to oracle queries until R makes ith query q_i to the oracle or outputs a result. If the first option happens, we choose the desired answer of $N^{G\oplus H}$ on q_i as follows.

Assume that $G \in I_{\beta_{i-1}}$ and C is an accepting computation of $N^{G\oplus H}$ on input q_i. We say that $\langle G, C\rangle$ is a *good pair* if the following holds. Let D be the set of all $x \in \Sigma^n$ such that computation C queries a string in the bag of x. The pair $\langle G, C\rangle$ is good if f^G is injective on the set $D \cup D_{i-1}$.

Assume first that there is a good pair $\langle G, C\rangle$. In this case we pick a good pair $\langle \tilde{G}, \tilde{C}\rangle$, define D as explained above and choose YES as the desired answer to ith query. The condition β_i is obtained from β_{i-1} by adding the requirements $y \in G$ for all $y \in \bigcup_{x\in D} B_x \cap \tilde{G}$ and the requirements $y \notin G$ for all $y \in \bigcup_{x\in D} B_x \setminus \tilde{G}$. Note that $N^{\tilde{G}\oplus H}(q_i) = 1$ for all $G \in I_{\beta_i}$.

If there is no good pair $\langle G, C\rangle$ then we choose NO as the desired answer to ith query and set $\beta_i = \beta_{i-1}$, $D_i = D_{i-1}$.

On some iteration $k \le p_R(n)$, R makes no new queries and outputs two strings x_1 and x_2, where f presumably collides. At this point we set oracle's value on all remaining strings in $Y_{2n+\log n}$ as follows. Pick any oracle $\tilde{G} \in I_{\beta_{k-1}}$ such that $f^{\tilde{G}}$ is injective on the set $D_k = D_{k-1} \cup \{x_1, x_2\}$ and such that for all $x \in \Sigma^n \setminus D_k$, $f^{\tilde{G}}(x) = 00\ldots0$. If n is large enough then there is such $\tilde{G}$. Indeed, the length of q_i is at most $p_R(n)$ and thus every computation of $N^{\tilde{G}\oplus H}$ on input q_i runs in time $p_N(p_R(n))$. Hence $|D_k|$ is bounded by the polynomial $p_R(n)p_N(p_R(n)) + 2$.

If 2^{n-1} is bigger than this bound then there are enough strings in the range of f to avoid collision in D_k.

We let β be the condition containing the requirements $y \in G$ for all $y \in \tilde{G}$ of length at most $p_N(p_R(n))$ and the requirements $y \notin G$ for all $y \notin \tilde{G}$ of length at most $p_N(p_R(n))$.

We claim that for all $G \in I_\beta$, $R^{N^{G\oplus H}}$ on 0^n computes the way how we determined. Indeed, if R computes differently for some $G \in I_\beta$ then there must be a query answered in the opposite way than we desire. Let q_i be the first such query. Note that q_i coincides with the ith query in our construction, as all the previous queries are answered by $N^{G\oplus H}$ as we desire. If we have chosen YES as the desired answer to ith query then by construction $N^{G\oplus H}(q_i) = 1$ and thus the desired answer is correct. Therefore this may happen only if we have chosen NO as the ith answer and $N^{G\oplus H}(q_i) = 1$.

By way of contradiction, assume that this is the case. Pick then an accepting computation C of $N^{G\oplus H}$ on q_i. We will show that there is $G' \in I_{\beta_{i-1}}$ such that $\langle G', C\rangle$ is a good pair. Let D be the set of all $x \in \Sigma^n$ such that computation C queries a string in the bag of x. Note that by construction f^G is injective on D_k. (However, f^G may be not injective on $D_{i-1} \cup D$ thus $\langle G, C\rangle$ may be not a good pair.)

We will add to G some strings from $\bigcup_{x\in D\setminus D_k} B_x$ so that for the resulting oracle G' the pair $\langle G', C\rangle$ is good. We may assume that 2^n, the cardinality of every set B_{jx}, is greater than the number of queries along C. For every $x \in D\setminus D_k$ and every j we can change jth bit of $f^G(x)$ to 1 by adding to G a non-queried string from B_{jx}. All of the 2^{n-1} values in the range of f can be obtained in this way. Thus we can change $f^G(x)$ for all $x \in D \setminus D_k$ one by one so that for the resulting oracle G', C is an accepting computation and $f^{G'}(x)$ is injective on $D \cup D_k$ and hence on $D \cup D_{i-1}$. □

4 Conclusion and Open Problems

Is there an oracle relative to which the polynomial-time hierarchy is proper and $\Sigma_k^p\mathrm{Q}$ is true for all k? As a corollary we would get a relativized world where the hierarchy is proper and $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$. The second statement remains open even relative to Kolmogorov generics and, if true, would give a relativized version of the polynomial-time hierarchy that acts like the arithmetic hierarchy.

Acknowledgments

We thank Steve Fenner and Marcus Schaefer for helpful discussions. The work of N. Vereshchagin was in part supported by RFBR grant 06-01-00122 and the work of M. Koucký was in part supported by grant GAČR 201/07/P276 and 201/05/0124.

References

[BGS75] Baker, T., Gill, J., Solovay, R.: Relativization of P=?NP question. SIAM J. Computing 4(4), 431–442 (1975)

[FFKL03] Fortnow, L., Fenner, S., Kurtz, S., Li, L.: An oracle builder's toolkit. Information and Computation 182, 95–136 (2003)

[FFNR03] Fortnow, L., Fenner, S., Naik, A., Rogers, J.: Inverting onto functions. Information and Computation 186, 90–103 (2003)

[FR03] Fortnow, L., Rogers, J.: Separability and one-way functions. Computational Complexity 11, 137–157 (2003)

[Hås89] Håstad, J.: Almost optimal lower bounds for small depth circuits. Advances in Computing Research 5, 143–170 (1989)

[IN88] Impagliazzo, R., Naor, M.: Decision trees and downward closures. In: Proceedings of the 3rd IEEE Structure in Complexity Theory Conference, pp. 29–38. IEEE, New York (1988)

[Kol65] Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Problems of Information Transmission 1(1), 1–7 (1965)

[LV97] Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Graduate Texts in Computer Science. Springer, New York (1997)

[MP91] Megiddo, N., Papadimitriou, C.: On total functions, existence theorems and computational complexity. Theoretical Computer Science 81(2), 317–324 (1991)

[MS72] Meyer, A., Stockmeyer, L.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, pp. 125–129. IEEE, New York (1972)

[MV96] Muchnik, A., Vereshchagin, N.: A general method to construct oracles realizing given relationships between complexity classes. Theoretical Computer Science 157, 227–258 (1996)

[Sip83] Sipser, M.: Borel sets and circuit complexity. In: Proceedings of the 15th ACM Symposium on the Theory of Computing, pp. 61–69. ACM, New York (1983)

[Sol64] Solomonoff, R.J.: A formal theory of inductive inference, part 1 and part 2. Information and Control 7(1-22), 224–254 (1964)

Proved-Patterns-Based Development for Structured Programs*

Dominique Cansell[1] and Dominique Méry[2]

[1] Université de Metz, Loria
[2] Nancy-Université, Université Henri Poincaré Nancy 1
Loria
BP 239, 54506 Vandœuvre-lès-Nancy

1 Introduction

Overview. The development of structured programs is carried out either using bottom-up techniques, or top-down techniques; we show how refinement and proof can be used to help in the top-down development of structured imperative programs. When a problem is stated, the incremental proof-based methodology of event B [9] starts by stating a very abstract model and further refinements lead to finer-grain event-based models which are used to derive an algorithm. The main idea is to consider each *procedure call* as an *abstract event* of a model corresponding to the development of the *procedure*; generally, a procedure is specified by a pre/post specification and then the refinement process leads to a set of events, which are finally combined to obtain the *body of the procedure*. It means that the abstraction corresponds to the procedure call and the body is derived using the refinement process. The refinement process may also use recursive procedures and supports the top-down refinement. The procedure call simulates the abstract event and the refinement guarantees the correctness of the resulting algorithm.

Proof-based Development. Proof-based development methods [6,1,14] integrate formal proof techniques in the development of software systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete events. The relationship between two successive models in this sequence is that of *refinement* [6,1]. The essence of the refinement relationship is that it preserves already proved *system properties* including safety properties and termination.

A development gives rise to a number of, so-called, *proof obligations*, which guarantee its correctness. Such proof obligations are discharged by the proof tool using automatic and interactive proof procedures supported by a proof engine [4].

At the most abstract level it is obligatory to describe the static properties of a model's data by means of an "invariant" predicate. This gives rise to proof obligations relating to the consistency of the model. They are required to ensure that data properties which are claimed to be invariant are preserved by the events of the model. Each refinement step is associated with a further invariant which relates the data of the more concrete model to that of the abstract model and states any additional invariant properties of the

* This work was supported by grant No. ANR-06-SETI-015-03 awarded by the Agence Nationale de la Recherche.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 104–114, 2007.

(possibly richer) concrete data model. These invariants, so-called *gluing invariants* are used in the formulation of the refinement proof obligations.

The goal of a event B development is to obtain a *proved model* and to implement the correctnes-by-contruction [13] paradigm. Since the development process leads to a large number of proof obligations, the mastering of proof complexity is a crucial issue. Even if a proof tool is available, its effective power is limited by classical results over logical theories and we must distribute the complexity of proofs over the components of the current development, e.g. by refinement. Refinement has the potential to decrease the complexity of the proof process whilst allowing for traceability of requirements.

B models rarely need to make assumptions about the *size* of a system being modelled, e.g. the number of nodes in a network. This is in contrast to model checking approaches [10]. The price to pay is to face possibly complex mathematical theories and difficult proofs. The re-use of developed models and the structuring mechanisms available in B help in decreasing the complexity.

Organisation of the paper. Section 2 introduces the modelling language called event B. It introduces definitions for event, refinement, model and corresponding proof obligations. Section 3 describes the development of the sorting problem and the relationship between models and programs; it illustrates the methodology for developing structured programs. Section 4 summarizes the general methodology. Finally, we conclude our work in the last section.

2 The Modelling Framework

2.1 Overview of Event B Development by Step-Wise Refinement

Event-based modelling. Our event-driven approach [2,9] is based on the B notation. It extends the methodological scope of basic concepts in order to take into account the idea of *formal models*. Roughly speaking, a formal model is characterised by a (finite) list x of *state variables* possibly modified by a (finite) list of *events*; an invariant $I(x)$ states properties that must always be satisfied by the variables x and *maintained* by the activation of the events. In the following, we briefly recall definitions and principles of formal models and explain how they can be managed by tools [4,11].

Generalised substitutions are borrowed from the B notation. They provide a means to express changes to state variable values. In its simple form, $x := E(x)$, a generalised substitution looks like an assignment statement. In this construct, x denotes a vector built on the set of state variables of the model, and $E(x)$ a vector of expressions. The interpretation we shall give here to this statement is not however that of an assignment statement. We interpret it as a *logical simultaneous substitution* of each variable of the vector x by the corresponding expression of the vector $E(x)$. There exists a more general normal form of this, denoted by the construct $x : |P(x_0, x)$. This should be read: "x is modified in such a way that the predicate $P(x_0, x)$ holds", where x denotes the *new value* of the vector and x_0 denotes its *old value*. This is clearly non-deterministic in general.

An event has two main parts: a *guard*, which is a predicate built on the state variables, and an *action*, which is a generalised substitution. An event can take one of the three normal forms. The first form ($evnt \;\widehat{=}\;$ **begin** $x : |P(x_0, x)$ **end**) shows an event

that is not guarded: it is thus always enabled and is semantically defined by $P(x, x')$. The second ($evt \ \widehat{=}$ **when** $G(x)$ **then** $x : |Q(x_0, x)$ **end**) and third ($evt \ \widehat{=}$ **any** t **where** $G(t, x)$ **then** $x : |R(x_0, x, t)$ **end**) forms are guarded by a guard which states the necessary condition for these events to occur. Such a guard is represented by WHEN $G(x)$ in the second form, and by **any** t **where** $G(t, x)$ (for $\exists\, t \cdot G(t, x)$) in the third form. We note that the third form defines a possibly non-deterministic event where t represents a vector of distinct local variables. The, so-called, before-after predicate $BA(x, x')$ associated with each of the three event types, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before (x) and just after (x') the "execution" of event evt. The second and the third forms are semantically equivalent to $G(x) \ \wedge \ Q(x, x')$ resp. $\exists\, t \cdot \ (G(t, x) \ \wedge \ R(x, x', t)$.

Proof obligations are produced from events in order to state that an invariant condition $I(x)$ is preserved. Their general form follows immediately from the definition of the before-after predicate, $BA(x, x')$, of each event:

$$I(x) \ \wedge \ BA(x, x') \ \Rightarrow \ I(x')$$

Note that it follows from the two guarded forms of the events that this obligation is trivially discharged when the guard of the event is false. When this is the case, the event is said to be "disabled".

Model Refinement. The refinement of a formal model allows us to enrich a model in a *step-by-step* approach, and is the foundation of our *correct-by-construction* [13] approach. Refinement provides a way to strengthen invariants and to add details to a model. It is also used to transform an abstract model in a more concrete version by modifying the state description. This is done by extending the list of state variables (possibly suppressing some of them), by refining each abstract event into a corresponding concrete version, and by adding new events. The abstract state variables, x, and the concrete ones, y, are linked together by means of a, so-called, *gluing invariant* $J(x, y)$. A number of proof obligations ensure that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines $skip$, (3) no new event takes control for ever, and (4) relative deadlock-freeness is preserved. Details of the formulation of these proofs follows.

We suppose that an abstract model AM with variables x and invariant $I(x)$ is refined by a concrete model CM with variables y and gluing invariant $J(x, y)$. If $BAA(x, x')$ and $BAC(y, y')$ are respectively the abstract and concrete before-after predicates of the same event, we have to prove the following statement, corresponding to proof obligation (1):

$$I(x) \ \wedge \ J(x, y) \ \wedge \ BAC(y, y') \ \Rightarrow \ \exists x' \cdot (BAA(x, x') \ \wedge \ J(x', y'))$$

Now, proof obligation (2) states that $BA(y, y')$ must refine $skip$ ($x' = x$), generating the following simple statement to prove (2):

$$I(x) \ \wedge \ J(x, y) \ \wedge \ BA(y, y') \ \Rightarrow \ J(x, y')$$

For the third proof obligation, we must formalise the notion of the system advancing in its execution; a standard technique is to introduce a variant $V(y)$ that is decreased by each new event (to guarantee that an abstract step may occur). This leads to the following simple statement to prove (3):

$$I(x) \ \wedge \ J(x,y) \ \wedge \ BA(y,y') \ \Rightarrow \ V(y') < V(y)$$

Finally, to prove that the concrete model does not introduce additional deadlocks, we give formalisms for reasoning about the event guards in the concrete and abstract models: $\mathsf{grds}(AM)$ represents the disjunction of the guards of the events of the abstract model, and $\mathsf{grds}(CM)$ represents the disjunction of the guards of the events of the concrete model. Relative deadlock freeness is now easily formalised as the following proof obligation (4):

$$I(x) \ \wedge \ J(x,y) \ \wedge \ \mathsf{grds}(AM) \ \Rightarrow \ \mathsf{grds}(CM)$$

To review, when one refines a model, one can either: refine an existing event by strengthening the guard and/or the before-after predicate (effectively reducing the degree of non-determinism), or add a new event in order to refine the skip event. Furthermore, such refinement guarantees that the set of traces of the refined model contains (modulo stuttering) the traces of the resulting model.

3 Development of Structured Programs

Using a very traditional problem, namely the sorting problem, the section illustrates the general methodology for deriving structured programs, which are using procedures and functions calls. We identify new proof obligations to ensure the correctness of the resulting program.

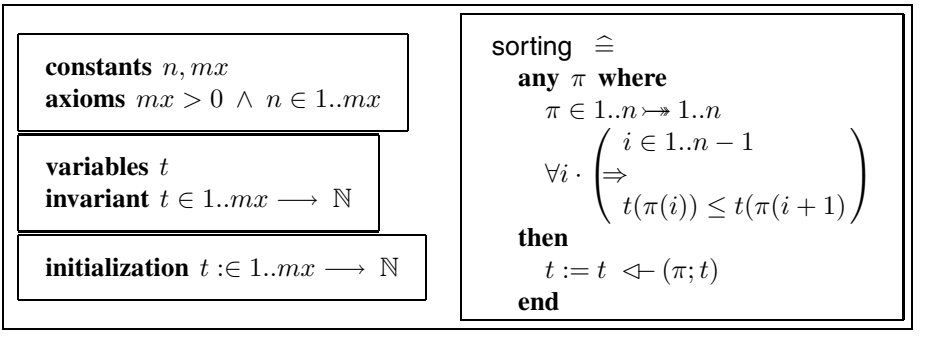
The sorting problem can be simply stated as follows: finding an algorithmic method to sort an array of values. What is a sorting algorithm? The sorting of an array t of mx values can be summarised, in an abstract way, by finding a permutation π over $1..mx$, such that it produces a sorted array from the initial array by applying the permutation. This statement is never included into the resulting code, even if the code is structured. Let us consider for instance the following possible resulting code produced in a classical way.

```
procedure sort(t, n, mx)
begin
  if n ≠ 1 then
    swap(t, imax(t, n, mx), n, mx);
    sort(t, n − 1, mx)
end
```

$swap$ is a procedure, which swaps two values of the array and $imax$ computes the index of the maximal value in the array t. $sort$ is a procedure which sorts the array t over $1..n$. To sort the array t one can call $sort(t, mx, mx)$. Our goal is to develop this structured algorithm using the modelling framework: the event B method.

3.1 First Model for the Sorting Problem

The first abstraction captures the essence of the sorting problem by defining an abstract event modelling the application of a permutation π which is sorting a prefix (between 1 and n) of the array. The event sorting can be considered as *a procedure specification* and it *non-deterministically* chooses a correct permutation. The variable t is a total function from $1..mx$ to $\mathbb{N}$, where mx is a constant value greater than 0. The variable t is initially set to any function from $1..mx$ to $\mathbb{N}$. We state that the execution of the event sorting produces the sorted array t (between 1 and n) in *one shot*, which is corresponding to a *procedure specification*. The constant n belongs to the interval $1..mx$.



In the event sorting, $1..n \rightarrowtail\!\!\!\!\twoheadrightarrow 1..n$ is the set of total one-to-one functions and $(\pi; t)$ is the composition of functions (first π then t) which is a total function over $1..n$. The event B expression $t \ \leftarrow\!\!\!\!\lhd \ (\pi; t)$ is a total function which is equal to $(\pi; t)$ on $1..n$ and t on $n+1..mx$ then only the first n value can change with $(\pi; t)$. The event sorting is simulated by the procedure call of the corresponding code, which is under development. The refinement introduces new details in the models and how the algorithmic process is working.

3.2 Second Model for the Sorting Problem

The second model is a refinement of the first model. It introduces the code of the procedure. The control is introduced by defining a set $STATUS$ which is equal to the set $\{start, call, end\}$. The variable $control$ indicates the current control of the execution. The procedure starts by the evaluation of the boolean expression $n \neq 1$. Either the result is FALSE and the event nothing is triggered; the code of the procedure provides its final result: the array is sorted. Or, the result is TRUE and then first, the code swaps the maximum (event swapmax) and the value at n; then the recursive call (event rec_call) before to complete the computation and to provide the final sorted array. New events modify the array and we introduce a new variable T which is initialised by the same value as t.

nothing $\widehat{=}$
when
$n = 1$
$control = start$
then
$control := end$
end

swapmax $\widehat{=}$
any im **where**
$n \neq 1$
$control = start$
$im \in 1..n \ \wedge \ \forall i.(i \in 1..n \Rightarrow T(i) \leq T(im))$
then
$T := T \ \lhd\!\!- \{im \mapsto T(n)\} \ \lhd\!\!- \{n \mapsto T(im)\}$
$control := call$
end

rec_call $\widehat{=}$
any π **where**
$\pi \in 1..n-1 \rightarrowtail\!\!\!\!\twoheadrightarrow 1..n-1$
$\forall i.(i \in 1..n-2 \Rightarrow t(\pi(i)) \leq t(\pi(i+1)))$
$control = call$
then
$T := T \ \lhd\!\!- (\pi; T) \ || \ control := end$
end

sorting $\widehat{=}$
when
$control = end$
then
$t := T$
end

We should prove that the current model refines the previous model and we use the following invariant:

$$
\begin{array}{l}
T \in 1..mx \longrightarrow \mathbb{N} \\
control \in STATUS \\
(control = end \Rightarrow \exists \pi. \cdot \begin{pmatrix} \pi \in 1..n \rightarrowtail\!\!\!\!\twoheadrightarrow 1..n \ \wedge \ T = (\pi; t) \ \wedge \\ \forall i.(i \in 1..n-1 \Rightarrow T(i) \leq T(i+1)) \end{pmatrix} \\
(control = call \Rightarrow \forall i.(i \in 1..n \Rightarrow T(i) \leq T(n))) \\
(control = call \Rightarrow \exists \pi.(\pi \in 1..n \rightarrowtail\!\!\!\!\twoheadrightarrow 1..n \ \wedge \ T = (\pi; t)) \ \wedge \ n \neq 1) \\
(control = start \Rightarrow T = t)
\end{array}
$$

It describes the computation states of the sorting algorithmic process and it can be used to derive a proved annotation of the resulting procedure. We address the derivation of a procedure in the next sub-section. The invariant was designed during the proof proccess. To prove the existence of a permutation over $1..n$ when the control variable is equal to end (third line of the invariant) we have suggested $(\pi_{guard} \cup \{n \mapsto n\}; \pi_{call})$ where π_{guard} is the permutation over $1..n-1$ given by the guard of event rec_call and π_{call} is the permutation over $1..n$ given by the fifth line of the invariant.

3.3 Towards a Procedure for Sorting an Array

The set of events can be structured according to the variable $status$, which contains the value of the program counter, following rules for deriving algorithms suggested by Abrial [3]. A first skeleton called **abstract** sorting expresses an algorithmic statement of the procedure sorting:

```
abstract sorting
begin
  if n ≠ 1 then
    swapmax;
    rec_call
  else
    nothing
  fi
  concrete sorting
end
```

```
procedure sorting(var t, n, mx)
precondition t = t0 ∧ t0 ∈ 1..mx ⟶ ℕ ∧ 1 ≤ n ≤ mx
postcondition [T, t := t, t0]Assert(end)
local variable T ∈ 1..mx ⟶ ℕ
begin
  T := t;   {start: Assert(start)}
  if n ≠ 1 then
    swapmax(T, n, mx);   {call: Assert(call)}
    sorting(T, n − 1, mx)
  fi {end: Assert(end)}
  t := T
end
```

New variables of the refinement model are local variables of the procedure and we get initial values of these variables in the corresponding part of the refinement model. The event swapmax is still non-deterministic but is defined for T and n. A further refinement is needed to derive a more deterministic procedure and we will reuse a former development corresponding to the procedure $swap$. Comparing the abstract event sorting and the concrete event rec_call, we observe that they are very similar and we can identify a *matching* defined by the following relations: t to T, n to $n - 1$.

Considering the question of proofs related to the development and to the resulting procedure, we can attach annotations to the control points of the procedure as follows: $\{$**ctrl:** $Assert(ctrl)\}$ where $Assert(ctrl)$ is the conjunction of predicates P such that $(control = ctrl \Rightarrow P)$ is in the invariant. The postcondition $[T, t := t, t_0]Assert(end)$ is the predicate $Assert(end)$ where T, t is subtituted by t, t_0 (t_0 is the initial value of the variable t). The predicate looks like the *before-after predicate* of the abstract event sorting.

Proof obligations for the procedure call express that concrete parameters of a call satisfy *properties of formal parameters* and initial conditions of the abstract model: When the *control* is *call*,

- $n - 1 \geq 1$: the actual value of n should be greater than 2.
- $n - 1 \leq mx$ holds.
- $T \in 1..mx \longrightarrow \mathbb{N}$ holds.
- The event rec_call refines the instantiated abstract event sorting.
- A variant decreases to ensure the termination of the recursive call.

Finally, the procedure can be transformed to eliminate redundant informations, like, for instance, the substitution of T by t and the removal of $t := t$; we can use the event keep, which can modify abstract variables in a refinement [5]. The call swapmax(T,n) should be made deterministic and we can develop a procedure by refining the one-shot model and the next sub-section gives details of this development.

3.4 Developing the Event swapmax into a Procedure

In the previous sub-section, the event swapmax is still non-deterministic and it should be transformed into a procedure by applying the same technique of refinement.

The new development starts from a model with the event swapmax executed. The event is generalised by eliminating the control state and the condition over n. We have identified a problem more general than the problem of swapping two cells of t in the procedure sorting; the problem is to swap two cells of t and is solved by the one-shot event:

swapmax $\widehat{=}$
any im **where**
$im \in 1..n$
$\forall i \cdot \left(\begin{array}{l} i \in 1..n \\ \Rightarrow \\ T(i) \leq T(im) \end{array} \right)$
then
$T := T \ \lhd\!\!- \{im \mapsto T(n)\}$
$\lhd\!\!- \{n \mapsto T(im)\}$
end

The variable T is typed by $T \in 1..n \longrightarrow \mathbb{N}$ and is initialised by any function of $1..n$ to $\mathbb{N}$. The refinement model introduces the scheduling in the behaviour by the set $STATUS\_swap$ equal to $\{start, sw, end\}$ and the variable $control$; the invariant is summarised by:

$Im \in 1..n$
$control \in STATUS\_swap$
$(control = sw \Rightarrow \forall i.(i \in 1..n \Rightarrow T(i) \leq T(Im)))$

The refinement model describes a computation such that, first, the value of the index of the maximum is computed and stored in the new variable Im and, secondly, the two cells of T are swapped.

imax $\widehat{=}$
any im **where**
$control = start$
$im \in 1..n$
$\forall i \cdot \left(\begin{array}{l} i \in 1..n \\ \Rightarrow \\ T(i) \leq T(Im) \end{array} \right)$
then
$Im := im \ || \ control := sw$
end

swapmax $\widehat{=}$
when
$control = sw$
then
$T := T \ \lhd\!\!- \{Im \mapsto T(n)\}$
$\lhd\!\!- \{n \mapsto T(Im)\}$
$control := end$
end

The refinement model provides us enough informations for deriving the abstract swapmax and the corresponding procedure by eliminating the control states.

abstract swapmax
begin
imax;
concrete swapmax
end

procedure swapmax(T, n, mx)
local variable Im
begin
imax(T, n, Im, mx);
$T := T \ \lhd\!\!- \{Im \mapsto T(n)\} \ \lhd\!\!- \{n \mapsto T(Im)\}$
end

The call imax(T,n,Im,mx) can be translated into a function call and the mechanism for assigning arguments to parameter T, namely argument passing, is the *call-by-value*

mechanism, since T is not modified. Obviously, the procedure imax is not satisfactory and should be refined into a procedure, which will return the value of the index of the maximal value of T. The process can derive further procedures and preserve the structure of the current program.

4 Principles for Deriving Structured Programs from Event B Developments

We summarise principles applied for deriving *correct-by-construction structured* programs. The main idea relies upon the development of structured programs following a top/down approach, which is clearly well known in earlier works of Dijkstra [12,14], and to use the refinement for controlling the correctness of the resulting program. It relies on a more fundamental question related to the notion of *problem to solve*.

G. Polya [15] describes a set of techniques or recipes which can be used to solve problems. The different steps advocated by Polya can be summarised as follows. A first step is the understanding the problem and list the data, conditions on the data, the unknown elements and the feasibility of the requirements listed in the statement of the problem. It is clearly important to identify the redundancy and the possible inconsistencies; elicitation of requirements is clearly a very important step and it can be driven following an incremental and progressive methodology based on proof checking. The methodology can be based on incremental proof-based developments. However, the link between the problem and the first model remains to be expressed and the refinement is a real help to justify in a very progressive way the choices of design.

Following the thesis of Polya, it appears that the link between the data and the unknown should be defined in an appropriate way. However, Polya mentions the use of auxiliary problems or sub-problems and they lead us to discover new solutions to the given sub-problems or to reuse existing problems having already been solved: the identification of an already seen problem is something that is not so easy to carry out. The identification of a new problem with a given existing (well-understood) problem is possible in a cognitive way but is not yet so obvious in a mathematical and logical framework. The identification of a problem is the question to be addressed; where the classification of the problem seems to be related to the solution and to the statement of the solution. For instance, the problem of searching something in a collection of data or the problem of computing a fixed-point over a structure can be formalised in a mathematical way and an algorithmic solution can be found; the problem of modelling the greedy method [8] is more complex to solve because there are a lot of greedy algorithms which are using the same principle and the question is to be able to provide a general framework for solving this problem on a special case. We should be able to produce a plan of the solution where it is made up of event-based models related by the refinement relationship and of logico-mathematical theories on data. The question is then to check if a problem is identical to another problem or if a problem is a weaker or a stronger instance. Clearly, refinement plays the role of a link between problems as long as we are able to attach a model to a problem. Moreover, the model should be as general as possible and reusable. The question of the analogy among problems is also a very important issue and it is related to an adaptation of the refinement.

The previous section illustrates the use of the refinement for *unfolding events* considered as procedure calls and it defines a general pattern based on refinement and proof, for deriving structured programs.

5 Concluding Remarks and Perspectives

Principles and rules applied in the development are quite simple and they are supported by proof obligations. The first principle is to state the problem by an abstract model, which generally contains an event modelling the one-shot execution of the target procedure. The main idea is to consider the one-shot event as a procedure call and the refinement helps in deriving a refinement model which is structuring the body of the procedure. The procedure call is refined by the text of the procedure which may be expanded in the text. The general approach is a top-down methodology and each refinement step can introduce new sub-problems to discover; the resolution of a problem is solved by the resolution of new sub-problems and each sub-problem is solved by a specific development that is delegated to another development. The refinement drives the structuration of the final solution. Abrial [3] has defined a decomposition notion to split events and variables in two (or more) parts to allow further independent refinements. These refinements produce more and more events which can occur and can then interleave with events of other parts of the decomposition. Our methodology looks like decomposition but is more constrained. We are sure that the code of all events of the refinement runs until completion and then another abstract event can occur: the procedure execution seems to be done in one shot like in the abstraction.

Future works will provide a list of proof obligations that should be added to guarantee the correctness of the final procedure. We have not clearly stated new proof obligations to check for establishing the correct-by-construction statement and we let this point for the full version of the paper. However, proof obligations should state that procedures are called in a correct state with respect to the preconditions of events. Our title was emphasizing the question of proved patterns for aiding in the development of structured programs from the statement of the problem. Finally, the implementation of techniques for helping the construction of models and the generation of proof obligations will be integrated as a plugin in the plateform RODIN and will partly support our approach.

References

1. Abrial, J.-R.: The B book - Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: $B^{\#}$: Toward a synthesis between z and b. In: Bert, D., Bowen, J.P., King, S. (eds.) ZB 2003. LNCS, vol. 2651. Springer, Heidelberg (2003)
3. Abrial, J.-R.: Event based sequential program development: Application to constructing a pointer program. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 51–74. Springer, Heidelberg (2003)
4. Abrial, J.-R., Cansell, D.: Click'n prove: Interactive proofs within set theory. In: TPHOL 2003, pp. 1–24 (2003)
5. Abrial, J.-R., Cansell, D., Méry, D.: Refinement and reachability in event_{b}. In: Treharne, H., King, S., Henson, M.C., Schneider, S. (eds.) ZB 2005. LNCS, vol. 3455, pp. 222–241. Springer, Heidelberg (2005)

6. Back, R.: On correct refinement of programs. Journal of Computer and System Sciences 23(1), 49–68 (1979)
7. Bjørner, D., Henson, M.C̃. (eds.): Logics of Specification Languages. In: EATCS Textbook in Computer Science. Springer, Heidelberg (2007)
8. Cansell, D., Méry, D.: Incremental parametric development of greedy algorithms. In: Merz, S., Nipkow, T. (eds.) Automatic Verification of Critical Systems - AVoCS 2006, 2006-09, Automatic Verification of Critical Systems (AVoCS 2006), Nancy/France, pp. 48–62 (2006)
9. Cansell, D., Méry, D.: The event-B Modelling Method: Concepts and Case Studies, pp. 33–140. Springer, Heidelberg (2007) See[7].
10. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
11. ClearSy, Aix-en-Provence (F). B4FREE (2004), http://www.b4free.com
12. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
13. Leavens, G.T., Abrial, J.-R., Batory, D., Butler, M., Coglio, A., Fisler, K., Hehner, E., Jones, C., Miller, D., Peyton-Jones, S., Sitaraman, M., Smith, D.R., Stump, A.: Roadmap for enhanced languages and methods to aid verification. In: Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006), pp. 221–235. ACM, New York (2006)
14. Morgan, C.: Programming from Specifications. International Series in Computer Science. Prentice-Hall, Englewood Cliffs (1990)
15. Polya, G.: How to Solve It, 2nd edn. Princeton University Press, Princeton, NJ (1957)

Planarity, Determinants, Permanents, and (Unique) Matchings

Samir Datta[1], Raghav Kulkarni[2,⋆], Nutan Limaye[3], and Meena Mahajan[3]

[1] Chennai Mathematical Institute, Siruseri, Chennai 603 103, India
sdatta@cmi.ac.in
[2] Dept. of Computer Science, Univ. of Chicago, U.S.A.
raghav@cs.uchicago.edu
[3] The Institute of Mathematical Sciences, Chennai 600 113, India
{nutan,meena}@imsc.res.in

Abstract. We explore the restrictiveness of planarity on the complexity of computing the determinant and the permanent, and show that both problems remain as hard as in the general case, i.e. GapL and #P complete. On the other hand, both bipartite planarity and bimodal planarity bring the complexity of permanents down (but no further) to that of determinants. The permanent or the determinant modulo 2 is complete for $\oplus$L, and we show that parity of paths in a layered grid graph (which is bimodal planar) is also complete for this class. We also relate the complexity of grid graph reachability to that of testing existence/uniqueness of a perfect matching in a planar bipartite graph.

1 Introduction

For many natural problems on graphs, imposing planarity does not reduce the complexity. For instance, vertex cover is NP-complete, and remains so even for planar degree-3 restrictions; so does planar 3-dimensional matching [15]. The circuit value problem is P-complete, and remains so even if the graph underlying the circuit is restricted to be planar. In [19] and [27], the complexity of several counting problems has been investigated under planar restrictions. More recently, [32] establishes that counting vertex covers remains #P-complete even when restricted to 3-regular planar bipartite graphs. Thus there is some evidence to believe that planarity is not a real restriction at all.

However, there are notable exceptions. In the circuit setting, for instance, monotone circuit value is P-complete, but monotone planar circuit value is in NC [33,14]. Constant-width circuits characterize NC^1 [7], while planar constant-width circuits characterize its subclass ACC^0 [16]. In the graph-theoretic setting, counting the number of perfect matchings in a bipartite graph is #P-hard [28], while counting it in a planar bipartite graph (or even in a planar non-bipartite graph) is in NC [30,21]. Another very recent exception has to do with reachability. Given a directed graph G and two vertices s and t, determining whether

⋆ Part of this work was done while this author was visiting the Chennai Mathematical Institute.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 115–126, 2007.

there is a path from s to t is the canonical complete problem for nondeterministic logspace NL. However, if the graph is planar, then a recent result from [9], building on the techniques of [25,4], shows that the presence, and even the absence, of a path can be detected in unambiguous logspace UL. While UL is known to coincide with NL in the non-uniform setting, and even in the uniform setting under a plausible hardness condition [6], as of now they are not known to coincide unconditionally. So the result of [9] is an instance of planarity reducing the complexity of a problem.

Thus we see that the condition of planarity could be exploited in establishing better upper bounds in some cases. Motivated by the need to better understand how planarity can help, we examine the complexity of determinant, permanent, and unique perfect matchings when restricted to planar instances. Recall that both the determinant and the permanent of the adjacency matrix of a graph G count the total weight of all cycle covers in G, with the one difference that the determinant considers the *signed* weight. Computing the determinant (over integers or rationals) is known to be GapL-complete [13,26,29,31], while computing the permanent is known to be #P-complete (see [28]; the 0-1 permanent equals the number of perfect matchings in a related bipartite graph). However, testing whether the 0-1 permanent is zero is in P and thus significantly easier than #P, whereas testing whether the 0-1 determinant is zero is complete for the exact-counting-in-logspace class $\mathsf{C}_=\mathsf{L}$ [3], and thus at least as hard for NL. Interestingly, the permanent mod 2 equals the determinant mod 2 and is thus easy to compute, in fact complete for the parity logspace class $\oplus\mathsf{L}$. Another complete problem for $\oplus\mathsf{L}$ is checking whether the number of $s \rightsquigarrow t$ paths in a directed acyclic graph is odd. Testing whether a bipartite graph has a perfect matching, B-PM, is known to be hard for NL [11], while testing whether a bipartite graph has a unique perfect matching, B-UPM, is known to be hard for NL and in $\mathsf{C}_=\mathsf{L} \cap \mathsf{NL}^{\oplus\mathsf{L}}$ [18].

We examine planar restrictions of these and related problems. Our main results are summarized in Table 1. (The involved terms are explained in the respective sections.)

This paper is organised as follows. Section 2 describes the notation needed to describe the results of the paper. Sections 3 and 4 describe the hardness and the membership results respectively concerning determinant and permanent. Section 5 describes the hardness of $\oplus$LGGR for $\oplus$L, and Section 6 describes the results concerning planar B-PM and B-UPM.

2 Notation and Preliminaries

L and P denote deterministic logspace and polynomial time computation, respectively. We consider the nondeterministic classes NP and NL, their counting counterparts #P and #L, and the closures of these under subtraction GapP and GapL. We also consider (1) the exact counting in logspace class $\mathsf{C}_=\mathsf{L}$; a language L is in $\mathsf{C}_=\mathsf{L}$ if and only if some GapL function vanishes exactly on strings in L, and (2) the parity logspace class $\oplus\mathsf{L}$; L is in $\oplus\mathsf{L}$ if and only if some GapL

Table 1. Main results

Problem	General bound	Restriction	Our New Bound
Total signed weight of cycle covers (Determinant of adjacency matrix)	GapL-complete	planar	GapL-hard
Total weight of cycle covers (Permanent of adjacency matrix)	#P-complete	planar	#P-hard
		planar bipartite	GapL-complete
		planar bimodal	GapL-complete
Total weight of perfect matchings (Permanent of bip-adjacency matrix)	#P-complete	planar bipartite	GapL-complete
Parity of $\#s \rightsquigarrow t$ paths in directed acyclic graph	$\oplus$L-complete	planar, even layered grid graph	$\oplus$L-hard
Bipartite UPM	NL-hard, in $\mathsf{C}_{=}\mathsf{L} \cap \mathsf{NL}^{\oplus\mathsf{L}}$	planar	in $\oplus$L, L-hard, co-LGGR-hard, equiv to GGUPM
Bipartite PM	NL-hard	planar	L-hard, GGR-hard, equiv to GGPM

function takes odd values exactly on strings in L. It is known that $\mathsf{NL} \subseteq \mathsf{C}_{=}\mathsf{L}$ and that $\oplus\mathsf{L}^{\oplus\mathsf{L}} = \oplus\mathsf{L}$. The canonical complete problem for NL is Reachability in a directed acyclic graph. A complete problem for GapL is computing the determinant of an integer matrix; hence testing singularity of a matrix is complete for $\mathsf{C}_{=}\mathsf{L}$. See for instance [1].

We consider planar graphs specified by planar combinatorial embeddings: such an embedding specifies, for each vertex, the cyclic ordering of edges incident on it in some plane drawing. Testing planarity and obtaining planar combinatorial embeddings can be done in L by the results of [5,24]. A planar embedding of a directed graph is bimodal if at every vertex, all the incoming edges appear contiguously in the cyclic ordering. Not every planar graph has a bimodal embedding. See for instance [23].

A grid graph is a directed graph with vertices laid out on the plane at integer coordinates, and edges going unit distance east-west or north-south only. A grid graph is layered if all horizontal edges are in the same direction (say left-to-right, or x-monotone), and so are all vertical edges (y-monotone).

We will frequently use the following observation:

Proposition 1. *A bipartite graph can be drawn on the plane with straight-line edges, and with no two crossings sharing the same coordinates. The combinatorial embedding corresponding to such a drawing can be obtained in logspace.*

(To draw $K_{n,n}$, place vertices of the first part on the x-axis, vertex u_i at $(0, i)$. Place vertices of the second part on the $x = 1$ line suitably spaced apart; place vertex v_j at $(1, n^{2j})$.)

For any directed graph H with a special source vertex s and sink vertex t, define the split graph Split(H) as follows: (1) split every node v into two nodes,

v_{in} and v_{out}, (2) for every edge (u, v) in the original graph, draw an edge from u_{out} to v_{in}, with the same weight, (3) draw the edges from v_{in} to v_{out} for each v, with weight 1, and (4) delete s_{in} and t_{out}; rename s_{out} and t_{in} as s and t. Note that Spilt(H) is always bipartite. Further, if H has a bimodal planar embedding, then Split(H) is also bimodal planar, and the witnessing embedding can be easily obtained from that of H. (If H is planar but not bimodal, then Split(H) may not be planar at all.)

Corresponding to any $n \times n$ matrix M, we can associate two graphs: G_M is a directed graph on n vertices, with edge $\langle i, j\rangle$ having weight $M(i, j)$, and H_M is an undirected bipartite graph on $2n$ vertices, with edge $(i, n + j)$ having weight $M(i, j)$. M is said to be the adjacency matrix of G_M and the bipartite adjacency matrix of H_M. A cycle cover in a graph is a collection of vertex disjoint cycles spanning the graph. The determinant of a matrix M, $\mathsf{Det}(M)$, equals the total signed weight of all cycle covers in G_M, while its permanent, $\mathsf{Perm}(M)$, equals the total unsigned weight of all cycle covers in G_M. The sign of a cycle cover is $(-1)^k$, where k is the number of even length cycles in the cover. $\mathsf{Perm}(M)$ also equals the total weight of all perfect matchings in H_M. Here the weight of a cycle cover or matching is the product of the weights of its constitutent edges.

3 Planarizing the Determinant and the Permanent: Retaining Hardness

Computing the determinant (over integers) is known to be GapL-complete [13,26,29,31]. We show that it remains hard if the matrix is restricted to be the adjacency matrix of a planar graph. Weights in {0,1} suffice, and if the graph is required to be bipartite then weights in {-1,0,1} suffice. Further, a natural complete problem for GapL is DAG-WT-$s \rightsquigarrow t$-PATHS: finding the total weight of all $s \rightsquigarrow t$ paths in a weighted directed acyclic graph DAG. We show that this problem remains GapL-hard even restricting the DAG to be planar, if we allow negative weights.

We also investigate the complexity of the planar permanent. The permanent itself is #P-complete, though the hardness is under Turing reductions. There are two types of planar restrictions we can consider, and they have quite a different flavour. We want to compute $\mathsf{Perm}(M)$ when either the graph G_M or the graph H_M (see Section 2) is planar. If we require H_M to be planar, then #P-hardness is lost, because the total weight of perfect matchings in a planar (bipartite or otherwise) graph can be done in GapL using the framework of Pfaffians; see [30,21]. We show that this is in fact not just in GapL but also GapL-complete. Though [21] shows that computing the Pfaffian is GapL-complete, the underlying graphs are not planar. We show hardness without recourse to Pfaffians.

If we require that the graph G_M is planar, then we are counting the total weight of cycle covers in a planar graph. We show that this restriction continues to be as hard as the original problem, i.e.#P-hard. On the other hand, if G_M is restricted to be bimodal planar, or simultaneously planar and bipartite, then we show that computing $\mathsf{Perm}(M)$ is GapL-hard. This is the best lower bound

possible, since in the next section we also show that in these cases we can also evaluate the permanent in GapL.

The results of this section can be summarized as follows:

Theorem 1. *The following problems are hard for* GapL *via* $\leq_m^{\log}$ *reductions.*

1. *DAG-WT-s* $\rightsquigarrow$ *t-PATHS for planar graphs (total weight of all* $s \rightsquigarrow t$ *paths in a weighted directed acyclic graph DAG).*
2. Det(M) *for planar* G_M *(total signed weight of cycle covers in planar graph).*
3. Perm(M) *for planar bipartite* G_M *(total weight of cycle covers in* G_M*).*
4. Perm(M) *for planar bimodal* G_M *(total weight of cycle covers in* G_M*).*
5. Perm(M) *for planar bipartite* H_M *(total weight of perfect matchings in* H_M*).*

Further, computing Perm(M) *for planar* G_M *(total weight of cycle covers in planar graph) is hard for* #P.

We now sketch the proofs for each of these claims.

GapL $\leq_m^{\log}$ Planar-DAG-WT-$s \rightsquigarrow t$-PATHS: We start with the canonical GapL-complete problem Directed Path Difference (see for instance [26,22]). The input is a directed graph G with special vertices s, t_+ and t_-, and the desired output $\#(G, s, t_+, t_-)$ is the difference in the number of $s \rightsquigarrow t_+$ paths and the number of $s \rightsquigarrow t_-$ paths. Without loss of generality, we can assume that (1) G is acyclic and layered (vertices appear in layers and all edges go from a layer to the next layer), (2) s is on the first layer and t_+ and t_- on the last layer, and all $s \rightsquigarrow t_+$ or $s \rightsquigarrow t_-$ paths are of even length, (3) all edges having weight 1, and (4) the number of vertices is odd.

We create a new vertex t and add edge $\langle t_+, t\rangle$ with weight 1, and edge $\langle t_-, t\rangle$ with weight -1, to get G_1. All $s \rightsquigarrow t$ paths are of odd length. The hard function is the total weight of all $s \rightsquigarrow t$ paths in G_1.

Now we planarize G_1 as follows: We draw G_1 in the plane, with edge crossings (as described in Proposition 1). We replace each crossing by the gadget shown alongside to get a planar graph G_2. Observe that for any vertices a, b in G_1, the weight of each $a \rightsquigarrow b$ path as well as the parity of the length of the path is preserved in G_2. Since G (and G_1) was bipartite, so is G_2. (Here bipartiteness is in the undirected sense: there are no undirected odd cycles.) Also, the embedding of G_2 we have is *upward planar*; it is planar and all edges are monotonic w.r.t. the x-coordinate.[1] In particular, this implies that the embedding of G_2 is bimodal. Without loss of generality, assume that G_2 has an odd number N of vertices.

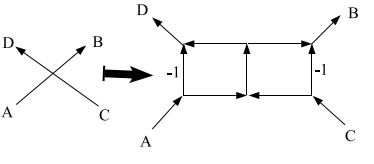


Fig. 1. Planarizing Gadget 1

We want to map paths in G_2 to (signed) cycle covers in a related graph. Toda [26] achieves this by subdividing every edge, adding self-loops everywhere except at s and then adding edge $\langle t, s\rangle$. We adapt this proof in two different ways.

[1] Using the techniques of Section 5, we can even ensure that G_2 is a *layered grid graph.*

GapL $\leq_m^{\log}$ PLANAR 0-1 DET: The method of [26] does not eliminate negative weights. To handle this, we selectively subdivide only those edges with weight 1. Edges with weight -1 are not subdivided, but their weight is changed to 1. We can then show that this graph, say G_3, has the desired properties.

GapL$\leq_m^{\log}$ {-1,0,1} BIPARTITE PLANAR BIMODAL DET/PERM: The above method loses bipartiteness not just because it adds self-loops, but also because of asymmetric subdivisions for weight 1 or -1. Instead, we can construct Split(G_2) and add to it edges $\langle v_{out}, v_{in}\rangle$ for each $v \notin \{s,t\}$, and the edge $\langle t,s\rangle$; all these edges have weight 1. Call this graph G_4; we can now show that it has the desired properties.

If A_3, A_4 are the adjacency matrices of G_3, G_4 respectively, then

$$\mathsf{Det}(A_4) = \mathsf{Perm}(A_4) = \mathsf{Det}(A_3) = \#(G_2,s,t) = \#(G_1,s,t) = \#(G,s,t_+,t_-)$$

GapL- $\leq_m^{\log}$ BIPARTITE PLANAR PERFECT MATCHINGS: Let G_5 be the undirected graph underlying Split(G_2); then G_5 is planar bipartite, and $s \rightsquigarrow t$ paths in G_2 are in 1-1 correspondence with perfect matchings in G_5 of the same weight. Thus the sum of the weights of the perfect matchings in G_5 is precisely $\#(G_2,s,t)$. (See [11,18] for details.)

PERM $\leq_m^{\log}$ PLANAR PERM: We now show that computing $\mathsf{Perm}(M)$, when G_M is planar, is as hard as computing arbitrary permanents (i.e. #P-hard). Recall that $\mathsf{Perm}(M)$ computes the total weight of all cycle covers in G_M. Let N be the $n \times n$ matrix whose permanent we wish to compute. Consider the matrix $A = \begin{pmatrix} 0_n & N \\ I_n & 0_n \end{pmatrix}$ where I_n and 0_n denote the identity and the all-zeros matrices of size n. Clearly $\mathsf{Perm}(A) = \mathsf{Perm}(N)$. Consider a drawing of the directed bipartite graph G_A as described in Proposition 1.

As was done for the determinant, we replace each crossing with a planarity gadget so as to preserve the total weights of cycle covers. The planarity gadget used is shown alongside. Cycle covers using exactly one of the two edges AB or CD will now use the corresponding length 3 path $AXYB$ or $CYXD$. Cycle covers using neither of these edges will now use the 2-cycle XY. Cycle covers using both edges are essentially spliced; locally, we use instead the paths AXD and BYC.

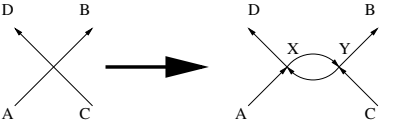


Fig. 2. Planarizing Gadget 2

Applying this planarity gadget to all crossings, we obtain a planar graph G_6 with adjacency matrix M. Since $\mathsf{Perm}(M) = \mathsf{Perm}(A) = \mathsf{Perm}(N)$, we have established the hardness of planar permanent.

Note that this planarity gadget preserves neither bipartiteness nor bimodality. This is not surprising, given the results of the next section.

4 Easy Versions of Planar Permanent Restrictions

We now show that certain planar restrictions of the permanent are significantly easier than #P, in fact, they are computable in GapL. We establish the following.

Theorem 2. *The following functions are computable in* GapL.

1. Perm(M) *for planar bipartite* G_M *(total weight of cycle covers in* G_M*).*
2. Perm(M) *for planar bimodal* G_M *(total weight of cycle covers in* G_M*).*
3. *Even-Odd Crossings Difference: The difference between the total weight of cycle covers with even number of crossings and the total weight of cycle covers with odd number of crossings, in a given plane drawing of a graph* G*.*

The proof of the first two results exploits the fact that finding the total weight of perfect matchings in planar graphs can be computed in GapL ([30,21]).

Let $G_M = (V, E)$ be the given bipartite (directed) graph, with bipartition $X \dot\cup Y$. Let E_1 be those edges of E directed from X to Y, and E_2 be the remaining edges, and let $G_i = (V, E_i)$ for $i = 1, 2$ be planar bipartite undirected graphs. Then, with an appropriate renumbering of vertices (that can be computed in logspace since bipartite-testing is in L as a consequence of [24]), we have $M = \begin{pmatrix} 0_n & A_1 \\ A_2 & 0_n \end{pmatrix}$ where $H_{A_1} = G_1$ and $H_{A_2} = G_2$. (If G_M were undirected, we would have $A_1 = A_2^T$.) Clearly, $\mathsf{Perm}(M) = \mathsf{Perm}(A_1) \times \mathsf{Perm}(A_2)$. But $\mathsf{Perm}(A_i)$ equals the total weight of perfect matchings in the planar graph G_i, this can be computed in GapL.

If G_M is planar bimodal, then Split(G_M) is planar bipartite bimodal, and the cycle-covers in the two graphs are in bijection. So Perm(M) is the total weight of cycle covers in Split(G_M); we have just shown that this is in GapL.

The third result is really an exploration into how far planarizing gadgets can be pushed. If we can replace the crossings in a graph drawing by a gadget which preserves the weighted sum of cycle covers and also preserves bipartiteness or bimodality, then arbitrary pemanents would be expressible as planar bipartite permanents, implying the unlikely collapse of #P to GapL. This suggests that such gadgets are unlikely to exist.

However, we do have a bipartiteness-preserving gadget which reduces the Even-Odd Crossings Difference problem to cycle covers in planar graphs: Given a specific drawing of the graph, count the difference between the number of cycle covers with even number of crossings and the number of cycle covers with odd number of crossings. The gadget shown alongside will do the job. Now, if we start with a bipartite graph, then the resulting graph will be bipartite planar. So, for bipartite graphs, Even-Odd Crossings Difference can be computed in GapL.

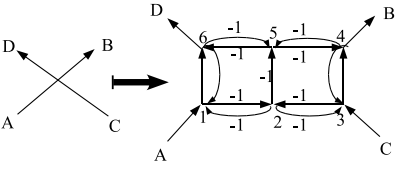


Fig. 3. Planarizing Gadget 3

5 Hardness of $\oplus$LGGR for $\oplus$L

Although the permanent is #P-hard, the permanent mod 2 equals the determinant mod 2 and is thus complete for $\oplus$L. A canonical $\oplus$L-complete problem is $\oplus$PATH-DAG: counting the number of $s \rightsquigarrow t$ paths, mod 2, in a directed acyclic graph (DAG). We show that this remains $\oplus$L hard (under $\leq_m^{\log}$-reductions) even if the DAG is planar, further, even if it is a layered grid graph. $\oplus$LGGR, referred to below, is layered grid graph reachability (LGGR) mod 2, that is, the problem of counting the number of $s \rightsquigarrow t$ paths mod 2 in a layered grid graph.

Theorem 3. $\oplus\mathsf{L} \leq_m^{log} \oplus\mathsf{LGGR}$

The result is significant because for the decision version (reachability in a DAG), the general case is NL-complete while its restriction to planar graphs is known to be in UL $\cap$ co-UL [9]. (Planar Directed Reachability PDR is known to be L-hard, and equivalent to reachability in grid graphs GGR [4], but its exact complexity is still unknown. Reachability in layered grid graphs LGGR is not even known to be L-hard. The complexity of various versions of grid graph reachability is investigated in [2].)

The following chain of reductions establishes the result.

$\oplus$PATH-DAG $\leq_m^{\log}$ $\oplus$PATH-PLANAR-DAG $\leq_m^{\log}$ $\oplus$PATH-x-MON-GG $\leq_m^{\log}$ $\oplus$PATH-LGG = $\oplus$LGGR

The first reduction considers a layered DAG (without loss of generality), draws it according to Proposition 1, and then uses the planarizing gadget of Figure 1, except that all edges have weight 1. This preserves the parity of the number of paths. From here, going to $\oplus$LGGR is achieved by using the grid-graph-embedding technique of [8,10].

6 (Unique) Perfect Matchings in Planar Bipartite Graphs

We now investigate the complexity of checking existence and uniqueness of a perfect matching in a bipartite graph, B-PM and B-UPM, respectively when restricted to planar instances. Both B-PM and B-UPM are known to be NL-hard ([11,18]), but B-UPM is believed to be easier since unlike B-PM, it is known to be in NC (in both $\mathsf{C}_=\mathsf{L}$ and $\mathsf{NL}^{\oplus\mathsf{L}}$ [18]). We provide two further pieces of evidence that B-UPM may be easier by considering the planar restrictions of these problems, Pl-B-PM and Pl-B-UPM. Firstly, we show that while both are L-hard, Pl-B-PM is hard for Planar Directed Reachability PDR, whereas Pl-B-UPM is hard only for co-Layered Grid Graph Reachability co-LGGR. (It is known that PDR is equivalent to co-PDR and to its restriction Grid Graph Reachability GGR, [4]). The hardness of Pl-B-PM for PDR can be viewed as a planarization of the result "Reachability reduces to B-PM". We do not know how to planarize the result "co-Reachability reduces to bipartite-UPM" from [18]. Secondly, we obtain an upper bound of $\oplus$L for Pl-B-UPM. This can be viewed as a planarization of the result "B-UPM is in Reach$^{\oplus\mathsf{L}}$" from [18]: our algorithm is a $\mathsf{GGR}^{\oplus\mathsf{L}}$ algorithm, and since Section 5 shows that $\oplus$LGGR is hard for $\oplus$L, it is in fact in $\mathsf{GGR}^{\oplus\mathsf{LGGR}}$.

We note, however, that the complexity of LGGR (and co-LGGR) is an interesting question in its own right. It is not known whether it is in L, or L- hard, or reducible to its complement co-LGGR. However, its best known upper bound is the same as that for PDR, namely UL ∩ co-UL.

Also, analogous to the equivalence of PDR and GGR, we show that Pl-B-PM and Pl-B-UPM are equivalent to searching for or testing uniqueness of perfect matchings in grid graphs GGPM and GGUPM respectively.

We also consider the related problem of testing uniqueness of a minimum-weight perfect matching. In a bipartite graph with unary weights, this is known to be hard for NL and in $\mathsf{L}^{\mathsf{C}_=\mathsf{L}}$ and $\mathsf{NL}^{\oplus\mathsf{L}}$ [18]. No better upper bound is known for the planar restriction, though the lower bound is also not known to hold. We show that GGR reduces to this planar restriction.

The results in this section can be summarized as follows. (See Figure 4. The pairs of dotted and dashed arrows show the planarizing results.)

Theorem 4. *1.* $(\mathsf{L} \cup \textit{co-}\mathsf{LGGR}) \leq_{proj} \text{Pl-B-UPM} \equiv_{proj} \mathsf{GGUPM} \in \oplus\mathsf{L}$
2. $(\mathsf{L} \cup \mathsf{GGR}) \leq_{proj} \text{Pl-B-PM} \equiv_{proj} \mathsf{GGPM}$
3. Testing uniqueness of a min-weight perfect matching in a planar bipartite graph with unary weights is hard for GGR.

$\mathsf{L} \leq_{proj}$ Pl-B-UPM; $\mathsf{L} \leq_{proj}$ Pl-B-PM: We start with the logspace-complete problem of determining whether there is an $s \rightsquigarrow t$ path in a directed forest G [12]. Given an instance (G, s, t), first construct its split graph G'. Then define H_1 to

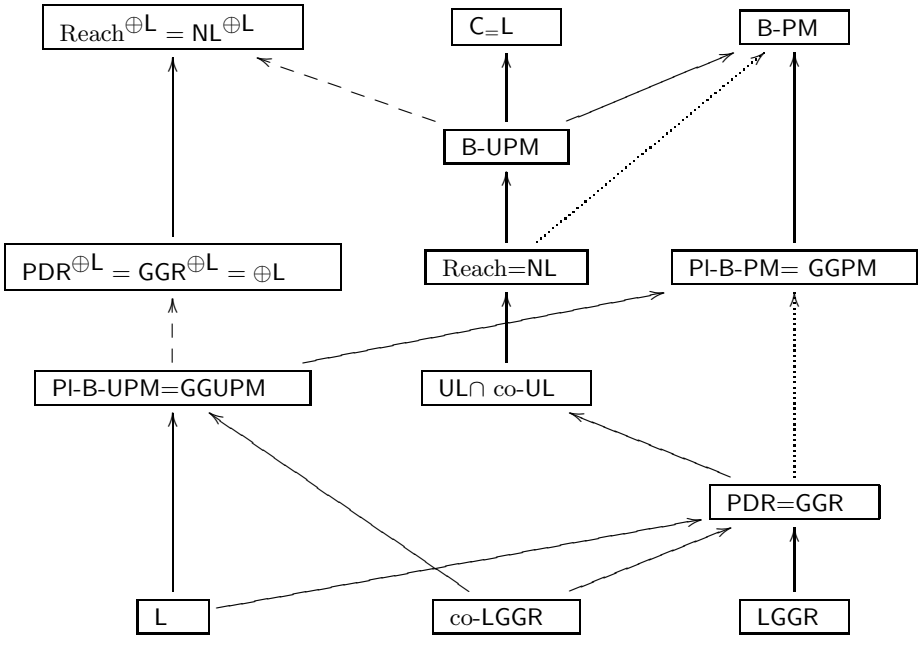


Fig. 4. Pl-B-UPM and Pl-B-PM and their relationships with other classes

be the undirected version of G' and H_2 to be $H_1 \cup \{(s,t)\}$. Since G was a forest, H_1 and H_2 are clearly planar bipartite. Also their construction involves simple projections; it is FO-uniform.

Now, as in [11,18], for every $s \rightsquigarrow t$ path in G, the alternate edges of the corresponding path in H, along with edges of the form (v_{in}, v_{out}) for vertices v not on the path, form a perfect matching in H_1 and H_2. H_1 has no other matching, H_2 has one more which is the added (s,t) edge along with all the edges of the form (v_{in}, v_{out}). Thus $H_1 \in$ Pl-B-PM if and only if $H_2 \in$ Pl-B-UPM if and only if (G,s,t) is not in Forest-Reachability.

co-LGGR$\leq_{proj}$ Pl-B-UPM; GGR$\leq_{proj}$ Pl-B-PM: This follows from carefully analysing the requirements in the above reduction, and some pre-processing.

Unique minimum weight Pl-B-UPM is hard for GGR: For the purpose of this section alone, the weight of a matching is the *sum* of its constituent edges.

Let (G,s,t) be the GGR instance; as discussed above, we can assume that G is bimodal and has s and t on the external face. We now assign weights to the edges of G according to the weighting scheme of [9] to get graph G'; this weighting scheme has the property that $s \rightsquigarrow_G t \iff s \rightsquigarrow_{G'} t \iff$ the minimum weight $s \rightsquigarrow_{G'} t$ path is unique. Now construct $H = \text{Split}(G')$, copying the weight of an edge (u,v) in G' to the edge (u_{out}, v_{in}) of H and assigning weight zero toall the edges of the form (v_{in}, v_{out}). H is a planar bipartite graph and can be obtained via simple projections.

If $(G,s,t) \notin$ GGR, then it is easy to see that H has *no* perfect matching.

If $(G,s,t) \in$ GGR, then the unique minimum-weight path $\rho : s \rightsquigarrow_{G'}$ can be extended to a perfect matching in H $M_\rho = \{(u_{out}, v_{in}) \mid \langle u,v\rangle \in \rho_{G'}\} \cup \{(v_{in}, v_{out}) | v \in G' \text{and } v \notin \rho\}$ of the same weight. Since all (v_{in}, v_{out}) edges in H have weight 0, it is easy to see that this matching is the unique minimum-weight matching in H.

Pl-B-UPM$\leq_m^{\log}$ GGUPM; Pl-B-PM$\leq_m^{\log}$ GGPM: Both these results hold because there is a parsimonious (in the number of perfect matchings) reduction from planar bipartite graphs to grid graphs. This reduction is obtained by a slight modification of the grid graph embedding technique of [4], applied on an equivalent graph with maximum degree 3; the equivalent graph can also be obtained in logspace ([20]).

Pl-B-UPM is in $\oplus$L: In [18], an $\mathsf{NL}^{\oplus\mathsf{L}}$ algorithm for B-UPM is described. Given a bipartite graph G, it proceeds in two stages. In the first stage, an $\mathsf{L}^{\oplus\mathsf{L}}$ procedure either constructs some perfect matching M, or detects that G is not in B-UPM. In the second stage, an NL procedure, with oracle access to M, verifies that M is indeed unique.

We show that for planar bipartite G, the second stage can be performed in $\mathsf{L}^{\mathsf{PDR}}$. Since PDR is known to be in UL$\cap$co-UL[9] which is contained in $\oplus$L, and since $\oplus\mathsf{L}^{\oplus\mathsf{L}} = \mathsf{L}^{\oplus\mathsf{L}} = \oplus\mathsf{L}$ ([17]), it then follows that Pl-B-UPM is in $\oplus$L.

The key idea in obtaining the $\mathsf{L}^{\mathsf{PDR}}$ bound is the following: As described in [18], a given perfect matching M is unique in a bipartite graph G if and only if G has no alternating (with respect to M) cycles. We can consider an auxiliary graph H where an alternating path of length 2 in G, beginning with an M-edge, becomes a directed edge in H; then M is unique in G if H has no cycles. We show that H is planar. This implies that detecting cycles in H is in $\mathsf{L}^{\mathsf{PDR}}$.

References

1. Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajicek, J. (ed.) Complexity of Computations and Proofs, Quaderni di Matematica, vol. 13, pp. 33–72. Seconda Universita di Napoli (2004). An earlier version appeared in the Complexity Theory Column, SIGACT News 28, vol. 4, pp. 2–15 (December 1997)
2. Allender, E., Barrington, D.A.M., Chakraborty, T., Datta, S., Roy, S.: Grid graph reachability problems. In: Proceedings of 21st IEEE Conference on Computational Complexity, pp. 299–313. IEEE Computer Society Press, Los Alamitos (2006)
3. Allender, E., Beals, R., Ogihara, M.: The complexity of matrix rank and feasible systems of linear equations. Computational Complexity 8(2), 99–126 (1999)
4. Allender, E., Datta, S., Roy, S.: The directed planar reachability problem. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 238–249. Springer, Heidelberg (2005)
5. Allender, E., Mahajan, M.: The complexity of planarity testing. Information and Computation 189(1), 117–134 (2004)
6. Allender, E., Rheinhardt, K., Zhou, S.: Isolation, matching and counting: uniform and nonuniform upper bounds. Journal of Computer and System Sciences 59, 164–181 (1999)
7. Barrington, D.: Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. Journal of Computer and System Sciences 38, 150–164 (1989)
8. Barrington, D.A.M.: Grid graph reachability problems. Talk presented at Dogstuhl Seminar on Complexity of Boolean funcions, Seminar Number 02121 (2002)
9. Bourke, C., Tewari, R., Vinodchandran, N.V.: Directed planar reachability is in unambiguous logspace. In: Proceedings of IEEE Conference on Computational Complexity CCC 2007 (to appear)
10. Chakraborty, T., Datta, S.: One-input-face MPCVP is hard for L, but in LogDCFL. In: Proc. of 26th FST TCS Conference. LNCS (2006)
11. Chandra, A., Stockmeyer, L., Vishkin, U.: Constant depth reducibility. SIAM Journal on Computing 13(2), 423–439 (1984)
12. Cook, S.A., McKenzie, P.: Problems complete for L. Journal of Algorithms 8, 385–394 (1987)
13. Damm, C.: DET=$\mathrm{L}^{(\#\mathrm{L})}$. Technical Report Informatik–Preprint 8, Fachbereich Informatik der Humboldt–Universität zu Berlin (1991)
14. Delcher, A.L., Kosaraju, S.R.: An NC algorithm for evaluating monotone planar circuits. SIAM Journal of Computing 24(2), 369–375 (1995)
15. Dyer, M.E., Frieze, A.M.: Planar 3DM is NP-complete. J. Algorithms 7(2), 174–184 (1986)
16. Hansen, K.: Constant width planar computation characterizes ACC^0. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 44–55. Springer, Heidelberg (2004)

17. Hertrampf, U., Reith, S., Vollmer, H.: A note on closure properties of logspace MOD classes. Information Processing Letters 75(3), 91–93 (2000)
18. Hoang, T.M., Thierauf, T., Mahajan, M.: On the bipartite unique perfect matching problem. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 453–464. Springer, Heidelberg (2006)
19. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Stearns, R.E.: The complexity of planar counting problems. SIAM Journal on Computing 27(4), 1142–1167 (1998)
20. Kulkarni, R., Mahajan, M.: Seeking a vertex of the planar matching polytope in nc. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 472–483. Springer, Heidelberg (2004)
21. Mahajan, M., Subramanya, P.R., Vinay, V.: The combinatorial approach yields an NC algorithm for computing Pfaffians. Discrete Applied Mathematics 143(1-3), 1–16 (2004)
22. Mahajan, M., Vinay, V.: Determinant: combinatorics, algorithms, complexity. Chicago Journal of Theoretical Computer Science, 5 (December 1997), http://www.cs.uchicago.edu/publications/cjtcs
23. Mohar, B., Thomassen, C.: Graphs on Surfaces. John Hopkins University Press, Maryland (2001)
24. Reingold, O.: Undirected *st*-connectivity in logspace. In: Proc. 37th STOC, pp. 376–385 (2005)
25. Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. SIAM J. Comp. 29, 1118–1131 (2000)
26. Toda, S.: Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. of Comp. Sc. & Information Mathematics, Univ. of Electro-Communications, Chofu-shi, Tokyo (1991)
27. Vadhan, S.: The complexity of counting in sparse, regular, and planar graphs. SIAM Journal on Computing 31(2), 398–427 (2001)
28. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)
29. Valiant, L.G.: Why is boolean complexity theory difficult? In: Paterson, M.S. (ed.) Boolean Function Complexity. London Mathematical Society Lecture Notes Series, vol. 169. Cambridge University Press, Cambridge (1992)
30. Vazirani, V.: NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. Information and Computation 80(2), 152–164 (1989)
31. Vinay, V.: Semi-unboundedness and complexity classes. PhD thesis, Indian Institute of Science, Bangalore (July 1991)
32. Xia, M., Zhao, W.: #3-regular bipartite planar vertex cover is #P-complete. In: TAMC, pp. 356–364 (2006)
33. Yang, H.: An NC algorithm for the general planar monotone circuit value problem. In: Proceedings of 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 196–203. IEEE Computer Society Press, Los Alamitos (1991)

Equivalence Problems for Circuits over Sets of Natural Numbers

Christian Glaßer, Katrin Herr, Christian Reitwießner,
Stephen Travers, and Matthias Waldherr

Julius-Maximilians-Universität Würzburg, Theoretische Informatik, Germany

Abstract. We investigate the complexity of *equivalence problems* for $\{\cup, \cap, {}^-, +, \times\}$-circuits computing sets of natural numbers. These problems were first introduced by Stockmeyer and Meyer (1973). We continue this line of research and give a systematic characterization of the complexity of equivalence problems over sets of natural numbers. Our work shows that equivalence problems capture a wide range of complexity classes like NL, $\mathrm{C_=L}$, P, Π_2^{P}, PSPACE, NEXP, and beyond. McKenzie and Wagner (2003) studied related *membership problems* for circuits over sets of natural numbers. Our results also have consequences for these membership problems: We provide an improved upper bound for the case of $\{\cup, \cap, {}^-, +, \times\}$-circuits.

1 Introduction

In 1973, Stockmeyer and Meyer [7] defined and investigated equivalence problems for *integer expressions*. They considered expressions that can be built up from single natural numbers by using Boolean operations ($^-$, $\cup$, $\cap$), addition ($+$), and multiplication ($\times$).

The *equivalence problem for integer expressions* is the question of whether two given such expressions describe the same set of natural numbers. Restricting the set of allowed operations results in equivalence problems of different complexities. Stockmeyer and Meyer [7] showed that the equivalence test for expressions over $\{^-, \cup, \cap, +\}$ is PSPACE-complete, and that this problem becomes Π_2^{P}-complete if one restricts to operations from $\{\cup, +\}$.

We continue these investigations and study equivalence problems over natural numbers in a systematic way. Despite of their simple definition, integer expressions are powerful enough to describe highly non-trivial sets. For instance, the set of primes can be described as

$$\textsc{Primes} = \overline{\overline{0 \cup 1} \times \overline{0 \cup 1}} \ \cap \ \overline{0 \cup 1}.$$

This can easily be verified: The complement of $\{0, 1\}$ multiplied with itself yields all composite numbers. Taking its complement gives the set consisting of 0, 1, and all primes. The intersection with $\overline{0 \cup 1}$ yields the set of primes. Expressions

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 127–138, 2007.

like this illustrate that equivalence problems for integer expressions comprise some of the most prominent, unsolved problems in mathematics.

In 1742, Christian Goldbach stated his famous conjecture as a footnote in a letter to Leonhard Euler: "*At least it seems that every number greater than* 1 *is a sum of three prime numbers.*"[1]

Euler answered with an equivalent version of this conjecture which nowadays we call the Goldbach conjecture.

Goldbach Conjecture: Every even integer ≥ 4 *is the sum of two primes.*

The following integer expression describes exactly the set of integers that are counter examples for the Goldbach conjecture.

$$\textsc{CounterExamples} = (2 \times \overline{0 \cup 1}) \cap \overline{\textsc{Primes} + \textsc{Primes}}$$

The left set of the intersection is the set of even integers greater than or equal to 4, while the right set consists of those integers that are not a sum of two primes. The Goldbach conjecture is true if and only if the set of counter examples is empty. Therefore,

$$\textit{Goldbach conjecture holds} \iff \textsc{CounterExamples} \textit{ is equivalent to } 0 \cap \overline{0}.$$

We have seen that the Goldbach conjecture can be formulated as an equivalence problem for integer expressions. So already at this point, the expressiveness of integer expressions makes us aware of the possibility that the general equivalence problem might be undecidable. Indeed, the decidability of the general equivalence problem will be one of our open questions.

Stockmeyer and Meyer's [7] motivation for the study of equivalence problems for integer expressions originated from equivalence problems for Kleene's regular expressions [4]. Since then, several variants and generalizations of integer expressions have been studied. Beside integer expressions (which we call integer formulas) researchers were also interested in *integer circuits* which were introduced by Wagner [10]. The latter represent expressions in a succinct way and so yield problems of higher complexity.

Wagner [10], Yang [12], and McKenzie and Wagner [5] studied the complexity of *membership problems* for formulas and circuits over natural numbers: Here, for a given circuit C and a number n, one has to decide whether n belongs to the set that is described by C. Breunig [2] studied membership problems for formulas and circuits over $\mathbb{N}^+$, the positive integers, while Travers [9] studied the variant for $\mathbb{Z}$, the integers.

In this paper, we study equivalence problems for formulas and circuits over natural numbers. In particular, this contains the equivalence problems for formulas that Stockmeyer and Meyer [7] were interested in. For most of these equivalence problems we can precisely characterize their complexity.

[1] Note that at that time, 1 was considered to be a prime.

It turns out that our results also have consequences for the known results about membership problems. In fact, our upper bound for the equivalence problem for $\{\cup, \cap, ^-, +, \times\}$-circuits yields an improved upper bound for the membership problem for $\{\cup, \cap, ^-, +, \times\}$-circuits. This is the first nontrivial upper bound for $\mathrm{MC}_{\mathbb{N}}(\cup, \cap, ^-, +, \times)$, the most general membership problem.

Our main open question is whether the unrestricted version of the equivalence problem, $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, ^-, +, \times)$, is decidable or not. While we can show that this problem is equivalent to the corresponding membership problem, the upper bound we provide is not a decidable upper bound. So if one proves that $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, ^-, +, \times)$ is undecidable, then it follows that $\mathrm{MC}_{\mathbb{N}}(\cup, \cap, ^-, +, \times)$ also is undecidable.

A summary of our results and a discussion of open problems can be found in the conclusions section.

2 Preliminaries

We fix the alphabet $\Sigma = \{0, 1\}$. Σ^* is the set of words, and $|w|$ is the length of a word $w \in \Sigma^*$. $\mathbb{N}$ denotes the set of the natural numbers, which include zero, whereas $\mathbb{N}^+$ denotes $\mathbb{N} - \{0\}$. For $a, b \in \mathbb{N}$ we define $[a, b] \stackrel{df}{=} \{a, a+1, \ldots, b-1, b\}$ if $a \leq b$ and $[a, b] \stackrel{df}{=} \emptyset$ otherwise. The binary representation of a natural number n is denoted by $\mathrm{bin}(n)$.

We extend the arithmetical operations $+$ and $\cdot$ to subsets of $\mathbb{N}$: Let $M, N \subseteq \mathbb{N}$. We define the sum of M and N as $M + N \stackrel{df}{=} \{m + n : m \in M \text{ and } n \in N\}$. We define the product of M and N as $M \times N \stackrel{df}{=} \{m \cdot n : m \in M \text{ and } n \in N\}$. In some cases we will identify the singleton $\{a\}$ with a. Unless otherwise stated, the domain of a variable is $\mathbb{N}$.

For a nondeterministic logarithmic space machine M, define $\mathrm{acc}_M(x)$ as the number of accepting paths of M on input x. The class #L consists of precisely these functions. A set A is in $\mathrm{C}_=\mathrm{L}$ if there exist $f, g \in \#\mathrm{L}$ such that

$$x \in A \Leftrightarrow f(x) = g(x) \text{ for every } x \in \Sigma^*.$$

See [1] for a survey on these counting classes.

For a complexity class $\mathcal{C}$, let $\exists^{\mathrm{P}} \cdot \mathcal{C}$ denote the class of languages L such that there exists a polynomial p and a $B \in \mathcal{C}$ such that for all x,

$$x \in L \iff \exists y \big(|y| \leq p(|x|), (x, y) \in B \big).$$

Let $\mathcal{C}$ and $\mathcal{D}$ be complexity classes. We define

$$\mathcal{C} \vee \mathcal{D} \stackrel{df}{=} \{A \cup B \,\big|\, A \in \mathcal{C}, B \in \mathcal{D}\}.$$

The symmetric difference of sets A and B is defined as $A \triangle B = (A - B) \cup (B - A)$. The complex version is defined as $\mathcal{C} \oplus \mathcal{D} = \{A \triangle B : A \in \mathcal{C}, B \in \mathcal{D}\}$.

For a class of languages $\mathcal{C}$ which is closed under union and intersection, the Boolean hierarchy over $\mathcal{C}$ [11] is the family of classes $\mathcal{C}(k)$ and co$\mathcal{C}(k)$ where $k \geq 1$,

$$\mathcal{C}(k) \stackrel{df}{=} \overbrace{\mathcal{C} \oplus \mathcal{C} \oplus \cdots \oplus \mathcal{C}}^{k \text{ times}}, \text{ and}$$
$$\text{co}\mathcal{C}(k) \stackrel{df}{=} \left\{\overline{L} : L \in \mathcal{C}(k)\right\}.$$

Unless stated otherwise, all hardness- and completeness-results are in terms of logspace many-one reducibility.

The class DLOGCFL was introduced by McKenzie and Wagner [5]. It is the deterministic restriction of the class LOGCFL [8]. A language L belongs to DLOGCFL if it can be accepted by a deterministic, logarithmic space-bounded Turing machine M that has also access to a pushdown store that is not subject to the logarithmic space-bound. Furthermore, M must have polynomial running time.

3 Decision Problems for Circuits over Sets of Natural Numbers

We define *circuits over sets of natural numbers* and related decision problems.

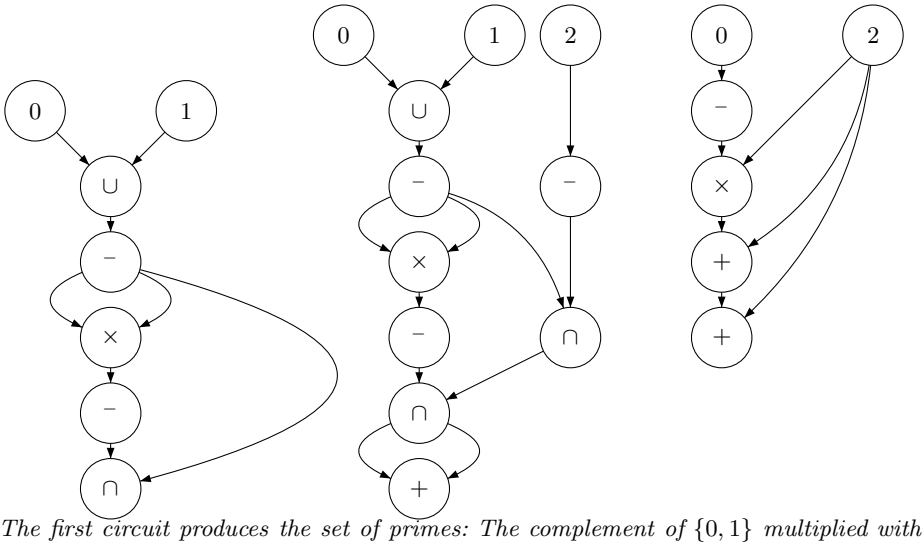
A *circuit* $C = (V, E, g_C)$ is a finite, non-empty, directed, acyclic graph (V, E) with a specified node $g_C \in V$. We remark that the graph can contain multi-edges, that it does not have to be connected, and that $V = \{1, 2, \ldots, n\}$ for some $n \in \mathbb{N}$. Moreover, the nodes in the graph (V, E) are topologically ordered, i.e., for all $v_1, v_2 \in V$, if $v_1 < v_2$, then there is no path from v_2 to v_1. The nodes in V are also called *gates*. Nodes with indegree 0 are called *input gates* and g_C is called *output gate*. If in a circuit there is an edge going from gate u to gate v, then we say that u is a *direct predecessor* of v and v is the *direct successor* of u. If there is a path from u to v but u is not a direct predecessor of v, then u is an *indirect predecessor* of v and v is an *indirect successor* of u.

Let $\mathcal{O} \subseteq \{\cup, \cap, ^-, +, \times\}$. An *$\mathcal{O}$-circuit* $C = (V, E, g_c, \alpha)$ is a circuit (V, E, g_c) with an attached labeling function $\alpha : V \to \mathcal{O} \cup \mathbb{N}$ such that the following holds: Each gate has an indegree in $\{0, 1, 2\}$, gates with indegree 0 have labels from $\mathbb{N}$, gates with indegree 1 have labels $^-$, and gates with indegree 2 have labels from $\{\cup, \cap, +, \times\}$. An *$\mathcal{O}$-formula* is an $\mathcal{O}$-circuit that only contains nodes with outdegree ≤ 1. For each of its gates g, the $\mathcal{O}$-circuit $C = (V, E, g_c, \alpha)$ computes a set $I(g) \subseteq \mathbb{N}$ as follows:

If g is an input gate, then $I(g) \stackrel{df}{=} \alpha(g)$. If g has label $^-$ and direct predecessor g_1, then $I(g) \stackrel{df}{=} \mathbb{N} - I(g_1)$. If g has label $\circ \in \{\cup, \cap, +, \times\}$ and direct predecessors g_1 and g_2, then $I(g) \stackrel{df}{=} I(g_1) \circ I(g_2)$.

The *set computed by* C is $I(C) = I(g_C)$; for simplification we will sometimes write C instead of $I(C)$.

Example 1. *We present circuits for the expressions given in the introduction.*



The first circuit produces the set of primes: The complement of $\{0, 1\}$ *multiplied with itself gives all composite numbers. So the complement of this set yields the set of primes and additionally* 0 *and* 1*. We can remove* $0, 1$ *with a* $\cap$*-gate and obtain the set of all primes. The second circuit produces all sums of two odd primes. The third circuit produces all even numbers greater than four. Hence, a terminating algorithm that decides whether the last two circuits are equivalent would answer the Goldbach conjecture.*

Definition 2. *Let* $\mathcal{O} \subseteq \{\cup, \cap, {}^{-}, +, \times\}$*. We define* membership problems *and* equivalence problems *for circuits and formulas.*

$$\mathrm{MC}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{(C, b) \,|\, C \text{ is an } \mathcal{O}\text{-circuit, } b \in \mathbb{N}, \text{ and } b \in I(C)\}$$
$$\mathrm{MF}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{(C, b) \,|\, C \text{ is an } \mathcal{O}\text{-formula, } b \in \mathbb{N}, \text{ and } b \in I(C)\}$$
$$\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{(C_1, C_2) \,|\, C_1, C_2 \text{ are } \mathcal{O}\text{-circuits such that } I(C_1) = I(C_2)\}$$
$$\mathrm{EF}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{(C_1, C_2) \,|\, C_1, C_2 \text{ are } \mathcal{O}\text{-formulas such that } I(C_1) = I(C_2)\}$$

When an $\mathcal{O}$-circuit $C = (V, E, g_c, \alpha)$ is used as input for an algorithm, then we use a suitable encoding such that it is possible to verify in deterministic logarithmic space whether a given string encodes a valid circuit. In the following, we will therefore assume that all algorithms start with such a validation of their input strings.

We summarize known results about the complexity of equivalence problems.

Theorem 3. *1.* [5] $\mathrm{EC}_{\mathbb{N}}(+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\mathrm{C}_{=}\mathrm{L}$*.*
2. [5] $\mathrm{EC}_{\mathbb{N}}(+, \times)$ *is in* coNP*.*
3. [7] $\mathrm{EF}_{\mathbb{N}}(\cup, +)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* Π_2^{P}*.*
4. [7] $\mathrm{EF}_{\mathbb{N}}({}^{-}, \cup, \cap, +)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* PSPACE*.*
5. [6] $\mathrm{EC}_{\mathbb{N}}(+, \times)$ *is in* coRP*.*

3.1 Relations to Membership Problems

In this subsection we discuss that in some cases (i.e., for several $\mathcal{O} \subseteq \{\cap, \cup, {}^-, +, \times\}$), the complexity of the equivalence problem $\mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ is related to the complexity of $\mathrm{MC}_{\mathbb{N}}(\mathcal{O})$ in a straightforward way. As a consequence, we obtain several general upper and lower bounds for $\mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ which we summarize below. In contrast, in the following sections more sophisticated arguments are needed to establish optimal bounds.

Lemma 4. *The following holds.*

1. *If* $\mathcal{O} \subseteq \{\cap, \cup, {}^-, +\}$, *then* $\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \in \mathrm{coNP}^{\mathrm{MC}_{\mathbb{N}}(\mathcal{O})}$ *and* $\mathrm{EF}_{\mathbb{N}}(\mathcal{O}) \in \mathrm{coNP}^{\mathrm{MF}_{\mathbb{N}}(\mathcal{O})}$.
2. *If* $\mathcal{O} \subseteq \{\cap, \cup, +, \times\}$, *then* $\mathrm{EF}_{\mathbb{N}}(\mathcal{O})$ *is in* $\mathrm{coNP}^{\mathrm{MF}_{\mathbb{N}}(\mathcal{O})}$.

The following proposition shows that in many cases, the intuition that equivalence problems are a generalization of membership problems is correct.

Proposition 5. *The following holds.*

1. *If* $\{\cap\} \subseteq \mathcal{O}$ *or* $\{\cup\} \subseteq \mathcal{O}$, *then* $\mathrm{MC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ *and* $\mathrm{MF}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{EF}_{\mathbb{N}}(\mathcal{O})$.
2. *If* $\mathcal{O} \subseteq \{+, \times\}$, *then* $\mathrm{MC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ *and* $\mathrm{MF}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{EF}_{\mathbb{N}}(\mathcal{O})$.

In some cases, equivalence problems are not harder than membership problems.

Proposition 6. 1. *If* $\{\cap, {}^-, \times\} \subseteq \mathcal{O}$ *or* $\{\cup, {}^-, \times\} \subseteq \mathcal{O}$, *then* $\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}_{\mathbb{N}}(\mathcal{O})$.
2. *If* $\mathcal{O} \subseteq \{\cup, \cap, {}^-\}$, *then* $\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MC}_{\mathbb{N}}(\mathcal{O})$ *and* $\mathrm{EF}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MF}_{\mathbb{N}}(\mathcal{O})$.
3. *If* $\mathcal{O} \subseteq \{+, \times\}$, *then* $\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}_{\mathbb{N}}(\mathcal{O} \cup \{\cap, \times\})$ *and* $\mathrm{EF}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MF}_{\mathbb{N}}(\mathcal{O} \cup \{\cap, \times\})$.
4. *If* $\mathcal{O} \subseteq \{\cap, +, \times\}$, *then* $\mathrm{EC}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MC}_{\mathbb{N}}(\mathcal{O} \cup \{\cap, \times\})$ *and* $\mathrm{EF}_{\mathbb{N}}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MF}_{\mathbb{N}}(\mathcal{O} \cup \{\cap, \times\})$.

In combination with the results by McKenzie and Wagner [5] we obtain the following lower bounds.

Corollary 7. *It holds that*

1. $\mathrm{EC}_{\mathbb{N}}({}^-, \cup, \cap, +, \times)$ *and* $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, +, \times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* NEXP.
2. $\mathrm{EF}_{\mathbb{N}}({}^-, \cup, \cap, +, \times)$, $\mathrm{EC}_{\mathbb{N}}({}^-, \cup, \cap, +)$, $\mathrm{EC}_{\mathbb{N}}({}^-, \cup, \cap, \times)$, $\mathrm{EF}_{\mathbb{N}}({}^-, \cup, \cap, +)$, $\mathrm{EF}_{\mathbb{N}}({}^-, \cup, \cap, \times)$, $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, +)$, $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, \times)$, *and* $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE.
3. $\mathrm{EC}_{\mathbb{N}}({}^-, \cup, \cap)$, $\mathrm{EC}_{\mathbb{N}}(\cup, \cap)$, $\mathrm{EC}_{\mathbb{N}}(\cap, +, \times)$, *and* $\mathrm{EC}_{\mathbb{N}}(+, \times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* P.
4. $\mathrm{EC}_{\mathbb{N}}(\cap, +)$, $\mathrm{EC}_{\mathbb{N}}(\cap, \times)$, *and* $\mathrm{EC}_{\mathbb{N}}(+)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* $\mathrm{C}_{=}\mathrm{L}$.
5. $\mathrm{EC}_{\mathbb{N}}(\cup)$, $\mathrm{EC}_{\mathbb{N}}(\cap)$, *and* $\mathrm{EC}_{\mathbb{N}}(\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* NL.
6. $\mathrm{EF}_{\mathbb{N}}({}^-, \cup, \cap)$, $\mathrm{EF}_{\mathbb{N}}(\cup, \cap)$, $\mathrm{EF}_{\mathbb{N}}(\cup)$, $\mathrm{EF}_{\mathbb{N}}(\cap)$, $\mathrm{EF}_{\mathbb{N}}(\cap, +)$, $\mathrm{EF}_{\mathbb{N}}(\cap, \times)$, $\mathrm{EF}_{\mathbb{N}}(\cap, +, \times)$, $\mathrm{EF}_{\mathbb{N}}(+)$, $\mathrm{EF}_{\mathbb{N}}(\times)$, *and* $\mathrm{EF}_{\mathbb{N}}(+, \times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* L.

4 Feasible Equivalence Problems

In this section, we present several equivalence problems for which we can show that efficient evaluation algorithms exist. While the algorithms in the first part all require deterministic polynomial time or less, randomization is needed in the second part of the section.

4.1 Equivalence Problems Solvable in Polynomial Time

From Proposition 6 and the results by McKenzie and Wagner [5] we obtain the following.

Corollary 8. $\mathrm{EC}_{\mathbb{N}}(\cup,\cap)$, $\mathrm{EC}_{\mathbb{N}}(\cup,\cap,^{-})$, $\mathrm{EC}_{\mathbb{N}}(\cap,\times) \in \mathrm{P}$, $\mathrm{EF}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{DLOGCFL}$, $\mathrm{EC}_{\mathbb{N}}(\cup)$, $\mathrm{EC}_{\mathbb{N}}(\cap) \in \mathrm{NL}$, *and* $\mathrm{EF}_{\mathbb{N}}(\cup)$, $\mathrm{EF}_{\mathbb{N}}(\cap)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\cap)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\cap,^{-})$, $\mathrm{EF}_{\mathbb{N}}(\times)$, $\mathrm{EF}_{\mathbb{N}}(+)$, $\mathrm{EF}_{\mathbb{N}}(\cap,\times) \in \mathrm{L}$.

We now show that $\mathrm{EC}_{\mathbb{N}}(\cap,+)$ is complete for the class $\mathrm{coC}_{=}\mathrm{L}(2)$, which is the complement of the second level of the Boolean hierarchy over $\mathrm{C}_{=}\mathrm{L}$. As a useful tool, we introduce *non-emptiness problems* for circuits.

Definition 9. *We define non-emptiness problems for $\mathcal{O}$-circuits and $\mathcal{O}$-formulas.*

$$\mathrm{NEC}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{C \mid C \text{ is an } \mathcal{O}\text{-circuit such that } C \neq \emptyset\}.$$
$$\mathrm{NEF}_{\mathbb{N}}(\mathcal{O}) \stackrel{df}{=} \{C \mid C \text{ is an } \mathcal{O}\text{-formula such that } C \neq \emptyset\}.$$

Lemma 10. $\mathrm{NEC}_{\mathbb{N}}(\cap,+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\mathrm{C}_{=}\mathrm{L}$ *and* $\mathrm{NEF}_{\mathbb{N}}(\cap,+) \in \mathrm{L}$.

The next proposition shows that in some cases, equivalence problems with addition can very easily be reduced to equivalence problems with multiplication.

Proposition 11. $\mathrm{EC}_{\mathbb{N}}(\cap,+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{N}}(\cap,\times)$ *and* $\mathrm{EC}_{\mathbb{N}}(+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{N}}(\times)$.

Corollary 12. $\mathrm{EF}_{\mathbb{N}}(\cap,+) \in \mathrm{L}$.

Theorem 13. $\mathrm{EC}_{\mathbb{N}}(\cap,+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\mathrm{coC}_{=}\mathrm{L}(2)$.

Corollary 14. $\mathrm{EC}_{\mathbb{N}}(\cap,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* $\mathrm{coC}_{=}\mathrm{L}(2)$, $\mathrm{EC}_{\mathbb{N}}(\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* $\mathrm{C}_{=}\mathrm{L}$.

4.2 Equivalence Problems Solvable by Randomized Algorithms

We explain that $\mathrm{EC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{BPP}$, i.e., decidable in bounded-error probabilistic polynomial time. Schönhage proved [6] that $\mathrm{EC}_{\mathbb{N}}(+,\times) \in \mathrm{coRP}$, and McKenzie and Wagner showed [5] that $\mathrm{MC}_{\mathbb{N}}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{N}}(+,\times)$. So $\mathrm{MC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{coRP}$. By Proposition 6, $\mathrm{EC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{P}^{\mathrm{coRP}}$, and hence $\mathrm{EC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{BPP}$ due to the self-lowness of BPP [3].

Corollary 15. $\mathrm{MC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{coRP}$ *and* $\mathrm{EC}_{\mathbb{N}}(\cap,+,\times) \in \mathrm{BPP}$.

5 Intractable Equivalence Problems

In this section we analyze equivalence problems which are more difficult to decide than the problems presented in the former section. The scope ranges from Π_2^{P}-complete for the more restricted problems like $\mathrm{EF}_{\mathbb{N}}(\cup,+)$ and $\mathrm{EF}_{\mathbb{N}}(\cup,\times)$ up to NEXP-hard for $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$, the most general membership problem we consider. The best upper bound we can give in that case is the Turing-degree of the halting problem.

5.1 Π_2^{P}-Complete Problems

We show Π_2^{P}-completeness for several equivalence problems. Lemma 4 already shows that some of these problems belong to Π_2^{P}, and it is known that $\mathrm{EF}_{\mathbb{N}}(\cup,+)$ is Π_2^{P}-complete [7]. Hence, it suffices to prove Π_2^{P}-hardness for $\mathrm{EF}_{\mathbb{N}}(\cup,\times)$. We first show that the following problem is Π_2^{P}-complete.

$$\mathrm{QPOS}_2 \stackrel{\mathrm{df}}{=} \big\{(x_1,\ldots,x_{2n},b) \,\big|\, x_1,\ldots,x_{2n},b \geq 1 \text{ and } \forall I \subseteq \{1,\ldots,n\} \; \exists J \subseteq \{n{+}1,\ldots,2n\} \; \big(\textstyle\prod_{i\in I} x_i \prod_{j\in J} x_j = b\big)\big\}$$

We then prove that $\mathrm{QPOS}_2 \leq_{\mathrm{m}}^{\log} \mathrm{EF}_{\mathbb{N}}(\cup,\times)$.

Theorem 16. QPOS_2 *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* Π_2^{P}.

Theorem 17. $\mathrm{EF}_{\mathbb{N}}(\cup,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* Π_2^{P}.

Furthermore, we can show that $\mathrm{EC}_{\mathbb{N}}(\cup,\times)$ and $\mathrm{EC}_{\mathbb{N}}(\cup,+)$ are in Π_2^{P} by using standard techniques. We obtain:

Theorem 18. *The following problems are all* $\leq_{\mathrm{m}}^{\log}$*-complete for* Π_2^{P}*:* $\mathrm{EC}_{\mathbb{N}}(\cup,+)$, $\mathrm{EC}_{\mathbb{N}}(\cup,\times)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\cap,+,\times)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\cap,+)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\cap,\times)$, $\mathrm{EF}_{\mathbb{N}}(\cup,+,\times)$, $\mathrm{EF}_{\mathbb{N}}(\cup,+)$, $\mathrm{EF}_{\mathbb{N}}(\cup,\times)$

5.2 More General Equivalence Problems

We analyze the complexity of the most general equivalence problem, $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$. From the Propositions 5 and 6 it follows that $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$.

Every decision algorithm for $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$ would enable us to automatically verify Goldbach's conjecture. This means that we run the algorithm on input of the circuit that formulates Goldbach's conjecture (this circuits is shown in the introduction of this paper) and the algorithm definitely tells us whether or not the conjecture is true.

It is possible that $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$ is undecidable, but at the moment, we cannot prove this. Observe that the problem $\mathrm{EF}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$ shares the same fate: Obviously, a terminating decision procedure for $\mathrm{EF}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$ can also be used to decide $\mathrm{EC}_{\mathbb{N}}(^{-},\cup,\cap,+,\times)$ by simply unfolding the circuit into a (possibly exponentially larger) formula before feeding it to the decision algorithm.

Circuits over positive natural numbers will turn out useful to analyze the most general equivalence problem. Breunig [2] showed that if we restrict the range to $\mathbb{N}^+$ (instead of $\mathbb{N}$), then the general membership problem for circuits is decidable in PSPACE. We will utilize this result to show that we can solve $\mathrm{EC}_{\mathbb{N}}(^-, \cup, \cap, +, \times)$ if the evaluation algorithm has oracle access to the halting problem.

Theorem 19 ([2]). $\mathrm{MC}_{\mathbb{N}^+}(^-, \cup, \cap, +, \times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* PSPACE.

Lemma 20. *There exists an oracle Turing machine* M *with oracle* K *(the halting problem) that on input of a* $\{^-, \cup, \cap, +, \times\}$*-circuit* C *over* $\mathbb{N}$ *outputs a* $\{^-, \cup, \cap, +, \times\}$*-circuit* D *over* $\mathbb{N}^+$ *and a set* $Z \subseteq \{0\}$ *such that* $C = D \cup Z$.

Theorem 21. $\mathrm{MC}_{\mathbb{N}}(^-, \cup, \cap, +, \times)$ *and hence also* $\mathrm{EC}_{\mathbb{N}}(^-, \cup, \cap, +, \times)$ *belong to* $\deg_{\mathrm{T}}(\mathrm{K})$ *where* K *denotes the halting problem.*

Since the general circuits can contain $^-$-gates, the following is easy to see:

Proposition 22. *The problem* $\mathrm{EC}_{\mathbb{N}}(^-, \cup, \cap, +, \times)$ *is recursively-enumerable if and only if* $\mathrm{EC}_{\mathbb{N}}(^-, \cup, \cap, +, \times)$ *is decidable.*

However, the equivalence problems become decidable if we forbid the combination of $^-$-, $+$-, and $\times$-gates.

Proposition 23. *1.* $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, +, \times)$ *and* $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$ *are in* $\mathrm{coNEXP}^{\mathrm{NP}}$.
2. $\mathrm{EC}_{\mathbb{N}}(^-, \cap, \cup, \times), \mathrm{EF}_{\mathbb{N}}(^-, \cap, \cup, \times), \mathrm{EC}_{\mathbb{N}}(\cap, \cup, \times) \in \mathrm{PSPACE}$

Testing equivalence for $\{\cup, +, \times\}$-circuits is likely to be more difficult than testing membership: While the membership problem is PSPACE-complete [5], we can show that the equivalence problem is NEXPTIME-hard. For the membership problem, it suffices to compute numbers in the length of the target number, as numbers can only become smaller when multiplied by 0. Intuitively, the difficulty when testing equivalence for $\{\cup, +, \times\}$-circuits is that we have to deal with very large (up to exponential in length) numbers.

It was observed [9] that a similar effect occurs when testing membership for $\{\cup, +, \times\}$-circuits over the *integers*: By describing a generic reduction, it was shown that $\mathrm{MC}_{\mathbb{Z}}(\cup, +, \times)$ is NEXP-complete [9]. An analysis of that involved proof yields that with minor modifications, it does also show that $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NEXP.

Corollary 24. *The problem* $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* NEXP.

6 Conclusions

In Table 1 we summarize upper and lower bounds for $\mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ and $\mathrm{EF}_{\mathbb{N}}(\mathcal{O})$ for different sets of operations. In general, equivalence problems are more difficult to solve than their corresponding membership problems. Two circuits are equivalent if for all natural numbers x, they coincide with respect to membership of x. The

Table 1. Upper and lower bounds for $\mathrm{EC}_{\mathbb{N}}(\mathcal{O})$ and $\mathrm{EF}_{\mathbb{N}}(\mathcal{O})$. All lower bounds are with respect to $\leq_{\mathrm{m}}^{\log}$-reductions and the numbers in parentheses refer to the corresponding theorems (T), corollaries (C), or lemmas (L). The PSPACE-completeness of $\mathrm{EF}_{\mathbb{N}}(^{-}, \cup, \cap, +)$ and the Π_2^{P}-completeness of $\mathrm{EF}_{\mathbb{N}}(\cup, +)$ were shown by Stockmeyer and Meyer [7]. The $\mathrm{C}_{=}\mathrm{L}$-completeness of $\mathrm{EC}_{\mathbb{N}}(+)$ was shown by McKenzie and Wagner [5]. $\mathrm{EC}_{\mathbb{N}}(+, \times) \in \mathrm{coRP}$ was shown by Schönhage [6]. For $\mathrm{EC}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{EF}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ we only have $\deg_{\mathrm{T}}(\mathrm{K})$, the Turing degree of the halting problem, as upper bound. It is possible that these problems are undecidable.

	$\mathrm{EC}_{\mathbb{N}}$		$\mathrm{EF}_{\mathbb{N}}$	
$\mathcal{O}$	Lower Bound	Upper Bound	Lower Bound	Upper Bound
$^{-}\ \cup\ \cap\ +\ \times$	NEXP (C7)	$\deg_{\mathrm{T}}(\mathrm{K})$ (T21)	PSPACE (C7)	$\deg_{\mathrm{T}}(\mathrm{K})$ (T21)
$^{-}\ \cup\ \cap\ +$	PSPACE (C7)	PSPACE (L4)	PSPACE (T3)	PSPACE (T3)
$^{-}\ \cup\ \cap\ \ \times$	PSPACE (C7)	PSPACE (P23)	PSPACE (C7)	PSPACE (P23)
$^{-}\ \cup\ \cap$	P (C7)	P (C8)	L (C7)	L (C8)
$\cup\ \cap\ +\ \times$	NEXP (C7)	$\mathrm{coNEXP}^{\mathrm{NP}}$ (L23)	Π_2^{P} (T18)	Π_2^{P} (T18)
$\cup\ \cap\ +$	PSPACE (C7)	PSPACE (L4)	Π_2^{P} (T18)	Π_2^{P} (T18)
$\cup\ \cap\ \ \times$	PSPACE (C7)	PSPACE (P23)	Π_2^{P} (T18)	Π_2^{P} (T18)
$\cup\ \cap$	P (C7)	P (C8)	L (C7)	L (C8)
$\cup\ \ +\ \times$	NEXP (C24)	$\mathrm{coNEXP}^{\mathrm{NP}}$ (L23)	Π_2^{P} (T18)	Π_2^{P} (T18)
$\cup\ \ +$	Π_2^{P} (T18)	Π_2^{P} (T18)	Π_2^{P} (T3)	Π_2^{P} (T3)
$\cup\ \ \ \times$	Π_2^{P} (T18)	Π_2^{P} (T18)	Π_2^{P} (T18)	Π_2^{P} (T18)
$\cup$	NL (C7)	NL (C8)	L (C7)	L (C8)
$\cap\ +\ \times$	P (C7)	BPP (C15)	L (C7)	DLOGCFL (C8)
$\cap\ +$	$\mathrm{coC}_{=}\mathrm{L}(2)$ (T13)	$\mathrm{coC}_{=}\mathrm{L}(2)$ (T13)	L (C7)	L (C12)
$\cap\ \ \times$	$\mathrm{coC}_{=}\mathrm{L}(2)$ (C14)	P (C8)	L (C7)	L (C8)
$\cap$	NL (C7)	NL (C8)	L (C7)	L (C8)
$\ \ +\ \times$	P (C7)	coRP (T3)	L (C7)	DLOGCFL (C8)
$\ \ +$	$\mathrm{C}_{=}\mathrm{L}$ (T3)	$\mathrm{C}_{=}\mathrm{L}$ (T3)	L (C7)	L (C8)
$\ \ \ \times$	$\mathrm{C}_{=}\mathrm{L}$ (C14)	P (C8)	L (C7)	L (C8)

difference between equivalence and membership becomes even more apparent if one realizes that in general, this universal quantifier is not polynomially bounded in the length of the circuits. For example, a circuit that contains $\times$-gates can produce numbers of exponential length in its output. An equivalence test has to make sure that two given circuits of size n agree with respect to membership of all x such that $|x| \leq 2^{2^n}$. In some cases (e.g., $\mathrm{EC}_{\mathbb{N}}(\cup, \times)$) it is possible to condense the search space such that one ends at a polynomially-bounded universal quantifier. In other cases (e.g., $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$) such a polynomial bound cannot be established. We leave open whether the bounds for $\mathrm{EC}_{\mathbb{N}}(\cup, +, \times)$ and $\mathrm{EC}_{\mathbb{N}}(\cup, \cap, +, \times)$ can be improved.

The most general case we consider is the equivalence problem for $\{^{-}, \cup, \cap, +, \times\}$-circuits. As discussed in the introduction, Goldbach's conjecture can be formulated as such an equivalence problem. This explains why we were not able to find decidable upper bounds for $\mathrm{EC}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{EF}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$.

However, with the Turing-degree of the halting problem we identify at least one non-trivial upper bound. This yields the first non-trivial upper bound for $\mathrm{MC}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{MF}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$. It is possible that $\mathrm{EC}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{EF}_{\mathbb{N}}(^{-}, \cup, \cap, +, \times)$ are undecidable, but so far the best provable lower bound is NEXP. We leave as our most challenging open question whether these problems are decidable. Here we only know that they are either decidable or not recursively enumerable.

When comparing the complexities of membership problems with their corresponding equivalence problems, we notice that usually the complexity either stays the same or increases significantly because of the earlier discussed universal quantifier. When looking at the equivalence problem for $\{\cap, +\}$-circuits we observe a completely different behavior. While the complexity of $\mathrm{MC}_{\mathbb{N}}(\cap, +)$ is $\mathrm{C}_{=}\mathrm{L}$, the complexity of $\mathrm{EC}_{\mathbb{N}}(\cap, +)$ is $\mathrm{coC}_{=}\mathrm{L}(2)$, which is the complement of the second level of the Boolean hierarchy over $\mathrm{C}_{=}\mathrm{L}$. So here we observe a moderate jump of the complexity. For the related problem $\mathrm{EC}_{\mathbb{N}}(\cap, \times)$ we obtain $\mathrm{coC}_{=}\mathrm{L}(2)$ as lower bound, but we leave open whether this is also an upper bound. Similarly, we would like to know matching bounds for $\mathrm{EC}_{\mathbb{N}}(\times)$.

Acknowledgements

The authors are grateful to Thomas Geier, Daniel Meister, and Marco Nehmeier for interesting discussions and various valuable hints.

References

1. Allender, E.: Making computation count: Arithmetic circuits in the nineties. SIGACT NEWS 28(4), 2–15 (1997)
2. Breunig, H.: The complexity of membership problems for circuits over sets of positive numbers. In: Proceedings 16th International Symposium on Fundamentals of Computation Theory. LNCS, Springer, Heidelberg 2007 (to appear)
3. Ko, K.: Some observations on the probabilistic algorithms and NP-hard problems. Information Processing Letters 14(1), 39–43 (1982)
4. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential time. In: Proceedings 13th Symposium on Switching and Automata Theory, pp. 125–129. IEEE Computer Society Press, Los Alamitos (1972)
5. McKenzie, P., Wagner, K.W.: The complexity of membership problems for circuits over sets of natural numbers. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 571–582. Springer, Heidelberg (2003)
6. Schönhage, A.: On the power of random access machines. In: ICALP, pp. 520–529 (1979)
7. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proceedings 5th ACM Symposium on the Theory of Computing, pp. 1–9. ACM Press, New York (1973)
8. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. Journal of the ACM 25(3), 405–414 (1978)

9. Travers, S.: The complexity of membership problems for circuits over sets of integers. Theoretical Computer Science 369(1), 211–229 (2006)
10. Wagner, K.: The complexity of problems concerning graphs with regularities. In: Proceedings Mathematical Foundations of Computer Science. LNCS, vol. 176, pp. 544–552. Springer, Heidelberg (1984)
11. Wagner, K.W., Wechsung, G.: On the boolean closure of NP. In: Budach, L. (ed.) FCT 1985. LNCS, vol. 199, pp. 485–493. Springer, Heidelberg (1985)
12. Yang, K.: Integer circuit evaluation is PSPACE-complete. In: IEEE Conference on Computational Complexity, pp. 204–213. IEEE Computer Society Press, Los Alamitos (2000)

Bouillon: A Wiki-Wiki Social Web

Victor Grishchenko

Institute of Physics and Applied Mathematics
Ural State University
51 Lenina st.
Yekaterinburg, Russia
gritzko@ural.ru

Abstract. The Bouillon project implements the vision termed a "Social Web". It is an extremely open collaboration environment employing social links for creation, filtering and dissemination of information. It is also an attempt to boost the wiki effect of knowledge crystallization.

Currently, the project is in stage of public testing, available at *http://oc-co.org*.

1 Introduction

1.1 General Theory

Many limitations of current online environments have their roots in the problem of trust. The author's general theory is that every information exchange environment has two key parameters: openness and (quality) control. Those parameters have to be balanced. Over-controlled environment has problems accommodating new information and thus become poor. Too open environment becomes polluted. These problems are generic and exist independently of a particular medium (paper, voice or bytes). This simple theory is further illustrated with some examples. Although openness and control seem to be contradicting objectives, the author's point is that more advanced environments combine more openness and more control. E.g. Wikipedia's success compared to Britannica [15] may be explained by the former being much more open while still keeping the level of control comparable to the latter. The objective of the Bouillon project is to create an online environment having a bit more of openness *and* control than those currently existing.

1.2 On Wiki Scalability

Original WikiWikiWeb [2] ideas targeted minimum-effort collaboration leading to information "automagically" crystallizing. Wiki is supposed to be a single medium for reading, collaborative editing, discussions and conversations. In many cases, "crystallization" appeared to be a more effective way of putting information together than the "glue" that search engines provide for WWW. Recognition of the wiki way's advantages is a Wikipedia article being the first

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 139–145, 2007.

entry for many Google searches on appropriate topics (if you still have no habit of checking Wikipedia first).

Do wiki face any challenges? Or, what might be more open than a wiki?

Any information environment have faced the scalability challenge; while some crises lead to the environment's decay (such as the Usenet's Eternal September), other crises were answered by a major breakthrough (such as Google's PageRank [23]). Wikis are increasingly popular; Wikipedia accommodates exponentially growing content [5] without any visible degradation of its quality.

Obviously, extreme openness (any visitor can edit) leads to regular abuses by spambots and vandals. Many of today's smaller wikis have to employ user accounts and passwords. This shifts their utility closer to ordinary web pages. Wikipedia, as an Internet-scale project, has some advantages in this regard. Namely, Wikipedia may have exhaustive statistics on strangers' behavior. The great attention resource of Wikipedia gives a unique possibility to remove vandalism and spam by manual labor. Still, the scope of the project is limited to encyclopedic factual information. Working with personal opinions, experiences and discussions, commercial or local information, manuals and guides etc etc are all beyond possibilities of Wikipedia's technology.

So, wiki openness and scalability have visible limits.

2 State of the Field

2.1 Social ∩ Web

One path for improving wiki scalability was proposed by the original wiki inventor Ward Cunningham [14] in 1997. The "folk memory" approach assumes a more social, peer-to-peer way of new material creation, filtration and dissemination – much like "folk tales or folk songs are remembered and propagated within a culture". Indeed, if it works in the real world, in real social networks, why wouldn't it work automatically online?

Another concept, the "Social Web", although rather fuzzy, seems relevant to our case. Wikipedia defines Social Web as "an open global distributed data sharing network" that "links people, organizations, and concepts". One attempted step in practical implementation of the Social Web is Augmented Social Network Initiative [18]. The initiative took some effort in introducing online social interaction standards based on OASIS XDI, but produced no usable outcome.

One more relevant ongoing effort is the NEPOMUK project (Networked Environment for Personalized, Ontology-based Management of Unified Knowledge [6]) which is a part of the Semantic Web movement. Wiki-related activities of NEPOMUK include creation and deployment of semantically annotated wikis [19].

2.2 Topology

This work follows the Folk memory vision of information propagation in a social network. Because of this, topological features of social networks are extremely important for this work. What do we know about social topology?

One recent breakthrough in this area was associated with the scale-free networks theory [10]. The most characteristic feature of scale-free networks is power-law distribution of node degree $P(k) \sim k^{-\gamma}, 2 < \gamma < 3$, where $P(k)$ is a probability of node degree being equal to k. Two results on scale-free networks are especially relevant in the context of this paper.

The first result is that scale-free networks have no epidemic threshold. Less prevalent memes (genes, pathogens, rumors) do not die out exponentially fast, albeit they are present in smaller populations [11]. An immediate conclusion is that even some less-demanded information will travel through a friend-to-friend network of such topology. To put it simply: long-tail content is welcomed.

Another result is that scale-free networks are ultrasmall. A scale-free network has a diameter of $\sim \log \log N$, where N is the number of nodes [13]. This fact is informally known as the *six degrees of separation* hypothesis, originally attributed to S. Milgram. Ultrasmall diameter guarantees that a network may undergo exponential growth without significant topology changes, i.e. it is of ultimate scalability. Also, this gives us good information reachability guarantees, see Sec. 3.1.

2.3 Related Research

Some recent advances are not directly related to the subject of this paper, but heavily intersect with it, so it is worth to be mentioned briefly.

Some previous work was focused on DHT-style distributed wiki hosting [25], that is supposed to change the technical aspect leaving collaboration aspects of wiki(pedia) intact. Steps were taken by different entities to introduce some Wiki Interchange Format, mostly for offline wiki synchronization, although the author is unaware of any widely accepted standard in this area.

There are numerous commercial and open-source collaborative real-time editors letting a limited number of participants to craft a common document. Notable products of this kind include SubEthaEdit [7] for Mac OS X, open-source editor ACE [1] and others. A web-based collaborative editor SynchroEdit [8] may also be used as a wiki editor.

During last five years, the internet has shown explosive growth of online social networking sites, such as LinkedIn, LiveJournal or MySpace.

Extensive work was undertaken to optimize peer-to-peer query flood algorithms [12,26,20]. That approach was originally used in the Gnutella P2P network [4], as well as in some later projects, and it is considered to be computationally inefficient. The main concern was exponential growth of the amount of queries compared to the number of users [24]. Bouillon's objective is to turn this exponential growth weakness into a strength, see Sec. 3.1. Sec. 3.2 specially addresses complexity issues of Bouillon's use of query flood.

3 Bouillon

Bouillon project aims to create a real-time WYSIWYG wiki without any distinction between reading and editing modes. The extreme openness is balanced

by the employed information filtering and dissemination process: any changes and opinions propagate from friend to friend, so content is sieved by the social network. This is much along the Folk Memory and Social Web visions mentioned earlier.

As in any wiki, every Bouillon page is identified by its name. Every Bouillon page is a tree of *pieces*. To better visualize pieces, one may think of sections, subsections and paragraphs of an article and their parent–child relations. Technically, Bouillon deals with a coarsened DOM tree [3] of an XML document. Every piece has a partially ordered set of versions. To choose among different versions and to glue separate pieces into a tree Bouillon employs *opinions*. Opinions can be of two kinds: either "a/b" claiming relevance of piece b as a child of piece a, or "$b : v$" claiming relevance of particular version v of a piece a. Opinions could be both positive and negative.

As it was mentioned, any information, opinions, requests and pieces are passed from friend to a friend; the network has social topology. Forming links (contacts) are weighted with reputation in both directions, $\rho_a(b) \in [0; 1]$. Reputations are supposed to be auto calculated based on past performance/compliance of one respective peer in the eyes of another. All opinions are weighted according to reputation distance to the author, where "reputation distance" stands for multiplicative weight of a path from the opinion's source to the current peer. Opinion calculations are made according to a formal model based on fuzzy logic which is explained in [17] and in more detail in [27].

To retrieve opinions from the social network peers use *requests* of two types. Root request $/a$ asks for anyone who knows about the piece a; child request $a/$ retrieves a/b and $a : v$ kind of opinions. Correspondingly, opinion retrieval goes in two phases. Root request floods vicinity of a peer to detect every promising direction where something is known on the target page. Such a flood covers just a sublinear amount of nodes (compared to the overall size of the network). The following recursive retrieval is destined to promising directions only; as the process advances deeper into the tree, the list of promising directions shrinks, until the whole tree of reachable material is retrieved.

After all relevant opinions are retrieved, the client assembles the page using the most recommended/fresh versions of accepted pieces and finally shows it to the user. Any further edits introduced by the user are committed as opinions and new versions of pieces. That edits immediately become available to the author's social vicinity. On silent approval by any other reading peer, new changes propagate further by the social network.

3.1 Reachability

It is not obvious whether information will propagate quick in such a network or the process will get stuck e.g. the network may separate into islands of incompatible changes. There is a simple topological result:

Lemma 1. *IF an average peer is able to serve its sublinear neighborhood (i.e. N^a nodes, assuming N to be the size of the network) THEN any information or change that has propagated to a corresponding sublinear proportion of nodes*

(N^{1-a}) is now broadly accessible (i.e. the majority of peers may retrieve it by requests).

Proof. (For Erdos-Renyi random graphs). Both nodes storing the information and nodes covered by the request are assumed to be random node subsets of size $O(N^{1-a})$ and $O(N^a)$ respectively. Thus, size of the intersection of two sets has an order of $pqN = \frac{N^{1-a}}{N} \times \frac{N^a}{N} \times N = 1$, where p and q are probabilities for a node to belong to the respective set. (Actually, we get a Bernoulli trial having success probability pq.) Mean intersection size of 1 may seem unreliable; this is easily fixed by multiplying size of either set by some constant factor c. Deviations from the mean follow binomial distribution, so this gives us success (request meets the information) in most of the cases. □

For scale-free graphs, the situation is even better. In a scale-free graph, highly-connected nodes (hubs) appear early in a vicinity ball growing from some center [13]. So, both sets of nodes that store the information and those covered by the request are supposed to heavily intersect with a third set of hubs (which also has sublinear size [16]). This boosts the effect described in the previous lemma.

What about information that gained no sublinear "prevalence"? First, most of information is of local value and is not supposed to spread globally. People's interests are supposed to correlate with social links, so such local information will likely reach all of its audience without disturbing the rest.

Because of the absence of epidemic threshold (Sec. 2.2) it is generally expected that no information would die out if it is of some interest to anyone. The six degrees of separation hypothesis contributes another optimistic expectation. If a vicinity ball covered by requests has a radius of two steps (degrees), then from any point of humankind to any other point a piece of information might ideally be relayed by two intermediate evaluations.

3.2 Computational Load

The Bouillon protocol has some resource-hungry features. First, the initial root request (vicinity flood) that must cover some significant sublinear proportion of nodes. Second, a separate request corresponds to each piece of a page. Third, each request has to be refreshed every 50 seconds to support real-time change notification feature. (The interval of 50 seconds is an empirical tradeoff between the need to refresh expired and to store unexpired requests).

Let's estimate bandwidth requirements for a peer in a network of 10^6 nodes, assuming node vicinity to be 10^3 nodes, average message size to be $100 - 200$ bytes, $10 - 20$ bytes compressed (high compress ratio is explained by repetitiveness of the XML format). As tokens stream by long-living TCP connections, network overhead is skipped. Average page size is 100 pieces and all peers are simultaneously reading different pages (no aggregation is possible). All peers reside at different hosts, no traffic is local. In this worst-case scenario, an average node has to process an order of $10^3 \times 100$ requests every 50 seconds, i.e. $2 * 10^3$ messages per second or $4 * 10^4$ bytes per second. Finally, we get $320 Kbit/s$, less than a typical BitTorrent download speed.

We may also suppose that a high clustering coefficient [21,22] of social networks may lead to unnecessary message duplication. (Clustering coefficient is the probability of two random friends of a random node being friends to each other). Let the clustering coefficient be as high as 0.9. Simplistically, this leads to the effect of 90% requests being passed to a node that already has a copy, so to cover 10^3 nodes 10^4 request copies are needed. Thus, the pessimistically estimated bandwidth consumption jumps to 3.2 $Mbit/s$, which is still has an order of an average ADSL connection.

(Needless to say, some datacenter-based deployment variants are always possible, users do not spend online 24 hours a day, some users read the same pages, some peers know nothing about a page, so they get no requests except for the root request, also clustering effect is less relevant for the outer layer of a vicinity ball, which is the most of the ball's mass, etc etc).

4 Conclusion

A wiki-wiki prototype of a Social-Web was implemented in ∼2000 lines of Java code for the engine plus ∼400 lines of original JavaScript code for the client (WYSIWYG editing is implemented by TinyMCE [9]). The technology is expected to scale very well, combining extreme openness and reasonable quality control for massively distributed collaboration. The prototype is always available at *http://oc-co.org*.

References

1. Ace collaborative text editor: http://ace.sf.net
2. Article on wikis at c2.com: http://c2.com/cgi/wiki?WikiWikiWeb
3. Document object model (dom): http://www.w3.org/DOM/
4. The gnutella protocol specification v0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
5. Modelling wikipedia's growth: http://en.wikipedia.org/wiki/Wikipedia:Modelling_Wikipedia's_growth
6. Nepomuk, the social semantic desktop project homepage: nepomuk.semanticdesktop.org/
7. Subethaedit product page: http://www.codingmonkeys.de/subethaedit/
8. Synchroedit collaborative editor product page: http://www.synchroedit.com/
9. Tinymce javascript content editor: http://tinymce.moxiecode.com
10. Barabasi, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
11. Boguna, M., Pastor-Satorras, R., Vespignani, A.: Absence of epidemic threshold in scale-free networks with connectivity correlations (2002), http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0208163
12. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable (2003)
13. Cohen, R., Havlin, S.: Scale-free networks are ultrasmall. Phys. Rev. Lett. 90, 058701 (2003)

14. Cunningham, W.: Folk memory: A minimalist architecture for adaptive federation of object servers (1997), http://c2.com/doc/FolkMemory.pdf
15. Giles, J.: Internet encyclopaedias go head to head. Nature 438, 900–901 (2005)
16. Grishchenko, V.: Computational complexity of one reputation metric. In: Proceedings of IEEE/Create-Net SecureComm 2005 workshops, Athens, Greece (2005) (ISBN 0-7803-9469-0)
17. Grishchenko, V.: Redefining web-of-trust: reputation, recommendations, responsibility and trust among peers. In: Proceedings of the First Workshop on Friend of a Friend, Social Networking and the Semantic Web, National University of Ireland, Galway, pp. 75–84 (September 2004)
18. Jordan, K., Hauser, J., Foster, S.: The augmented social network: Building identity and trust into the next-generation internet (2003), http://www.firstmonday.dk/issues/issue8_8/jordan/
19. Kotelnikov, M., Polonsky, A., Kiesel, M., Volkel, M., Sogrin, M., et al.: Nepomuk. interactive semantic wikis (2007), http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main1/D1-1
20. Lin, T., Wang, H., Wang, J.: Search performance analysis and robust search algorithm in unstructured peer-to-peer networks, pp. 346–354 (2004)
21. Newman, M.E.J.: Scientific collaboration networks. I. network construction and fundamental results. Phys. Rev. E 64(1) (2001)
22. Newman, M.E.J.: Scientific collaboration networks. II. clustering and preferential attachment in growing networks. Phys. Rev. E 64(2) (2001)
23. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)
24. Ritter, J.: Why gnutella can't scale. no, really (2001), http://www.darkridge.com/jpr5/doc/gnutella.html
25. Urdaneta, G., Pierre, G., van Steen, M.: A decentralized wiki engine for collaborative wikipedia hosting. In: Proceedings of the 3rd International Conference on Web Information Systems and Technologies (March 2007), http://www.globule.org/publi/DWECWH_webist2007.html
26. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer networks. pp. 5–14 (2002)
27. Грищенко, В.С.: Исчисление мнений. *Известия Уральского государственного университета. Серия Компьютерные науки и информационные технологии*, 43, 139–153 (2006)

A PDL-Like Logic of Knowledge Acquisition

Bernhard Heinemann

Fakultät für Mathematik und Informatik,
FernUniversität in Hagen,
58084 Hagen, Germany
bernhard.heinemann@fernuni-hagen.de

Abstract. Starting off with Moss and Parikh's view of knowledge we develop a kind of dynamic logic of knowledge acquisition. Corresponding procedures are basically modelled by functions changing the knowledge states of the agent under discussion, and the machinery of deterministic PDL is then utilized for knowledge-based programming. The main issues of the paper are the fundamental meta-theorems on the arising logic, KAL. We prove, in particular, the soundness and completeness with respect to the intended class of structures as well as the decidability of KAL.

Keywords: logics of knowledge, knowledge acquisition, propositional dynamic logic, topological reasoning.

1 Introduction

The approach undertaken here relies on the particular logic of knowledge set out in the papers [1] and [2], respectively. That system, $\mathcal{L}$, emphasizes the interrelation between knowledge and topology. We briefly recall the basics of $\mathcal{L}$ at the beginning of this paper. In $\mathcal{L}$, the knowledge of an agent in question is represented by the space of all *knowledge states*. These are the sets of states the agent considers possible at a time. If an effort is made to acquire knowledge, then this appears as a *shrinking procedure* regarding that space of sets. The formulas of the language underlying $\mathcal{L}$ may contain both a modality K describing knowledge and an operator $\Box$ expressing effort. The semantic domains are triples $(X, \mathcal{O}, V)$ called *subset spaces,* which consist of a non-empty set X of states, a set $\mathcal{O}$ of subsets of X representing the knowledge states of the agent (sometimes called *the opens*), and a valuation V determining the states where the atomic propositions are true. The operator K then quantifies over some knowledge state $U \in \mathcal{O}$, whereas $\Box$ quantifies 'downward' over $\mathcal{O}$ since shrinking elements of $\mathcal{O}$ and gaining knowledge correspond to each other.

The fact that $\Box$ implicitly models some knowledge acquisition procedure is the starting point to the following. What we do below is making such procedures *explicit*. To this end, we enrich subset spaces with partial functions operating on $\mathcal{O}$. It is intended that these functions represent the 'elementary' procedures of that kind; the more complex ones then come into play by means of the program constructs known from PDL; cf, eg, [3], § 10.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 146–157, 2007.

Epistemic dynamic logic is an active field of research, and several relevant systems were proposed in the literature; see, eg, [4,5,6]. The present approach differs from most of these to the effect that it allows a topological interpretation of knowledge acquisition. Actually, corresponding procedures may be viewed as computable approximations in certain spaces of sets, in particular, topological ones. Thus, by modelling knowledge acquisition in the way indicated above the basis of formal topological reasoning is widened at the same time.

The subsequent technical part of the paper is organized as follows. In the next section, we define the new language describing knowledge acquisition. We give also some examples concerning expressiveness there. In Section 3, we turn to the arising logic, which we call KAL (Knowledge Acquisition Logic). We prove the soundness and completeness of KAL with respect to the class of all appropriately enriched subset spaces. After that, we show that KAL is decidable, and we touch on complexity questions. Concluding the paper, we summarize and point to future research.

2 The Language

We now introduce the language underlying KAL and give some sample specifications. Throughout this paper, we confine ourselves to the single-agent case.

We first define the syntax. Let PROP $= \{A, B, \ldots\}$ be a denumerable set of symbols called *proposition letters,* and let $\mathcal{F} = \{F, G, \ldots\}$ be a set of one-place function symbols. The set KAP of *(knowledge acquisition) procedures* and the set $\mathfrak{F}$ of all *formulas* over PROP $\cup\, \mathcal{F}$ are simultaneously defined by the rules

$$P ::= F \mid P;Q \mid P \cup Q \mid P^* \mid K\alpha? \quad \text{and} \quad \alpha ::= A \mid \neg\alpha \mid \alpha \wedge \beta \mid \mathsf{K}\alpha \mid [P]\alpha,$$

respectively, where the letter K has to be substituted in a way that will be explained in a moment. The missing boolean connectives are treated as abbreviations, as needed. The duals of the modal operators are indicated by putting the corresponding letters in angle brackets; thus $\langle\mathsf{K}\rangle$ denotes the dual of K and $\langle P\rangle$ denotes the dual of $[P]$. Now, $K = \mathsf{K}$ or $K = \langle\mathsf{K}\rangle$. Thus tests are restricted to knowledge formulas here.[1]

Secondly, we define the semantics. For a start, we fix the relevant domains. We let $\mathcal{P}(X)$ designate the powerset of a given set X.

Definition 1 (Subset frames and subset spaces with procedures)

1. *A* subset frame with procedures *is a triple* $\mathcal{S} := (X, \mathcal{O}, \{R_P \mid P \in \text{KAP}\})$ *such that*
   (a) X *is a non-empty set,*
   (b) $\mathcal{O} \subseteq \mathcal{P}(X)$ *is a set of subsets of* X*, and*
   (c) *for every* $P \in$ KAP*,* $R_P \subseteq \mathcal{O} \times \mathcal{O}$ *is a binary relation satisfying* $U' \subseteq U$ *if* $U\, R_P\, U'$*, for all* $U, U' \in \mathcal{O}$*.*

[1] In this place, we could be a bit more general by allowing boolean combinations of formulas prefixed by K. However, this is not important to the purposes of this paper.

2. *Let* $\mathcal{S} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\})$ *be a subset frame with procedures. The* set of neighbourhood situations of $\mathcal{S}$ *is* $\mathcal{N}_\mathcal{S} := \{(x, U) \mid x \in U \text{ and } U \in \mathcal{O}\}$.
3. *Let* $\mathcal{S} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\})$ *be a subset frame with procedures and* $V : \mathrm{PROP} \longrightarrow \mathcal{P}(X)$ *be a mapping. Then* V *is called an* $\mathcal{S}$–valuation.
4. *Let* $\mathcal{S} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\})$ *be a subset frame with procedures and* V *be an* $\mathcal{S}$*–valuation. Then,* $\mathcal{M} := (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\}, V)$ *is called a* subset space with procedures *(or, in short, an* SSP*). In this case we say that* $\mathcal{M}$ *is* based on $\mathcal{S}$.

Note that the requirement '$U' \subseteq U$ if $U\, R_P\, U'$' in item 1 (c) of this definition is to reflect knowledge acquisition. Furthermore, unlike the case of usual PDL the relations hold between sets of states.

Now, let an SSP $\mathcal{M}$ be given. We define the relation of satisfaction, $\models_\mathcal{M}$, between neighbourhood situations of the underlying frame and formulas from $\mathfrak{F}$. In the following, neighbourhood situations are written without brackets.

Definition 2 (Satisfaction; validity). *Let* $\mathcal{M} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\}, V)$ *be an SSP based on* $\mathcal{S} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\})$, *and let* $x, U \in \mathcal{N}_\mathcal{S}$ *be a neighbourhood situation. Then*

$$
\begin{aligned}
x, U \models_\mathcal{M} A &\iff x \in V(A)\\
x, U \models_\mathcal{M} \neg\alpha &\iff x, U \not\models_\mathcal{M} \alpha\\
x, U \models_\mathcal{M} \alpha \wedge \beta &\iff x, U \models_\mathcal{M} \alpha \text{ and } x, U \models_\mathcal{M} \beta\\
x, U \models_\mathcal{M} \mathsf{K}\alpha &\iff \text{for all } y \in U : y, U \models_\mathcal{M} \alpha\\
x, U \models_\mathcal{M} [P]\alpha &\iff \forall\, U' \in \mathcal{O} : (\text{if } U\, R_P\, U' \text{ and } x \in U', \text{ then } x, U' \models_\mathcal{M} \alpha),
\end{aligned}
$$

for all $A \in \mathrm{PROP}$, $P \in \mathrm{KAP}$, *and* $\alpha, \beta \in \mathfrak{F}$. *In case* $x, U \models_\mathcal{M} \alpha$ *is true we say that* α holds in $\mathcal{M}$ at *the neighbourhood situation* x, U. *Furthermore, a formula* α *is called* valid in $\mathcal{M}$ *iff it holds in* $\mathcal{M}$ *at every neighbourhood situation. (Manner of writing:* $\mathcal{M} \models \alpha$.*)*

Note that the meaning of proposition letters is independent of neighbourhoods by definition, thus 'stable' with respect to $[P]$. This fact is reflected by a special axiom later on; see Sec. 3.

The intended domains are certain special SSPs rather than arbitrary ones. These are introduced in the next definition.

Definition 3 (Standard SSPs). *Let* $\mathcal{M} = (X, \mathcal{O}, \{R_P \mid P \in \mathrm{KAP}\}, V)$ *be an SSP.* $\mathcal{M}$ *is called* standard *iff the following conditions are satisfied for all* $F \in \mathcal{F}$, $P, Q \in \mathrm{KAP}$ *and* $\alpha \in \mathfrak{F}$:

1. R_F *is a partial function,* $R_F : \mathcal{O} \longrightarrow \mathcal{O}$,
2. $R_{P;Q} = R_P \circ R_Q$,
3. $R_{P \cup Q} = R_P \cup R_Q$,
4. $R_{P^*} = (R_P)^*$, *and*
5. *for all* $U, U' \in \mathcal{O}$, *it holds that* $U\, R_{K\alpha?}\, U'$ *iff* $U' = U$ *and* $y, U \models_\mathcal{M} K\alpha$ *for some* $y \in U$.

Note that the '*' on the right-hand side of the equation in item 4 of this definition denotes the reflexive and transitive closure.

The reader will easily convince himself or herself that every standard SSP is in fact an SSP, i.e., satisfies the requirement in item 1 (c) of Definition 1.

The semantics of the knowledge acquisition procedures defined above and the one of the knowledge-based programs considered in [7], Sec. 7, turn out to be fairly different. While the latter is based on a fine-grained description of the interplay between knowledge and action in multi-agent scenarios, we confine ourselves to simplified terms by allowing procedures to transform entire knowledge states. In this way, we obtain an easily comprehensible system which might be sufficient for simple specification tasks.

Concluding this section we present two examples. The first one contains some program constructs involving the (non-)knowledge of the agent.

Example 1. A program expressing 'IF the agent knows α THEN P ELSE Q' reads $(\mathsf{K}\alpha?; P) \cup (\langle\mathsf{K}\rangle\neg\alpha?; Q)$. Similarly, 'WHILE the agent does not know α DO P' can be expressed by $(\langle\mathsf{K}\rangle\neg\alpha?; P)^*; \mathsf{K}\alpha?$ (the outer brackets are omitted).

The second example deals with the topological interpretation of the new language.

Example 2. During this example we assume that the $\Box$–operator from $\mathcal{L}$ is available, too. Moreover, we let the underlying subset frame $(X, \mathcal{O})$ be a topological space. – Let $P \in \mathrm{KAP}$. Then we have that $\Box\alpha \to [P^*]\,\alpha$ is a validity, for all $\alpha \in \mathfrak{F}$; cf Definition 1.1(c).[2] Now, a 'weak converse' of this schema is imposed, viz $[P^*]\,\alpha \to \Box\Diamond\alpha$. According to Definition 2, this means that the iterated computation of P eventually leads to arbitrarily small neighbourhoods. Therefore, since P^* step-by-step yields smaller and smaller opens we may say that effective approximations in topological spaces can be specified by formulas of the language up to a certain degree.

Example 2 builds a bridge to so-called *dynamic topological logics* in a sense, which were designed, among other things, for modelling hybrid systems; see, eg, [8] and, more recently, [9].

3 The Logic

In this section, we first propose an axiomatization of the logic KAL. After that we sketch the proof of the soundness and semantic completeness of KAL with respect to the class of all SSPs. This takes up most of the time expended here. At the end of this section, we establish the decidability of KAL and comment on its complexity.

The axioms are divided into three groups. To begin with, we list the axioms for the knowledge modality K.

[2] In fact, $\Box\alpha \to [P]\,\alpha$ is globally true for *all* procedures $P \in \mathrm{KAP}$.

1. All instances of propositional tautologies.
2. $\mathsf{K}(\alpha \to \beta) \to (\mathsf{K}\alpha \to \mathsf{K}\beta)$
3. $\mathsf{K}\alpha \to (\alpha \wedge \mathsf{K}\mathsf{K}\alpha)$
4. $\langle \mathsf{K} \rangle \alpha \to \mathsf{K}\langle \mathsf{K} \rangle \alpha$,

where $A \in \mathrm{PROP}$ and $\alpha, \beta \in \mathfrak{F}$. It is expressed in this way that, for every Kripke model M validating these axioms, the accessibility relation of M belonging to the knowledge operator is an equivalence.

Now let $\mathcal{F}$ be a fixed set of function symbols and KAP be the relevant set of procedures. The axioms describing the elements of KAP then read as follows.

5. $[P](\alpha \to \beta) \to ([P]\alpha \to [P]\beta)$
6. $\langle F \rangle \alpha \to [F]\alpha$
7. $(A \to [F]A) \wedge (\langle F \rangle A \to A)$
8. $[P;Q]\alpha \leftrightarrow [P][Q]\alpha$
9. $[P \cup Q]\alpha \leftrightarrow [P]\alpha \wedge [Q]\alpha$
10. $[P^*]\,\alpha \to \alpha \wedge [P]\,[P^*]\,\alpha$
11. $[P^*]\,(\alpha \to [P]\alpha) \to (\alpha \to [P^*]\,\alpha)$
12. $[K\alpha?]\beta \leftrightarrow (K\alpha \to \beta)$,

where $F \in \mathcal{F}$, $P \in \mathrm{KAP}$, $A \in \mathrm{PROP}$ and $\alpha, \beta \in \mathfrak{F}$. Most of these schemata are well-known from usual PDL; cf [3], § 10. We point to the different ones only. Axiom 6 captures the deterministic case, Axiom 7 is due to the special semantics given here (see the remark following Definition 2), and Axiom 12 reflects the present restrictions concerning tests.

The most interesting group of axioms deals with the interplay between knowledge and procedures.

13. $\mathsf{K}[F]\alpha \to [F]\mathsf{K}\alpha$
14. $[F]\mathsf{K}\alpha \to \mathsf{K}[F]\alpha \vee [F]\beta$,

where $F \in \mathcal{F}$ and $\alpha, \beta \in \mathfrak{F}$. A couple of points are worth mentioning. Axiom 13 represents the variant of the *Cross Axiom* from [2] which is appropriate to the context of this paper. A commutation axiom of this type is typical of every logic of knowledge and effort. Axiom 14 determines to what extent the converse of Axiom 13 is valid. Note that both schemata need to be required only for the basic procedures. Moreover, both axioms have a counterpart in the linear time version of temporal $\mathcal{L}$; cf [10].

From this list, we obtain the logical system KAL by adding the standard proof rules of modal logic, i.e., *modus ponens* and *necessitation with respect to each modality.*

Definition 4 (The logic). *The smallest set of formulas containing the axiom schemata 1 – 14 and being closed under the application of the following rule schemata is denoted* KAL :

$$(\textsc{modus ponens}) \quad \frac{\alpha \to \beta, \alpha}{\beta} \qquad (\Delta\text{–}\textsc{necessitation}) \quad \frac{\alpha}{\Delta\alpha},$$

where $\alpha, \beta \in \mathfrak{F}$ *and* $\Delta \in \{\mathsf{K}\} \cup \{[P] \mid P \in \mathrm{KAP}\}$.

It is easy to see that KAL is sound for standard SSPs. In the remaining part of this section, we show that KAL is also complete with respect to this class of structures. The argumentation in proving this is partly similar to that in case of (deterministic) PDL. Thus we may be brief regarding this and let the differences take first place instead.

Let $\alpha_0 \in \mathfrak{F}$ be not contained in KAL, and let Σ_0 be the Fischer-Ladner closure of α_0; cf [3], p 112. We do some more work on Σ_0. First, we close the set $\Sigma_0 \cup \{\neg\alpha \mid \alpha \in \Sigma_0\}$ under finite conjunctions of pairwise distinct elements. Second, we form the closure under single applications of the operator $\langle \mathsf{K} \rangle$. And finally, we collect all subformulas of the formulas obtained up to now. (This step is necessary since $\langle \mathsf{K} \rangle$ is an abbreviation.) Let Σ denote the resulting set of formulas. Then Σ is finite and subformula closed.

Furthermore, let KAP_{Σ_0} be the smallest subset of KAP including the set $\mathcal{F}_0$ of all elements of $\mathcal{F}$ occurring in Σ_0 and all tests contained in Σ_0, and being closed under ;, $\cup$, and *.

We must construct an SSP falsifying α_0. To this end, we start off with the canonical model of KAL restricted to the relations determined by KAP_{Σ_0} : $\mathcal{M}^c = (X^c, R^c_{\mathsf{K}}, \{R^c_P \mid P \in \mathrm{KAP}_{\Sigma_0}\}, V^c)$. This model is then collapsed in the usual way. Since we are free to choose a filtration of the relations R^c_F, where $F \in \mathcal{F}_0$, we take the minimal one for this. The minimal filtration too is taken for the modality K. Let $\overline{\mathcal{M}} = \big(\overline{X}, \overline{R_{\mathsf{K}}}, \{\overline{R_P} \mid P \in \mathrm{KAP}_{\Sigma_0}\}, \overline{V}\big)$ be the resulting structure. (For later purposes it should be mentioned that $\overline{V}(A) = \emptyset$ holds for all $A \notin \Sigma_0$, without loss of generality.) Then we have the following proposition.

Proposition 1. *$\overline{\mathcal{M}}$ is a filtration of $\mathcal{M}^c$, the PDL–reduct of $\overline{\mathcal{M}}$ is a standard model (in the spirit of usual PDL), and $\neg\alpha_0$ is satisfied in $\overline{\mathcal{M}}$ at some point x_0.*

Proof. See [3], Theorem 10.7 and Corollary 10.9. Only the case of tests is different from that of PDL here and requires attention thus. However, this case does not cause any complications.

So far we have exploited the properties of Σ_0 and the axioms (5 and) 8 – 12. Now we utilize the structure of Σ as well as the axioms 1 – 4 and 13 – 14.

Proposition 2. *The model $\overline{\mathcal{M}}$ satisfies the following properties:*

1. *$\overline{R_{\mathsf{K}}}$ is an equivalence relation.*
2. *For all $F \in \mathcal{F}_0$ and $x, y, z \in \overline{X}$ such that $x\,\overline{R_F}\,y\,\overline{R_{\mathsf{K}}}\,z$, there exists $v \in \overline{X}$ such that $x\,\overline{R_{\mathsf{K}}}\,v\,\overline{R_F}\,z$.*
3. *For all $F \in \mathcal{F}_0$ and $x, y, z \in \overline{X}$ such that $x\,\overline{R_{\mathsf{K}}}\,y\,\overline{R_F}\,z$ and x has some $\overline{R_F}$–successor at all, there exists $v \in \overline{X}$ such that $x\,\overline{R_F}\,v\,\overline{R_{\mathsf{K}}}\,z$.*

Proof. For each item, one first shows that the corresponding property holds for the canonical model and then that it passes down to the filtration. In particular, the structure of the filter set Σ, the S5–properties of K, and Axiom 13, are used for item 2. For item 3, Axiom 13 has to be replaced with Axiom 14. Note that a rather involved argument like that in the proof of Proposition 7 from [11] is required each time.

The remaining difficulties of the proof are connected with Axiom 6. It is a known fact that the functionality of a relation gets lost in passing down to a filtration. This shortcoming has to be rectified now. For a start, we have at least a partial result.

Proposition 3. *Let $F \in \mathcal{F}_0$, $x, y \in \overline{X}$ satisfy $x\,\overline{R_F}\,y$, and $\langle F\rangle\alpha \in \Sigma$. Suppose that $\overline{\mathcal{M}}, y \models \alpha$, where $\models$ denotes the PDL satisfaction relation. Then $\overline{\mathcal{M}}, z \models \alpha$ is true for all $\overline{R_F}$–successors z of x.*

Proof. See [12], 3.5. Note that this is the place where Axiom 6 plays its part.

The further proceeding relies heavily on the proof of [12], Theorem 4.1. We quote those points of this proof that we shall need below in the following lemma.

Lemma 1. *1. For each $x \in \overline{X}$, let $D(x) := \{\langle F\rangle\alpha \in \Sigma \mid \overline{\mathcal{M}}, x \models \langle F\rangle\alpha\}$. Then there exists a finite tree T_x such that*

(a) the nodes of T_x are labelled by elements of $\overline{X}$ and the edges of T_x are labelled by elements of $\mathcal{F}_0$,

(b) the root is labelled by x,

(c) if there is an edge labelled by F from y to z, then $y\,\overline{R_F}\,z$,

(d) for each node on the tree and all $F \in \mathcal{F}_0$ there is at most one outgoing edge labelled by F,

(e) for all $\langle F\rangle\alpha \in D(x)$, if $\langle F\rangle\alpha$ equals $\langle P_1\rangle \ldots \langle P_k\rangle\beta$, where β is not of the form $\langle P\rangle\gamma$, then there exists a node y on the tree such that $x\,\overline{R_{P_1}} \circ \ldots \circ \overline{R_{P_k}}\,y$ and $\overline{\mathcal{M}}, y \models \beta$.

2. The trees from item 1 can be composed in such a way that a finite deterministic standard model T of $\neg\alpha_0$ results in case α_0 is in the K–free fragment of the language.

Proof. Item 1 is essentially [12], Lemma 4.6, (a) – (d). Only the definition of $D(x)$ is slightly different here (Σ_0 is replaced with Σ), but this does not change the original proof. Item 2 is a rather lax summary of the final part of the proof of Theorem 4.1 there. We shall be more precise in a minute when we modify the construction of T appropriately.

We now describe how a structure realizing $\neg\alpha_0$ can be obtained, which is both a deterministic PDL model and a Kripke model with respect to $\overline{R_{\mathsf{K}}}$; moreover, the knowledge axioms as well as the conditions from Proposition 2 will be satisfied. In a following step, this model is converted into a semantically equivalent SSP. First we state one more technical lemma.

Lemma 2. *Let $x \in \overline{X}$, and let T_x be the tree given by Lemma 1. Moreover, let T', T'' be trees such that*

1. *both T' and T'' satisfy (a) – (d) from item 1 of Lemma 1,*
2. *T' and T'' are isomorphic (i.e., there is a bijection, respecting the labelling of edges, between the sets of nodes of T' and T'').*

Finally, assume that T' is a subtree of T_x. Then there exists a tree $\hat{T}_x$ which contains T'' as a subtree and satisfies all the requirements from Lemma 1.1.

Proof. Proposition 3 as well as further concepts and results from [12] (Lemma 4.3, Definition 4.4, and Lemma 4.5) have to be applied. The detailed proof of Lemma 2 is deferred to the full version of this paper.

The construction of a set of deterministic trees (in fact, a forest) underlying the intermediate model falsifying α_0 proceeds in five steps. The first one is taken from [12] (final part of the proof of Theorem 4.1).

1. Let $x_0 \in \overline{X}$ be such that $\overline{\mathcal{M}}, x_0 \models \neg\alpha_0$ (see Proposition 1). Let T_0 be x_0 and, for all $n \in \mathbb{N}$, T_{n+1} be T_n with the following modifications:
   (a) if x is a leaf of T_n and T_x has not been used yet, then replace x with T_x,
   (b) if x is a leaf of T_n and T_x has already been used, then take T_x, delete x, and draw the corresponding edge from the predecessor of x in T_n to the root of T_x.

   Then let $T^0 := \bigcup_n T_n$. T^0 is in fact a finite deterministic tree.
2. Let $\{x_0, x_1, \ldots, x_m\}$ be the $\overline{R_{\mathsf{K}}}$–equivalence class of x_0. Construct deterministic trees $T^1, \ldots, T^m$ being rooted in $x_1, \ldots, x_m$, respectively, like T^0 in the previous step.
3. Define the *joint tree* of $\{T^0, T^1, \ldots, T^m\}$, denoted $\tilde{T}$, in the following way by induction.
   (a) Take a new point $y_0 \notin \overline{X}$. Let y_0 be the root of $\tilde{T}$. Label y_0 by the empty sequence ().
   (b) Let $\tilde{T}$ be already defined up to level n. Take any (temporary) leaf x and suppose that x is labelled by $(F_{i_1}, \ldots, F_{i_n})$, where $F_{i_1}, \ldots, F_{i_n} \in \mathcal{F}_0$. For every $F \in \mathcal{F}_0$, do the following exactly once: if for some $i \in \{0, \ldots, m\}$ there is an $(F_{i_1}, \ldots, F_{i_n}, F)$–path in T^i starting at the root, then take a new point y (i.e., $y \notin \overline{X}$ and y has not been used up to now), let y be a node of level $n+1$ of $\tilde{T}$, and label y by $(F_{i_1}, \ldots, F_{i_n}, F)$.
4. For $i = 0, \ldots, m$, let T'_i be the maximal subtree of T^i which is isomorphic to a subtree T''_i of $\tilde{T}$. Let $\sigma^i : T''_i \longrightarrow T'_i$ be a corresponding isomorphism. Pass the labelling of nodes from T'_i over to T''_i by means of $(\sigma^i)^{-1}$. Then let $\hat{T}^i_0$ be the tree containing T''_i whose existence is guaranteed by Lemma 2.
5. For each of the trees $\hat{T}^i_0$ there exists a canonical mapping $\pi^i_0 : U^i_0 \longrightarrow \overline{X}$, where U^i_0 is the set of nodes of $\hat{T}^i_0$. Now, if need be, change the labelling of the nodes of $\hat{T}^i_0$ in such a way that, whenever $y, y', z, z' \in T''_i \cap T''_j$, $\pi^i_0\sigma^i(y)\,\overline{R_F}\,\pi^i_0\sigma^i(z)$, $\pi^j_0\sigma^j(y')\,\overline{R_F}\,\pi^j_0\sigma^j(z')$, and $\pi^i_0\sigma^i(y)\,\overline{R_{\mathsf{K}}}\,\pi^j_0\sigma^j(y')$, then $\pi^i_0\sigma^i(z)\,\overline{R_{\mathsf{K}}}\,\pi^j_0\sigma^j(z')$, for all $i, j = 0, \ldots, m$ and $F \in \mathcal{F}_0$. Let $\hat{T}^i$ be the resulting tree ($i = 0, \ldots, m$).

The following lemma is almost immediate from Proposition 2.3 and Lemma 2.

Lemma 3. *All the requirements stated in item 5 above can be satisfied.*

Now, let U_i be the set of nodes of $\hat{T}^i$. We may assume that these sets are pairwise disjoint (which can easily be achieved, if need be). Let $\pi_i : U_i \longrightarrow \overline{X}$ be the canonical mapping ($i = 0, \ldots, m$). Moreover, let $X' := \bigcup_i U_i$. For every $F \in \mathcal{F}$, define $R'_F \subseteq X' \times X'$ by $s\, R'_F\, t :\Longleftrightarrow$ there are $i \in \{0, \ldots, m\}$ such that $s, t \in U_i$ and an edge from s to t in $\hat{T}^i$ labelled by F, for all $s, t \in X'$. Extend these relations to KAP in such a way that the standard model conditions are fulfilled. Furthermore, let $s\, R'_{\mathsf{K}}\, t :\Longleftrightarrow \exists\, i, j \in \{0, \ldots, m\} : s \in U_i,\ t \in U_j$ and $\pi_i(s)\, \overline{R_{\mathsf{K}}}\, \pi_j(t)$, for all $s, t \in X'$. And finally, let a Kripke valuation $V' : \mathrm{PROP} \longrightarrow \mathcal{P}(X')$ be given by $\mathcal{M}', s \models A :\Longleftrightarrow \exists\, i \in \{0, \ldots, m\} : s \in U_i$ and $\overline{\mathcal{M}}, \pi_i(s) \models A$, for all $s \in X'$ and $A \in \mathrm{PROP}$. With that, we have the next lemma.

Lemma 4. *1. Each of the relations R'_F is a partial function, and R'_{K} is an equivalence.*
2. The resulting model $\mathcal{M}' = (X', R'_{\mathsf{K}}, \{R'_P \mid P \in \mathrm{KAP}\}, V')$ validates all the above axioms.
3. $\mathcal{M}', x_0 \models \neg\alpha_0$.

The proof of this lemma is omitted here as well, but we mention that, among other things, Proposition 2 is used for it.

We now turn to the final step of the construction, giving us the desired SSP $\mathcal{M}$. Let the carrier set X of $\mathcal{M}$ consist of all paths f *through* any of the trees $\mathcal{M}'$ consists of. For every $F \in \mathcal{F}_0$, define a relation of *precedence* between R'_{K}–equivalence classes by $[s] \preccurlyeq_F [t] :\Longleftrightarrow \exists\, s' \in [s], t' \in [t]$ such that there is an edge from s' to t' labelled by F ($s, t \in X'$). Now, every chain $[s_0] \preccurlyeq_{F_1} \ldots \preccurlyeq_{F_n} [s_n]$ starting at the smallest class $[s_0]$ (which consists precisely of the roots of the trees $\hat{T}^i$) determines an open $U_{[s_0]^{F_1} \ldots {}^{F_n}[s_n]}$ in the following way. If f is any element of X, then

$$f \in U_{[s_0]^{F_1} \ldots {}^{F_n}[s_n]} :\Longleftrightarrow \begin{cases} [s_0] \preccurlyeq_{F_1} \ldots \preccurlyeq_{F_n} [s_n] \text{ and , for all } i = 0, \ldots, n, \\ f \text{ passes through } [s_i] \text{ by respecting } F_i. \end{cases}$$

Let $\mathcal{O} := \{U_{[s_0]^{F_1} \ldots {}^{F_n}[s_n]} \mid n \in \mathbb{N},\ s_0, \ldots, s_n \in X',\ F_1, \ldots, F_n \in \mathcal{F}_0$ and $[s_0] \preccurlyeq_{F_1} \ldots \preccurlyeq_{F_n} [s_n]\}$. For all $F \in \mathcal{F}$, define $R_F \subseteq \mathcal{O} \times \mathcal{O}$ by

$$U_{[s_0]^{F_1} \ldots {}^{F_n}[s_n]}\, R_F\, U_{[t_0]^{G_1} \ldots {}^{G_l}[t_l]} :\Longleftrightarrow \begin{cases} l = n+1,\ [s_i] = [t_i] \text{ for all } i = 0, \ldots, n, \\ F_j = G_j \text{ for all } j = 1, \ldots, n-1, \\ \text{and } G_l = F. \end{cases}$$

Extend these relations to KAP as above. Letting additionally A hold at the neighbourhood situation $f, U_{[s_0]^{F_1} \ldots {}^{F_n}[s_n]}$ iff A is true in $\mathcal{M}'$ at some node t of f (where $A \in \mathrm{PROP}$, $s_0, \ldots, s_n \in X'$ and $F_1, \ldots, F_n \in \mathcal{F}_0$), we obtain the final lemma of this section.

Lemma 5. *The structure $\mathcal{M} = (X, R_{\mathsf{K}}, \{R_P \mid P \in \mathrm{KAP}\}, V)$ is a standard SSP falsifying α_0.*

Proof. We first argue that R_F is a partial function, for all $F \in \mathcal{F}$. We may restrict attention to the function symbols contained in $\mathcal{F}_0$. It was achieved, in particular, by step 5 of the above construction (right before Lemma 3) that not only every element s of X' has at most one R_F–successor in $\mathcal{M}'$, but also the whole class $[s]$ of s. To be more precise, the construction ensured that, for all $s, s', t, t' \in X'$, if $s\, R'_F\, t$, $s'\, R'_F\, t'$ and $s\, R'_{\mathsf{K}}\, s'$, then $t\, R'_{\mathsf{K}}\, t'$. This shows that each R_F is really a partial function.

Now we prove that $U\, R_F\, U'$ implies $U' \subseteq U$, for all $F \in \mathcal{F}_0$ and $U, U' \in \mathcal{O}$. Let $U' = U_{[s_0]^{F_1 \ldots F_n}[s_n]}$, for some $n \in \mathbb{N}$, $s_0, \ldots, s_n \in X'$ and $F_1, \ldots, F_n \in \mathcal{F}_0$. Let $g \in U'$. Then, for all $i = 0, \ldots, n$ there exists a uniquely determined node g_{s_i} of g being contained in $[s_i]$, and for all $i = 0, \ldots, n-1$ the edge leading from g_{s_i} to $g_{s_{i+1}}$ is labelled by F_i. Since $U\, R_F\, U'$ is assumed to be valid, we infer $U = U_{[s_0]^{F_1 \ldots F_{n-1}}[s_{n-1}]}$ from that. This proves that $g \in U$. Consequently, $U' \subseteq U$.

Our next task is to show that V is well-defined. Let $A \in \mathrm{PROP}$ be given. Then A is true in $\mathcal{M}'$ at some node of f, iff A is true in $\mathcal{M}'$ at every node of f. Actually, $\mathcal{M}'$ inherits this property from $\overline{\mathcal{M}}$, for which it is valid because of Axiom 7 and the fact that we chose the minimal filtration of the accessibility relations of the canonical model; see also the remark preceding Proposition 1.

Finally, the following property, which suffices for the remaining part of Lemma 5, can be proved by structural induction (notations as above):

> For all $\alpha \in \Sigma$, $f \in X$, and $U = U_{[s_0]^{F_1 \ldots F_n}[s_n]} \in \mathcal{O}$ such that $f \in U$, we have that $f, U \models_{\mathcal{M}} \alpha$ iff $\mathcal{M}', f_{s_n} \models \alpha$.

The induction is not carried out here. (Again, the reader is referred to the full version of the paper.) – This finishes the proof of the lemma.

The main part of the subsequent first theorem of this paper is an immediate consequence of Lemma 5.

Theorem 1 (Soundness and completeness). KAL *is sound and complete with respect to the class of all standard SSPs.*

We have even prepared virtually everything we need to prove the second of our main results.

Theorem 2 (Decidability). KAL *is a decidable set of formulas.*

Proof. We showed, in particular, the *finite frame property* of KAL. It is a known fact that this property implies the decidability of the logic; cf [13],

Concluding this section we attend to the complexity of the KAL-satisfiability problem, S_{KAL}. In case of deterministic PDL, the corresponding problem is EXPTIME-complete; cf [12], Sec. 5. By using this we can show that S_{KAL} is hard for EXPTIME anyway. As to the upper bound, we mention the method applied in the proof of Theorem 5.1 of the paper [12], which is usually called *Elimination of Hintikka sets;* cf [13], Sec. 6.8. Roughly speaking, superfluous

elements of the filtration $\overline{\mathcal{M}}$ are faithfully eliminated with that. Now, the essential difference to the present case is that the filter set Σ has another order of magnitude here. Thus even if we apply the *Elimination* method successfully we shall not get a matching complexity bound. – Unfortunately, the determination of the exact complexity of S_{KAL} must be postponed to future research.

4 Concluding Remarks

In this paper, we developed a dynamic logic of knowledge acquisition, KAL, by combining a variant of Moss and Parikh's logic of knowledge and deterministic PDL. Looking into the merits of the new system we can tell that on the one hand knowledge-based programming and on the other hand topological reasoning is supported to some extent. The theoretical results we obtained back up this thesis. We proved both the semantic completeness and the decidability of the arising logic.

We combined only the most basic systems of the respective fields in this very first step. Thus there remains a lot to be done with regard to both theory and practical applications. Two problems concerning the first area already appeared above: the integration of the $\Box$–operator (see Example 2) and the determination of the complexity of KAL. Because of their fundamental character these problems should be solved next.

Concerning the integration of the operator $\Box$, there is a 'solution' suggesting itself provided that there are only finitely many elementary procedures, $\mathcal{F} = \{F_0, \ldots, F_n\}$, and the iterated application of the elements of $\mathcal{F}$ is exhaustive in the following sense:

$$\mathcal{O} = \left(\bigcup_{F \in \mathcal{F}} R_F \right)^* ,$$

where $\mathcal{O}$ is the distinguished set of subsets of the underlying frame. In fact, $\Box$ is then definable by

$$\Box\alpha :\equiv \left[(F_0 \cup \ldots \cup F_n)^* \right] \alpha,$$

for all $\alpha \in \mathfrak{F}$. Now, the formula $[P^*]\,\alpha \rightarrow \Box\Diamond\alpha$ from Example 2 actually says that the particular procedure P represents a computable contraction; cf once again the literature we referred to at the end of Section 2.

In connection with the $\Box$–operator, one might also think of a combination of the temporal systems from [10] or [11] and the PDL-like one presented in this paper. This means adding an S5–component to *dynamic linear time temporal logic;* cf [14]. Moreover, multi-agent versions of such systems are very desirable in view of recent developments in dynamic epistemic logic.

Acknowledgement

I would like to thank the anonymous referees very much for their constructive reviews of this paper.

References

1. Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. In: Moses, Y. (ed.) Theoretical Aspects of Reasoning about Knowledge (TARK 1992), pp. 95–105. Morgan Kaufmann, Los Altos, CA (1992)
2. Dabrowski, A., Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. Annals of Pure and Applied Logic 78, 73–110 (1996)
3. Goldblatt, R.: Logics of Time and Computation. In: Center for the Study of Language and Information, 2nd edn., Stanford, CA. CSLI Lecture Notes, vol. 7 (1992)
4. Schmidt, R.A., Tishkovsky, D.: Combining dynamic logics with doxastic modalities. In: Balbiani, P., Suzuki, N.Y., Wolter, F., Zakharyaschev, M. (eds.) Advances in Modal Logic 4, pp. 371–391. King's College Publications, London (2003)
5. Baltag, A., Moss, L.S.: Logics for epistemic programs. Synthese 139, 165–224 (2004)
6. van Ditmarsch, H., van der Hoek, W., Kooi, B. (eds.): Dynamic Epistemic Logic. Synthese Library, vol. 337. Springer, Heidelberg (2007)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge, MA (1995)
8. Artemov, S., Davoren, J., Nerode, A.: Modal logics and topological semantics for hybrid systems. Technical Report MSI 97-05, Cornell University (1997)
9. Konev, B., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Dynamic topological logics over spaces with continuous functions. In: Governatori, G., Hodkinson, I., Venema, Y. (eds.) Advances in Modal Logic 6, pp. 299–318. College Publications, London (2006)
10. Heinemann, B.: A modal logic for discretely descending chains of sets. Studia Logica 76, 67–90 (2004)
11. Heinemann, B.: Temporal Aspects of the Modal Logic of Subset Spaces. Theoretical Computer Science 224, 135–155 (1999)
12. Ben-Ari, M., Halpern, J.Y., Pnueli, A.: Deterministic propositional dynamic logic: Finite models, complexity, and completeness. Journal of Computer and System Sciences 25, 402–417 (1982)
13. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
14. Henriksen, J.G., Thiagarajan, P.S.: Dynamic linear time temporal logic. Annals of Pure and Applied Logic 96, 187–207 (1999)

Resource Placement in Networks Using Chromatic Sets of Power Graphs

Navid Imani[1], Hamid Sarbazi-Azad[2,1], and Selim G. Akl[3]

[1]IPM School of Computer Science, Tehran, Iran
[2] Sharif University of Technology, Tehran, Iran
[3] School of Computing, Queen's University, Kingston, Ontario, Canada
imani@ipm.ir, azad@sharif.edu, akl@cs.queensu.ca

Abstract. In this paper, using the chromatic properties of power graphs we propose a new approach for placing resources in symmetric networks. Our novel placement scheme guarantees a perfect placement when such a solution is feasible in the topology.

Keywords: Interconnection networks, Resource placement, Symmetric graphs, Power graphs, Chromatic number.

1 Introduction

There might be several types of resources in a multicomputer that each processor needs to access. These resources may encompass I/O processors, memory or software packages. However, it is not often an economical nor it is a feasible or logical choice to place one resource from each type in each node of the system. In general, then, the problem of resource placement is how to disseminate some limited resources over the nodes of the network giving comparable access to all processors.

Resource placement has extensively been studied in the literature. Most of the solutions to this problem have been put forth for specific topologies such as the hypercube [9], [10], [22], [18], [24] and the torus [19], [20], [21], [5], [6], [7], [3]. Resource placement in the star interconnection network has been studied in [4]. While a variety of techniques have been envisioned, most of the presented placement schemes use the error correcting linear coding theory for distinguishing the nodes which contain the resources from the other nodes of the network.

Let G be a graph with a vertex set $V(G)$ and edge set $E(G)$. We denote by $\deg_G(v)$ the degree of a given vertex v in G and by $dist_G(v,u)$ the distance between two given vertices u and v in G. Throughout this paper we use N , $Dim(G)$, and $\deg(G)$, to refer to the number of vertices in G , the diameter and the degree of G, respectively. The k^{th} power of G denoted by G^k is a graph obtained from G by adding edges to G between any two vertices whose distance from each other is less than or equal to k for some $k \geq 1$. Obviously, $G^1=G$. Due to their interesting properties, power graphs have been the source of much attention in the past and power of several classes of graph have been studied extensively in the literature [16], [8], [12], [13], [1]. The applications of power graphs appear in different fields like routing in networks, quantum random walks in physics, etc.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 158–167, 2007.

The problem of coloring powers of graphs has also been considered in the past where the powers of some specific classes of graphs like planer graphs [1] and chordal graphs [2] have been studied. Vertex coloring of power graphs has been used to solve different problems like interleaving [15], distributing data storage and sphere packing [11], [17].

In this paper, we propose a novel approach, using the notion of *chromatic sets*, for deriving resource placement schemes for symmetric network. In particular, we study the chromatic properties of powers of symmetric graphs, and show that the chromatic sets of the k^{th} power of a symmetric graph can be used to derive resource placement schemes for the former network. Our proposed solution for resource placement is optimal in the sense that it results in a perfect placement whenever such a placement is feasible depending on the topology, number of resources and distance at which the placement strategy is desired. We will also propose some criteria for checking the feasibility of the resource placement schemes.

2 Preliminaries and Definitions

A resource allocation strategy is said to be *perfect* iff no two resource nodes are adjacent to each other, and all non-resource nodes are adjacent to the same number of resource nodes [4]. Similarly, a *perfect distance-d placement* is achieved when the resources are distributed such that every non-resource node is within a distance *d* or less from exactly one resource node and the distance between no two resource nodes is less than or equal to *d*. An allocation strategy is called an *m-perfect adjacency* if it is perfect, and each non-resource node is adjacent to exactly *m* resource nodes.

Definition 1: We call a source placement scheme a *k-perfect distance-d placement* if 1) every non-resource node is within a distance *d* or less from exactly *k* resource nodes and 2) the distance between any two resource nodes is more than *d*. Thus, perfect distance-*d* placement and *m*-perfect adjacency are two specific cases of *k*-perfect distance-*d* placement obtained when d=1 and k=1, respectively.

While a perfect placement scheme is always preferable, but a perfect solution may not exist for some graph topologies for some *k* and *d* values. Whenever referring to a placement scheme which does not satisfy the perfectness criteria, we would simply omit the term "perfect", e.g. *k* distance-*d* placement is a placement scheme which does not satisfy the second condition in definition 1. In [4] it is proved that any perfect resource set is a dominating set and a maximum independent set for the corresponding graph topology. We would use this result later in order to put forth our initial idea for resource placement.

Theorem 1: Given a regular graph *G*, the number of resource copies required to achieve an *m*-adjacency placement for *G* is given by

$$R(G,m) = \frac{Nm}{\deg(G)+m}$$

For perfect *m*-adjacency (if it exists), *R* is an integer number.

Proof: If R nodes are to be chosen as the resource nodes, out of the entire N nodes in the network, $N-R$ nodes would remain as non-resource nodes. According to the definition of perfect adjacency no two resource nodes can be adjacent. Therefore, each resource node would be adjacent to $\deg(G)$ non-resource nodes. The total number of edges in G which connect some resource node to some non-resource node then could be derived as $R\deg(G)$. As each non-resource node is adjacent to m resource node, the number of edges in G which connect a non-resource node to a resource node would be $m(N-R)$. As G is a digraph the number of edges from resource nodes to non-resource nodes is equal to the number of edges from non-resource nodes to the resource nodes. Hence, $m(N-R)=\deg(G)R$ and from here we can derive R as the proposed equation. □

A graph G is *symmetric* if it is both vertex and edge transitive, i.e. each pair of vertices and each pair of edges are equivalent under some automorphism group [14]. Every symmetric graph is a regular graph but the inverse is not generally true. Hereafter, we use the term 'symmetric' to refer to 'connected symmetric' graphs since all of the graphs being considered in the context of interconnection networks are connected, while the symmetric networks are the most attractive yet common topologies.

The *chromatic number* of a graph G, denoted by $\chi(G)$ is the smallest number of colors $\gamma(G)$ needed to color the vertices of G so that no two adjacent vertices share the same color [23]. The number of graph vertices in the largest complete sub-graph of G, denoted by $\omega(G)$ is called the *clique number* of G. It is easy to see that the chromatic number of a graph must be greater than or equal to its clique number. G is said to be *perfect* if for every induced subgraph H of G, the clique number we have $\omega(H)=\chi(H)$. Similarly, we define a *quasi-perfect* graph as a graph whose clique number and chromatic number are equal. It is easy to see that any perfect graph is quasi-perfect as well but the inverse is not true. Obviously, the quasi-perfectness is considered a weaker property than the perfectness.

The clique number is a measure of the local connectivity of a graph. If G is symmetric we can easily conclude that every vertex of G is contained in a clique of size $\omega(G)$. The $\chi(G)$ parameter on the other hand, expresses some of the properties of the network as well. In particular, we expect a symmetric graph whose chromatic number is somewhat bigger then its clique number to expose some irregularity in its general structure. Conversely, in this study we introduce the quasi-perfect graphs as a class of topologies with extremely regular structures. We will show that this attractive property would make the problem of resource placement quite straightforward for this class of graph topologies.

Definition 2: Let G be a graph with the chromatic number $\chi(G)$, i.e. G can be colored with $\chi(G)$ colors so that no two adjacent edges have the same color. Each such coloring divides the vertices of the graph to $\chi(G)$ sets each containing the vertices which the same color. We call each such a set a *chromatic set* for G.

Theorem 2: Let G be any symmetric graph. G^d is symmetric of degree $Vol_G(d)$ where $Vol_G(d)$ is the *excluding volume of radius d* of G which is defined as the number of vertices within distance d from any given vertex.

Proof: If G is a symmetric graph then each vertex v of G has k vertices at the 1-distance. We can state the same argument for the vertices at the 1-distance of v to derive the 2-distance neighbors of v. In general, having the vertices at the distance i of v, we can derive the vertices at the distance i+1 by finding the neighbors of the vertices at the distance i of v and ignoring the already visited vertices. As G is symmetric and v was initially chosen arbitrarily, starting from any other vertex u of G we will have the same situation at each stage while finding vertices of larger distance. Hence, the number of vertices within distance d from any vertex of G is equal to $Vol_G(d)$. As in G^k any vertex is connected to all vertices within distance d from that vertex in G, the degree of each vertex in G^k would be $Vol_G(d)$. Let $A:V(G)\rightarrow V(G)$ be the automorphism group defined on the vertex set of G which maps v to u and let R denote the adjacency relation. The automorphism group on the edge set of G, $B:V(G)\times V(G)\rightarrow V(G)\times V(G)$ is defined as $B(xRy)=A(x)RA(y)$ for any two arbitrary adjacent vertices in G. Hence to derive the automorphism group for G^d we define B as $B(xR^dy)=A(x)R^dA(y)$. Then, A and B define vertex and edge automorphism groups for G^d under which x and its distance 1 to d neighbors would be equivalent to y and its distance 1 to d neighbors. As x and y were chosen arbitrarily for any two vertices (edges) in G^d, there exists an automorphism group under which the two vertices (edges) are equivalent. Therefore, G^k is symmetric. □

3 Chromatic Properties and Adjacency Placement

In this section, we study the chromatic properties of symmetric graphs and propose an adjacency placement for the symmetric networks accordingly. Our initial idea for using the chromatic numbers for resource placement was shaped based on the fact that the graph coloring by its definition assigns a particular color to the vertices which are disseminated in the network. We can always assure that no two nodes in a chromatic set are adjacent. This is one of the conditions which are required for a perfect adjacency placement. The chromatic number of a graph on the other hand, is a measure of the connectivity of the corresponding network i.e. the chromatic number of a highly connected network is rather big which means that the number of nodes in each of its chromatic set is small. This is in correspondence with the fact that for a highly connected network fewer resources are needed as each node can gain access to many resource nodes within few hops.

In order to study the effectiveness of the resource set in a resource placement scheme we need to define some metrics for the resource sets.

Definition 3: Let G be a symmetric connected graph and $s \subset V(G)$ be a dominating set for G. Then, the *suffusion ratio* of a set s in G is defined as:

$$\sigma_s = \frac{2}{|s|(|s|-1)} \sum_{u,v \in s, u \neq v} \frac{d(u,v)}{d^*}$$

or equivalently as:

$$\sigma_s = \frac{\underset{u,v \in s}{E}\left(d(u,v)\right)}{d^*}, \quad u \neq v$$

where $\underset{u,v \in s}{E}$ is defined as the expected value calculated over all different vertex pairs of (s, d) which specifies the distance between any two given vertices and $d^* = Dim(G)$.

The parameter σ_s is a measure of the dissemination of the nodes of s throughout the other nodes of the graph. It calculates the average distance between any pair of nodes in the set related to the diameter of the network which is always an upper bound for any distance in the network.

In the context of resource placement, it would be desired to choose s of a specific size such that it maximizes σ_s. Next, we show that the chromatic sets provide the maximum σ_s and hence are the best candidates for the resource set.

Theorem 3: Let G be a quasi-perfect symmetric graph with clique number of $\omega(G)$; then any given vertex v of G is contained in a clique of size $\omega(G)$.

Proof: As the clique number of G is $\omega(G)$ there should be at least a single clique of this size in G. Yet as G is a symmetric graph no two vertices of G differ in their connectivity to their neighboring nodes, i.e. each vertex can be obtained from another vertex with an automorphism group. Hence, if there exists such a clique C in the graph with a vertex v contained in it, every other vertex of G should also be in some clique C' obtained from C with an automorphism group. □

Theorem 4: Let G be a quasi-perfect symmetric graph, then any chromatic set s of G is a dominating set for G.

Proof: As G is quasi-perfect, as a result of last theorem, each arbitrary vertex of G is contained in a clique of size $\chi(G)$. First, we show that each such a clique contains exactly a single vertex from each of different $\chi(G)$ chromatic sets of G. A clique by definition is a complete subgraph, i.e. any two arbitrary vertices in a clique are connected. Hence, while coloring the graph, no two vertices of the clique can be colored with the same color. As the size of the clique is $\chi(G)$, it contains exactly one vertex from each chromatic set. This fact implies that each given vertex v of G has neighbors from all different chromatic sets. Then, for any arbitrary chromatic set s corresponding to the vertices which are colored in a color C_s, v is either colored in C_s (i.e. contained in s) or is adjacent to s. Therefore, s is a dominating set for G. □

Theorem 5: Let G be a quasi-perfect symmetric graph. Then any chromatic set s of G is a maximum independent set for G.

Proof: Let s be the set of entire vertices colored in a specific color C_s and u and v be any two arbitrary vertices in s. As the membership of u and v is defined by the graph coloring and because u and v have the same color, they cannot be neighboring vertices. As we chose u and v arbitrarily, no two vertices in s all neighbors and hence

this results in the independency of s. To prove that s is maximal, we show that adding any other vertex of G to s abrogates it independency. Let x be any given arbitrary vertex of G such that $x \notin s$. x belongs to a clique of size $\chi(G)$ and hence a neighboring vertex of G in that clique belongs to s. Therefore adding x to s causes the set to loose its independency. Therefore, s is maximal. □

Theorem 6: For a given quasi-perfect graph G, we have $|V(G)| \mid \chi(G)$ i.e. $|V(G)|$ is divisible to $\chi(G)$. Furthermore, each chromatic set s of G contains $\frac{|V(G)|}{\chi(G)}$ vertices.

Proof: Let us assume that $V(G)$ is not divisible to $\chi(G)$. Then $|V(G)| \bmod \chi(G)$ of the chromatic sets of G has at least one vertex more than the other chromatic sets, i.e. they have $\left\lceil \frac{|V(G)|}{\chi(G)} \right\rceil$ vertices while the others have only $\left\lfloor \frac{|V(G)|}{\chi(G)} \right\rfloor$ vertices. Let s be a chromatic set containing $\left\lfloor \frac{|V(G)|}{\chi(G)} \right\rfloor$ vertices. As G is a symmetric graph, we can conclude that each vertex in G is contained in some clique of size $\omega(G)$. As some sets like s have fewer members, we expect that some vertices in G do not have a neighboring vertex in s. On the other hand, due to the fact that G is symmetric, the size of the clique for all the vertices is the same hence $\omega(G) \leq \chi(G) - 1$ which contradicts our assumption that G is quasi-perfect. Therefore, we have $|V(G)| \mid \chi(G)$.

Using the same way to prove that all the chromatic sets have equal number of vertices, we can show that if two chromatic sets have different number of vertices then the clique number of graph would be less than its chromatic number. Suppose s and s' be two chromatic sets with different number of vertices and $|s|>|s'|$. We can infer that some vertices in G are not neighbors of s' while they are neighbors of s, and as all of the vertices in G belong to a clique of the same size, we can infer that $\omega(G) \leq \chi(G) - 1$ that is again contradictory. Hence, the vertices of the graph are divided to $\omega(G)$ equal pieces each of size $\frac{|V(G)|}{\chi(G)}$. □

Corollary 1: Given a quasi-perfect graph G**,** each pair of the chromatic sets of G are equivalent under some automorphism group and thus have an equal suffusion ratio which is maximal.

Theorem 7: Any vertex v of a given chromatic set has at least $\chi(G)-1$ edges to the vertices contained in other $\chi(G)-1$ different chromatic sets.

Proof: As G is symmetric, for each vertex v in G, a subset of G which correspond to $K_{\omega(G)}$ can be built such that it contains v. This implies that each arbitrary vertex v in G has a degree of at least $\omega(G) -1$. If G is quasi-perfect then $\chi(G) = \omega(G)$. Now let us consider v and the set of its neighbors which are in a $K_{\omega(G)}$ with v. As they build a $K_{\omega(G)}$, each vertex is adjacent to all of the other vertices in that set; hence, each vertex in the set must receive a different color and belong to a different chromatic set.

Therefore, any arbitrary vertex of this set as v is adjacent to the vertices contained in $\chi(G) - 1$ different chromatic sets. □

Finally, using the results obtained in the last theorems, in what follows we prove that the chromatic sets provide the maximum suffusion ratio.

Theorem 8: Let G be a given symmetric graph. Then, a chromatic set s of G provides the maximal suffusion ratio throughout all dominating sets of size $|s|$.

Proof: Let us assume that s is not maximal. After replacing some vertex $x \in s$ by some vertex $y \notin s$ of G a set s' is obtained where its suffusion ratio is bigger than s, i.e. y is farther from the vertices in s relative to x. We show that no such a vertex could ever exist in G. As y is not in s, knowing that s is a dominating set we can conclude that y is adjacent to some vertex (vertices) in s. It is while any given vertex of s (like x) is not adjacent to any other vertices in s as they share the same color. Hence, replacing x with y would decrease the suffusion ratio and s is maximal.

Now that we proved the optimality of the chromatic sets we shall figure out to which class of placement would the resulting placements belong. The next two theorems investigate this issue in detail.

Theorem 9: Let G be a given symmetric quasi-perfect graph. Then, choosing one of the $\chi(G)$ chromatic sets of G as the resource set of the network results in a perfect m-adjacency placement where m can be obtained as $m = \dfrac{\deg(G)}{\chi(G) - 1}$.

Proof: We proved that any of the chromatic sets of a symmetric quasi-perfect graph is a maximum independent set and a dominating set for G. Suppose we choose a chromatic set s as the resource set. As s is an independent set, 1) no two resource nodes are adjacent and hence the resource placement is perfect. Furthermore, as s is a dominating set, 2) each given vertex in G is either a resource node or is adjacent to a resource node, i.e. at least a copy of the resource is accessible within a single hop from each non-resource node. Next we will derive the equation proposed for the number of adjacent resource nodes. Reminding from section 1, we derived the number of resource nodes required to achieve a perfect m-adjacency placement as:

$$R(G, m) = \frac{Nm}{\deg(G) + m}.$$

Now that we are using a chromatic set as the resource set, the number of resource nodes is known and equals $\dfrac{|V(G)|}{\chi(G)}$ where $|V(G)|$ (or equivalently N) denotes the number of nodes in the network. By replacing this value in the last equation for R we have:

$$\frac{N}{\chi(G)} = \frac{Nm}{\deg(G)+m}.$$

Finally, after solving the equation for m we get $m = \frac{\deg(G)}{\chi(G)-1}$. □

As we saw in the last theorem using a chromatic set as the resource set of a network results in a known resource placement. This placement is defined by the graph topology and hence there is no guarantee that it matches the exigencies of our specific network. In general, what is desired is a placement strategy which provides some degree of freedom for choosing or fine-tuning parameters such as the number of resources in the network, and the distance of placement. Although, the problem of resource placement is more dictated by the network topology rather than the designer of the network which means that many resource placement schemes are not achievable in a given graph topology, it is sometimes preferable to neglect some of the rigid conditions (e.g. perfectness) in order to obtain some flexibility while choosing crucial parameters such as number of resources in the network. In the next theorem, we would prove that by selecting more than one chromatic set as the resource set, one can obtain regular adjacency placements which let the presence of more resources in the network.

Theorem 10: Let G be a given symmetric quasi-perfect graph. By choosing the union of k arbitrary sets, $1 \le k \le \chi(G)$, out of the $\chi(G)$ chromatic sets of G as the resource set of the network, an mk-adjacency placement is achieved where m can be obtained as $m = \frac{\deg(v)}{w(G)-1}$. While this obviously does not suggest a perfect placement as the resource nodes can be adjacent, it guarantees the existence of mk neighboring resource nodes for each non-resource node.

Proof: If the union of more than one chromatic sets are chosen as the resource set, then the resource vertices still build a dominating set for G as adding new members to a dominating set does not disrupt its dominating property. Yet, this time the resource set cannot be an independent set as the resource nodes can be adjacent. Hence, the corresponding placement would be an mk-adjacncy. The number of resources adjacent to each node is obtained as the product of the number of chosen chromatic sets m and the number of vertices in each chromatic set which are adjacent to each non-resource node k. □

4 Conclusions

In this paper, by utilizing the chromatic properties of symmetric graphs, we showed that under some criteria the chromatic sets of a symmetric graph are the dominating sets and hence a solution to the problem of resource placement. To solve the problem for distances more than one, we studied the chromatic properties of power of

symmetric graphs. The results of this works are interesting in the sense that ours is the first study to consider the problem of resource placement in general for a class of networks. By using some heuristics, the same scheme can be applied to other classes of non-symmetric networks which were not discussed in this work.

Particularly, our work reduces the problem of resource placement to the well-know problem of graph coloring which has been extensively studied in graph theory and thus the existing high performance algorithms for deriving the chromatic set can be well utilized.

References

1. Agnarsson, G., Halldórsson, M.M.: Coloring powers of planar graphs. ACM-SIAM Symp. Discrete Algorithms, ACM-SIAM, 654–662 (2000)
2. Agnarsson, G., Greenlaw, R., Halldorsson, M.M.: On powers of chordal graphs and their colorings. Congress Numerantium 100, 41–65 (2000)
3. Albdaiwi, B.F., Livingston, M.L.: Perfect Distance-d Placements in 2D Toroidal Networks. J. Supercomp. 29, 45–57 (2004)
4. Alrabady, I., Mahmud, S.M., Chaudhary, V.: Placement of Resources in the Star Network. In: Proc. IEEE Int. Conf. on Algorithms and Architectures for Parallel Processing (1996)
5. Bae, M., Bose, B.: Resource placement in torus-based networks. In: Proc. IEEE International Parallel Processing Symposium, pp. 327–331. IEEE Computer Society Press, Los Alamitos (1996)
6. Bae, M., Bose, B.: Spare processor allocation for fault-tolerance in torus-based multicomputers. In: Proc. IEEE Int'l Symp. Fault-Tolerant Computing, pp. 282–291. IEEE Computer Society Press, Los Alamitos (1996)
7. Bae, M., Bose, B.: Resource placement in torus-based networks. IEEE Transaction on Computers 46, 1083–1092 (1997)
8. Brandstädt, A., Chepoi, V.D., Dragan, F.F.: Perfect elimination orderings of chordal powers of graphs. Discrete Mathematics 158, 273–278 (1996)
9. Chen, H., Tzeng, N.: Fault-tolerant resource placement in hypercube computers. In: Proc. International Conference on Parallel Processing, pp. 515–524 (1991)
10. Chen, H., Tzeng, N.: Efficient resource placement in hypercubes using multiple-adjacency codes. IEEE Transaction on Computers 43, 23–33 (1994)
11. Conway, G.H., Sloane, N.J.A.: SpherePacking, Lattices and Groups. Springer, Heidelberg (1988)
12. Dahlhaus, Duchet, E.P.: On strongly chordal graphs. Ars Combinatoria 24B, 23–30 (1987)
13. Effantin, B., Kheddouci, H.: The b-chromatic number of some power graphs. Discrete Mathematics and Theoretical Computer Science 60, 45–54 (2003)
14. Holton, D.A., Sheehan, J.: The Petersen Graph. Cambridge University Press, Cambridge, England (1993)
15. Jiang, M., Bruck, C.J.: Optimal t-Interleaving on Tori. In: Proc. IEEE Int'l Symposium on Information Theory, pp. 22–31. IEEE Computer Society Press, Los Alamitos (2004)
16. Kheddouci, H., Saclé, J.F., woźniak, M.: Packing of two copies of a tree into its fourth power. Discrete Mathematics 213, 169–178 (2000)
17. Linial, N., Peleg, D., Rabinovich, Y., Saks, M.: Sphere packing and local majorities in graphs. In: Proc. 2nd ISTCS, pp. 141–149 (1993)
18. Livingston, M., Stout, Q.: Distributing resources in hypercube computers. In: Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications, pp. 222–231 (1988)

19. AlBdaiwi AlMohammad, B., Bose, B.: I/O placements in 2D toroidal networks. In: Proc. IEEE Int'l Parallel Processing Symposium, pp. 431–438. IEEE Computer Society Press, Los Alamitos (1998)
20. AlBdaiwi AlMohammad, B., Bose, B.: On distance-d placements in 3D tori. In: Proc. Int'l Conf. on Par. and Distr. Processing Tech. and Appl., pp. 1733–1739 (2001)
21. AlBdaiwi AlMohammad, B., Bose, B.: On resource placements in 3D tori. In: Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics, pp. 96–101 (2001)
22. Reddy, A.L.N., Banerjee, P.: Design, analysis, and simulation of I/O architectures for hypercube multiprocessors. IEEE Trans. on Parallel and Distributed Systems 1, 140–151 (1990)
23. Skiena, S.: Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Addison-Wesley, Reading. MA (1990)
24. Tzeng, N., Feng, G.: Resource allocation in cube network systems based on the covering radius. IEEE Trans. on Parallel and Distributed Systems 7, 328–342 (1996)
25. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)

Conjunctive Grammars over a Unary Alphabet: Undecidability and Unbounded Growth

Artur Jeż[1,⋆] and Alexander Okhotin[2,3,⋆⋆]

[1] Institute of Computer Science, University of Wrocław, Poland
aje@ii.uni.wroc.pl
[2] Academy of Finland
[3] Department of Mathematics, University of Turku, Finland
alexander.okhotin@utu.fi

Abstract. It has recently been proved (Jeż, DLT 2007) that conjunctive grammars (that is, context-free grammars augmented by conjunction) generate some nonregular languages over a one-letter alphabet. The present paper improves this result by constructing conjunctive grammars for a larger class of unary languages. The results imply undecidability of a number of decision problems of unary conjunctive grammars, as well as nonexistence of an r.e. bound on the growth rate of generated languages. An essential step of the argument is a simulation of a cellular automaton recognizing positional notation of numbers using language equations.

1 Introduction

Formal languages over an alphabet consisting of a single letter, known as *unary languages*, can be regarded as sets of natural numbers, and the questions of representation of such sets by the standard devices of formal language theory form a special topic of study. Regular unary languages are just ultimately periodic sets, though there remain nontrivial questions, such as the economy of description studied by Chrobak [1]. Context-free unary languages are well-known to be regular, though, as shown by Domaratzki et al. [3], context-free grammars give very succinct descriptions of ultimately periodic sets. Simple types of cellular automata, such as *trellis automata* studied by Culik et al. [2], Ibarra and Kim [6] and others, are also limited to regular languages when considered over $\{a\}$.

All the above families of languages are characterized by systems of language equations of the general form

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad \vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{*}$$

with different operations allowed in their right-hand sides. As established by Ginsburg and Rice [4], context-free languages are obtained by using concatenation and union in (*). If concatenation is restricted to one-sided linear then the

⋆ Supported by MNiSW grant number N206 024 31/3826, 2006–2008.
⋆⋆ Supported by the Academy of Finland under grant 118540.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 168–181, 2007.

solutions represent exactly the regular languages. Finally, trellis automata, as shown by Okhotin [12], can be simulated by equations with union, intersection and two-sided linear concatenation.

The first example of a nonregular unary solution of a language equation was given by Leiss [9], who constructed a single equation with concatenation and completementation, such that its unique solution is $\{a^n \,|\, \text{the octal notation of } n \text{ starts with 1, 2 or 3}\}$. The general case of such language equations was recently studied by Okhotin and Yakimova [14].

While equations with complementation as the only Boolean operation are of a purely theoretical interest, some classes of language equations (*) constitute natural extensions of the context-free grammars. One of these classes are *conjunctive grammars* introduced by Okhotin [10], represented by language equations with union, intersection and concatenation. These grammars have good practical properties (such as efficient parsing algorithms) and are surveyed in a recent article [13].

The expressive power of conjunctive grammars over a unary alphabet was one of the most important open problems in the area [10,13], and it was conjectured that only regular languages are generated. This conjecture has recently been disproved by Jeż [7], by constructing a grammar for the language $\{a^{4^n} \mid n \in \mathbb{N}\}$, as well as grammars for a large class of languages of an exponential growth.

This result leaves us with some natural questions to ponder. How far does the expressive power of unary conjunctive languages extend? Are these languages restricted to exponential growth? Can their standard decision problems, such as emptiness, equivalence, etc., be effectively decided? In this paper we answer these questions by showing that the emptiness problem and other related decision problems are undecidable, while the growth is not recursively bounded, which by far exceeds earlier expectations on the power of this class of unary languages.

2 Conjunctive Grammars and Trellis Automata

Definition 1. *A conjunctive grammar [10] is a quadruple $G = (\Sigma, N, P, S)$, in which Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols respectively; P is a finite set of grammar rules, each of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_n \quad (\text{where } A \in N,\ n \geqslant 1 \text{ and } \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*) \qquad (1)$$

while $S \in N$ is a nonterminal designated as the start symbol.

Informally, a rule (1) states that if a word is generated by each α_i, then it is generated by A. This semantics can be formalized using term rewriting, which generalizes Chomsky's word rewriting.

Definition 2. *Given a grammar G, the relation $\Longrightarrow$ of immediate derivability on the set of terms is defined as follows: (I) Using a rule $A \to \alpha_1 \& \ldots \& \alpha_n$, a subterm A of any term $\varphi(A)$ can be rewritten as $\varphi(A) \Longrightarrow \varphi(\alpha_1 \& \ldots \& \alpha_n)$. (II) A conjunction of several identical words can be rewritten by one such word:*

$\varphi(w\&\ldots\&w) \Longrightarrow \varphi(w)$, *for every* $w \in \Sigma^*$. *The language generated by a term* $\mathcal{A}$ *is* $L_G(\alpha) = \{w \mid w \in \Sigma^*,\ \mathcal{A} \Longrightarrow^* w\}$. *The language generated by the grammar is* $L(G) = L_G(S) = \{w \mid w \in \Sigma^*,\ S \Longrightarrow^* w\}$.

An equivalent definition can be given using language equations. This definition generalizes the well-known characterization of the context-free grammars by equations due to Ginsburg and Rice [4].

Definition 3. *For every conjunctive grammar* $G = (\Sigma, N, P, S)$, *the associated system of language equations [11] is a system of equations in variables* N, *in which each variable assumes a value of a language over* Σ, *and which contains the following equation for every variable* A:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in P} \bigcap_{i=1}^{m} \alpha_i \quad (\textit{for all } A \in N)\,. \tag{2}$$

Each instance of a symbol $a \in \Sigma$ *in such a system defines a constant language* $\{a\}$, *while each empty string denotes a constant language* $\{\varepsilon\}$. *A solution of such a system is a vector of languages* $(\ldots, L_C, \ldots)_{C \in N}$, *such that the substitution of* L_C *for* C, *for all* $C \in N$, *turns each equation (2) into an equality.*

It is known that every such system has solutions, and among them the *least solution* with respect to componentwise inclusion, and this solution consists of exactly the languages generated by the nonterminals of the original conjunctive grammar: $(\ldots, L_G(C), \ldots)_{C \in N}$ [11]. This representation by language equations constitutes an equivalent semantics of conjunctive grammars, and it is this semantics, and not the fairly artificial derivation, that accounts for the intuitive clarity of conjunctive and context-free grammars.

Let us give conjunctive grammars for some standard examples of non-context-free languages:

Example 1. [10] The following conjunctive grammar generates the language $\{wcw \mid w \in \{a,b\}^*\}$:

$$\begin{array}{ll} S \to C\&D & D \to aA\&aD \mid bB\&bD \mid cE \\ C \to aCa \mid aCb \mid bCa \mid bCb \mid c & A \to aAa \mid aAb \mid bAa \mid bAb \mid cEa \\ E \to aE \mid bE \mid \varepsilon & B \to aBa \mid aBb \mid bBa \mid bBb \mid cEb \end{array}$$

The nonterminal D generates the language $\{uczu \mid u, z \in \{a,b\}^*\}$. This is done as follows: the rules for D match a symbol in the left part to the corresponding symbol in the right part using A or B, and the recursive reference to aD or bD makes the remaining symbols be compared in the same way. An intersection with the language $\{ucv \mid u, v \in \{a,b\}^*, |u| = |v|\}$ generated by C completes the grammar.

Example 2. [7] The following conjunctive grammar with the start symbol A_1 generates the language $\{a^{4^n} \mid n \geqslant 0\}$:

$$\begin{array}{l} A_1 \to A_2A_2\&A_1A_3 \mid a \\ A_2 \to A_{12}A_2\&A_1A_1 \mid aa \\ A_3 \to A_{12}A_{12}\&A_1A_2 \mid aaa \\ A_{12} \to A_3A_3\&A_1A_2 \end{array}$$

To understand this grammar, note that each A_i generates the language of all words a^ℓ, such that the base-4 notation of ℓ is given by the digit(s) i followed by zeroes.

This construction can be generalized to the following:

Theorem 1 (Jeż, 2007 [7]). *For every $k \geqslant 2$ and for every finite automaton A over $\{0, \ldots, k-1\}$, there exists a conjunctive grammar over $\{a\}$ generating all strings a^n, such that the k-ary notation of n is in $L(A)$.*

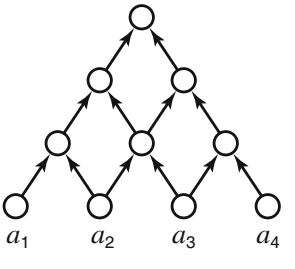
Let us now define an important subclass of conjunctive grammars, defined by analogy with linear context-free grammars. A conjunctive grammar is called *linear conjunctive* [10], if every rule it contains is either of the form $A \to u_1 B_1 v_1 \& \ldots \& u_n B_n v_n$, where $n \geqslant 1$, $u_i, v_i \in \Sigma^*$ and $B_i \in N$, or of the form $A \to w$, where $w \in \Sigma^*$. Note that the grammar in Example 1 is linear, while the grammar in Example 2 is not.

The family of languages defined by linear conjunctive grammars has actually been known for almost thirty years before these grammars were introduced [12]: this is the family recognized by one of the simplest types of cellular automata. These are *trellis automata*, also known as one-way real-time cellular automata, which were studied by Culik, Gruska and Salomaa [2], Ibarra and Kim [6], and others. Let us explain this concept following Culik et al. [2], who proposed it as a model of parallel computation in some electronic circuits.

A trellis automaton (TA), defined as a quintuple $(\Sigma, Q, I, \delta, F)$, processes an input string of length $n \geqslant 1$ using a uniform array of $n(n+1)/2$ processor nodes, as presented in the figure below. Each processor computes a value from a fixed finite set Q. The processors in the bottom row obtain their values directly from the input symbols using a function $I : \Sigma \to Q$. The rest of the processors compute the function $\delta : Q \times Q \to Q$ of the values in their predecessors. The string is accepted if and only if the value computed by the topmost processor belongs to the set of accepting states $F \subseteq Q$.

Definition 4. *A trellis automaton is a quintuple $M = (\Sigma, Q, I, \delta, F)$, in which: Σ is the input alphabet, Q is a finite non-empty set of states, $I : \Sigma \to Q$ is a function that sets the initial states, $\delta : Q \times Q \to Q$ is the transition function, and $F \subseteq Q$ is the set of final states.*
Extend δ to a function $\delta : Q^+ \to Q$ by $\delta(q) = q$ and

$$\delta(q_1, \ldots, q_n) = \delta(\delta(q_1, \ldots, q_{n-1}), \delta(q_2, \ldots, q_n)) \ ,$$



while I is extended to a homomorphism $I : \Sigma^* \to Q^*$.
Let $L_M(q) = \{w \mid \delta(I(w)) = q\}$ and define $L(M) = \bigcup_{q \in F} L_M(q)$.

Linear conjunctive grammars and trellis automata are computationaly equivalent:

Theorem 2 ([12]). *A language $L \subseteq \Sigma^+$ is recognised by a linear conjunctive grammar if and only if L is recognized by a trellis automaton. These representations can be effectively transformed into each other.*

Trellis automata over a unary alphabet recognize only regular languages, since in this case they behave as DFAs. Another property of TA which we often use is their closure under quotient with singletons [12], that is, whenever a language $L \in \Sigma^*$ is recognized by some trellis automaton M, then for every $a \in \Sigma$ the languages $a^{-1}L$ and La^{-1} are recognized by some TA M' and M''. These M', M'' can be effectively computed from M and a. From this the following result can be concluded:

Lemma 1. *Let L be a linear conjunctive language over an alphabet Σ, let $u, v \in \Sigma^*$ and $a \in \Sigma$. Then the language $L \cap ua^*v$ is regular.*

Proof. Let $K = L \cap ua^*v$. Then the language $\widetilde{K} = \{u\}^{-1} \cdot K \cdot \{v\}^{-1}$ is linear conjunctive by the closure of this family under quotient with singletons. Since $\widetilde{K}$ is a unary linear conjunctive language, it is regular. Then $K = u\widetilde{K}v$ is regular as well. □

3 Representing Grammars in Positional Notation

Words over a unary alphabet $\{a\}$ can be regarded as natural numbers, and, accordingly, languages over $\{a\}$ represent sets of numbers. Our constructions are based upon representing these numbers in positional notation, that is, we fix a number $k \geqslant 2$, define the alphabet $\Sigma_k = \{0, 1, 2, \ldots, k-1\}$ of k-ary digits and consider words over this alphabet that represent numbers. Unary languages are accordingly represented by languages over Σ_k.

Let the empty string $\varepsilon \in \Sigma_k^*$ denote the number 0. No representation of a number shall begin with 0, that is, the set of valid representations of numbers is $\Sigma_k^* \setminus 0\Sigma_k^*$. Define the bijection $f_k : \Sigma_k^* \setminus 0\Sigma_k^* \leftrightarrow a^*$ as

$$f_k(w) = \{a^n \mid w \text{ read as } k\text{-ary notation represents } n\} .$$

We extend this function for languages in a usual way, preserving the bijectiveness.

Let us define a language-theoretic operator $\boxplus_k : \Sigma_k^* \times \Sigma_k^* \to \Sigma_k^*$, $\boxplus$ for simplicity, which represents addition of numbers in k-ary notation:

$$u \boxplus v = \{\text{the } k\text{-ary notation of } i+j \mid u \text{ is the } k\text{-ary notation of } i, \\ v \text{ is the } k\text{-ary notation of } j\} .$$

It is extended to languages in the standard way: $K \boxplus L = \{u \boxplus v \,|\, u \in K, v \in L\}$. Now this operation can be used in the context of language theory, e.g., if $k = 10$, then one can say that $9^+ \boxplus 2 = 10^*1$. Let us assume that concatenation has the highest precendence, followed by $\boxplus$ and then by intersection and union.

Define the corresponding subtraction operator on languages as follows:

$$K \boxminus L = \{\text{the } k\text{-ary notation of } i-j \mid i \geqslant j, \text{ the } k\text{-ary notation of } i \text{ is in } K, \\ \text{the } k\text{-ary notation of } j \text{ is in } L\} .$$

Consider systems of language equations of the form $X_i = \varphi(X_1, \ldots, X_n)$ (i=$1, \ldots, n$) over the alphabet Σ_k with $\cup, \cap, \boxplus$ as allowed operations and with

regular constants. Since all operations are monotone, any such system has a least solution of the same kind as in systems from Definition 3. The following tight correspondence between these two types of systems can be established:

Definition 5. *Let $k \geqslant 2$. Let $X_i = \varphi(X_1, \ldots, X_n)$ be a system of language equations over the alphabet $\{a\}$ using concatenation and Boolean operations. The corresponding k-ary system of language equations over Σ_k is $Y_i = \psi(Y_1, \ldots, Y_n)$, obtained by replacing each X_j with Y_j, each concatenation operator with $\boxplus$ and each constant language $L \subseteq a^*$ with the language $f_k^{-1}(L)$.*

Lemma 2. *Let $X_i = \varphi_i(X_1, \ldots, X_n)$ be a system of language equations over the alphabet $\{a\}$ and let $Y_i = \psi_i(Y_1, \ldots, Y_n)$ be the corresponding language equations over Σ_k. Then a vector of languages $(f_k(L_1), \ldots, f_k(L_n))$ is a solution of the former system if and only if the vector of languages $(L_1, \ldots, L_n)$ is a solution of the latter system. In particular, if $(f_k(L_1), \ldots, f_k(L_n))$ is the least solution of the former system, then $(L_1, \ldots, L_n)$ is the least solution of the latter system.*

Note that for all languages $K, L \subseteq \Sigma_k^*$, $f_k(K \boxplus L) = f_k(K) \cdot f_k(L)$. The rest of the proof is by a straightforward structural induction.

Example 3. Consider the conjunctive grammar from Example 2, consider the system of language equations over $\{a\}$ corresponding to it by Definition 3, and then consider the system over $\Sigma_4 = \{0, 1, 2, 3\}$ corresponding to that system according to Definition 5.

$$\begin{aligned} X_1 &= (X_2 \boxplus X_2 \cap X_1 \boxplus X_3) \cup \{1\} \\ X_2 &= (X_{12} \boxplus X_2 \cap X_1 \boxplus X_1) \cup \{2\} \\ X_3 &= (X_{12} \boxplus X_{12} \cap X_1 \boxplus X_2) \cup \{3\} \\ X_{12} &= X_3 \boxplus X_3 \cap X_1 \boxplus X_2 \end{aligned}$$

The least solution of the system is $(10^*, 20^*, 30^*, 120^*)$, cf. Example 2.

4 A Representation of Trellis Automata

Let us now show how the computation of a trellis automaton can be simulated by the class of language equations introduced in the previous section.

Lemma 3. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k, such that $L(M) \cap \mathtt{0}^* = \varnothing$, there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using operations $\cup$, $\cap$ and $\boxplus$ and regular constants, such that the least solution of this system contains a component $((\mathtt{1} \cdot L(M)) \boxminus \mathtt{1}) \cdot \mathtt{10}^*$.*

Proof (Outline). In this proof we abuse the notation of $\boxplus$ and $\boxminus$ by allowing their arguments and the result to have leading zeroes. We shall do this only for the second argument equal to $\mathtt{1}$. Under these conditions we define the result to have the same length as the first argument, e.g., $\mathtt{0100} \boxminus \mathtt{1}$ is deemed to be $\mathtt{0099}$.

We shall never use this notation in a context where these requirements cannot be fulfilled, that is, for $(k-1)^+ \boxplus 1$ and for $0^* \boxminus 1$. This abused notation is used only in the text of the proof, while language equations strictly adhere to the definition.

Given a trellis automaton $M = (\Sigma_k, Q, I, \delta, F)$, we represent its input words $w \in \Sigma_k^*$ as words of the form $1(w \boxminus 1)10^m$, for all $m \geqslant 0$. This representation allows us to concatenate symbols to words both on the right and on the left by adding some appropriate numbers.

For a given M, we shall construct language equations with the set of variables X_q for $q \in Q$, and with an additional variable Y, such that their least solution is $X_q = L_q$, $Y = L$, where

$$L_q = 1\Big(\big(L_M(q) \setminus 0^*\big) \boxminus 1\Big)10^* = \{1w10^\ell \mid \ell \geqslant 0, w \notin 9^+, w \boxplus 1 \in L_M(q)\} \quad \text{and}$$

$$L = 1\Big(\big(L(M) \setminus 0^*\big) \boxminus 1\Big)10^* = \{1w10^\ell \mid \ell \geqslant 0, w \notin 9^+, w \boxplus 1 \in L(M)\}\,.$$

Let us define expressions λ_i and ρ_j, for $i, j \in \Sigma_k$, which depend upon the variables X_q, and which we use as building blocks for the equations for X_q.

$$\lambda_i(X) = 1i\Sigma_k^* \cap \bigcup_{i'} \Big((X \cap 1i'\Sigma^*) \boxplus 10^* \cap 2i'\Sigma_k^* \Big) \boxplus (k+i-2)0^* \text{ , for } i=0,1$$

$$\lambda_i(X) = 1i\Sigma_k^* \cap \bigcup_{i'} \Big((X \cap 1i'\Sigma^*) \boxplus 10^* \cap 2i'\Sigma_k^* \Big) \boxplus 1(i-2)0^* \text{ , for } i \geqslant 2$$

$$\rho_j(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus (k+j-2)10^* \Big) \cap \\ \cap\, 1\Sigma_k^* j10^* \text{ for } j = 0,1$$

$$\rho_j(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus 1(j-2)10^* \Big) \cap \\ \cap\, 1\Sigma_k^* j10^* \text{ for } 2 \leqslant j \leqslant k-2$$

$$\rho_{k-1}(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus (k-3)10^* \Big) \cap \\ \cap\, 1\Sigma_k^*(k-1)10^*$$

Let us also define constants R_q, for all $q \in Q$, which are regular by Lemma 1.

$$R_q = 1\Big(\big((0^*(\Sigma_k \setminus 0) \cup (\Sigma_k \setminus 0)0^*) \cap L_M(q)\big) \boxminus 1\Big)10^*$$

Using this notation, the system of language equations is constructed as follows:

$$\begin{cases} X_q = R_q \cup \bigcup\limits_{\substack{q',q'':\delta(q',q'')=q \\ i,j\in\Sigma_k}} \lambda_i(X_{q''}) \cap \rho_j(X_{q'}) \quad \text{(for all } q \in Q\text{)} \\ Y = \bigcup_{q\in F} X_q \end{cases}$$

In these equations the sets R_q represent the starting part of X_q used to compose longer words. A word $w \in \Sigma^{\geqslant 2}$ belongs to $L_M(q)$ if and only if there are

states q', q'' such that $\delta(q', q'') = q$ and $\Sigma_k^{-1} w \in L_M(q'')$ and $w\Sigma_k^{-1} \in L_M(q')$. And so a word $1(w \boxminus 1)10^*$ should belong to X_q if and only if there are two witnesses belonging to $X_{q''}$ and $X_{q'}$ (with some additional constraints). This is specified in λ and ρ, respectively. These expressions represent adding digits at some specific positions, so that selected digits in the original word could be modified in the resulting word, while the rest of the digits remain the same. The main technical difficulty is to force the addition of digits at proper positions. This is achieved by adding the digits in two phases, and by checking the form of intermediate and final results using intersection with regular constants.

The correctness of the construction is stated as follows.

Main Claim. The least solution of the system is $(\ldots, L_q, \ldots, L)$.

The first to be established are the correctness statements for the expressions λ_i and ρ_j. For each λ_i, its value on any singleton of the form $\{1w10^m\}$ is characterized as follows:

Claim 1. For every word $1w10^m \in 1\Sigma_k^+ 10^* \setminus 1(k-1)^*10^*$,

$$\lambda_i(\{1w10^m\}) = \{1iw10^m\}.$$

Given a number with the notation $1w10^m$, the expression λ_i, adds a number notated $10^{|w|+1+m}$ to it to obtain $2w10^m$. The subsequent addition of $(k+i-2)0^{|w|+1+|m|}$ results in a number with the notation $1iw10^m$. The intersections used in λ_i ensure that no other additions are possible.

Since λ_j is an operation on languages defined as a superposition of $\cup$, $\cap$ and $\boxplus$, it is additive in the sense that $\lambda_i(K) = \bigcup_{w \in K} \lambda_i(\{w\})$ for every language K. Hence Claim 1 actually describes $\lambda_i(K)$ for every $K \subseteq 1\Sigma_k^+ 10^* \setminus 1(k-1)^*10^*$.

A similar statement is established for ρ_j:

Claim 2. For every word $u \in 1\Sigma_k^+ 10^* \setminus 1(k-1)^*10^*$,

$$\rho_j(\{u\}) = \begin{cases} \{1wj10^m\}, & \text{if } u = 1w10^{m+1} \text{ and } j = k-1\ , \\ \{1wj10^m\}, & \text{if } u = 1(w \boxminus 1)10^{m+1} \text{ and } j \neq k-1\ . \end{cases}$$

Now let us substitute the intended solution $(\ldots, L_q, \ldots, L)$ into λ_i and ρ_i. Then Claims 1 and Claim 2 easily imply the following equalities:

Claim 3. $\lambda_i(L_q) = 1(i(L_M(q) \setminus 0^*) \boxminus 1)10^*$.

Claim 4. $\rho_j(L_q) = 1((L_M(q) \setminus 0^*)(j+1 \mod k) \boxminus 1)10^*$.

The next claim is that the words in the intended solution $(\ldots, L_q, \ldots, L)$ belong to the corresponding components of every solution of the system.

Claim 5. For the least solution $(\ldots, S_q, \ldots)$ of the system and for every $q \in Q$, $L_q \subseteq S_q$.

For every $1w10^n \in L_q$ it has to be shown that $1w10^n$ is contained in S_q as well. The proof is done inductively on $|w|$, following the structure of the computation of M. The basis is given by the set R_q, which places in S_q all words $1w10^n$ for all w such that $w \boxplus 1$ has one nonzero digit, which is the last one or the first one. For the induction step, one has $w \boxplus 1 = iuj \in L_M(q)$ and one has to prove that $1w10^n$ is in S_q. The definition of a trellis automaton implies $iu \in L_M(q')$ and $uj \in L_M(q'')$ for some q', q'', such that $\delta(q', q'') = q$. Then the induction hypothesis gives that $1(iu \boxminus 1)10^{n+1} \in S_{q'}$ and $1(uj \boxminus 1)10^n \in S_{q''}$, and from these two words the equation for X_q produces $1w10^n \in S_q$, according to Claims 1 and 2.

Let us now show that the substitution of the intended solution $(\ldots, L_q, \ldots, L)$ into the system is contained in this intended solution.

Claim 6. For every $q \in Q$, $L_q \supseteq \varphi_q(\ldots, L_q, \ldots)$, where φ_q denotes the right-hand side of the equation for X_q.

Any word $1w10^n$ in $\varphi_q(\ldots, L_q, \ldots)$ is contained both in $\lambda_i(L_{q''})$ and in $\rho_j(L_{q'})$, for some q', q'' such that $\delta(q', q'') = q$. By Claim 4, $(w \boxplus 1)\Sigma_k^{-1} \in L_M(q')$, and by Claim 3, $\Sigma_k^{-1}(w \boxplus 1) \in L_M(q'')$. Then $w \boxplus 1 \in L_M(q)$, which yields the claim.

The proof of the main claim follows by the elementary properties of fixed points. Taking the componentwise inclusions of vectors

$$(\ldots, \varnothing, \ldots) \sqsubseteq (\ldots, L_q, \ldots) \sqsubseteq (\ldots, S_q, \ldots),$$

which holds by Claim 5, one can iteratively apply the right-hand sides of the system to each of the three components, obtaining the following limits:

$$(\ldots, S_q, \ldots) \sqsubseteq (\ldots, L_q, \ldots) \sqsubseteq (\ldots, S_q, \ldots).$$

This shows that the intended solution of the system is its least solution. □

Lemma 4. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using the operations $\cup$, $\cap$ and $\boxplus$ as well as regular constants, such that its least solution contains a component $1 \cdot L(M)$.*

Proof. For every $j \in \Sigma_k$, consider the language $L(M) \cdot \{j\}^{-1}$. By the closure properties of trellis automata, this language is generated by a trellis automaton M_j. Then, by Lemma 3, there exists a system of language equations, such that one of its variables, Y_j, represents the language $(L(M) \cdot \{j\}^{-1}) \boxminus 1$.

Let us combine these equations for all j into a single system, and add a new equation

$$Z = \bigcup_{j=0}^{k-1} (Y_j \cap 1\Sigma^*1) \boxplus (1j \boxminus 1).$$

This equation uses the same technique as in Lemma 3. The value of Z is $L(M)$. □

Theorem 3. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k, such that no words in $L(M)$ start with 0, there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using the operations $\cup$, $\cap$ and $\boxplus$ as well as regular constants, such that its least solution contains a component $L(M)$.*

Proof. For every $i \in \Sigma_k \setminus \{0\}$, the language $\{i\}^{-1} \cdot L(M)$ is generated by a certain trellis automaton. By Lemma 4, there is a system of language equations, such that one of its variables, Z_i, represents the language $\{i\}^{-1} \cdot L(M)$.

Combine these systems and add a new variable T with the following equation:

$$T = \bigcup_{i,i'} \Big((Z_i \cap 1i'\Sigma_k^*) \boxplus (i-1)0^* \Big) \cap ii'\Sigma_k^*.$$

The right-hand side of this equation evaluates to $L(M)$ on the least solution, which is shown by the same method as in Lemmata 3 and 4. □

We now translate the above result to the languages generated by the unary conjunctive grammars. We need a small technical lemma to handle the cases of $k = 2, 3$.

Lemma 5. *Let $\ell = k^n$ for some natural $n > 0$. Then for every languages L, L' such that $f_k(L) = f_\ell(L')$, L is linear conjunctive if and only if L' is linear conjunctive. Given a trellis automaton for either of the languages, a trellis automaton for the other language can be effectively constructed.*

The proof is by a straightforward grouping of digits, and it is omitted.

Theorem 4. *For every $k \geqslant 2$ and for every trellis automaton M over $\Sigma_k = \{0, 1, \ldots, k-1\}$, such that no words in $L(M)$ start with 0, there exists and can be effectively constructed a conjunctive grammar generating $f_k(L(M))$.*

Proof. For big enough k, that is $k \geqslant 4$, by Theorem 3, there exists a resolved system of language equations over Σ_k with regular constants that defines $L(M)$. According to Lemma 2, there exists a corresponding system over $\{a\}$, which uses constants of the form $f_k(K)$ for regular languages $K \subseteq \Sigma_k^*$. But, according to Theorem 1, every such constant is generated by a conjunctive grammar. Combining these grammars with the system of equations, the requested conjunctive grammar is obtained.

For $k < 4$ we use Lemma 5 and the already proved results for $k = 4, 9$. □

5 Decision Problems for Unary Conjunctive Grammars

One of the main techniques of proving undecidability results in formal language theory is by representing one or another form of the language of computations of a Turing machine. Given a TM T over an input alphabet Ω, one represents its computations as words over an auxiliary alphabet Γ. For every $w \in L(T)$,

let $C_T(w) \in \Gamma^*$ denote some representation of the accepting computation of T on w. The language

$$\mathrm{VALC}(T) = \{w \natural C_T(w) \mid w \in \Omega^* \text{ and } C_T(w) \text{ is an accepting computation}\}$$

over the alphabet $\Omega \cup \Gamma \cup \{\natural\}$ is the language of valid accepting computations of T. It was shown by Hartmanis [5] that for a certain simple encoding $C_T : \Omega^* \to \Gamma^*$ the language $\mathrm{VALC}(T)$ is an intersection of two context-free languages, while the complement of $\mathrm{VALC}(T)$, denoted $\mathrm{INVALC}(T)$, is context-free. Being able to represent these languages is one of the crucial properties of trellis automata.

Proposition 1 ([12]). *For every Turing machine T there exists an encoding $C_T : \Omega^* \to \Gamma^*$ of its computations, such that $\mathrm{VALC}(T)$ is recognized by a trellis automaton.*

This leads to a number of undecidability results for TA, which are inherited by linear conjunctive grammars [12] and hence by conjunctive grammars of the general form [10]. However, it appears hard to replicate these results for the case of a unary alphabet—a straightforward approach fails due to the apparent lack of structure in words, on which all known encodings of $\mathrm{VALC}(T)$ rely. Contrary to this intuition, Theorem 4 asserts that if the computation histories in $\mathrm{VALC}(T)$ are regarded as notations of numbers, then, as a linear conjunctive language, $\mathrm{VALC}(T)$ can be specified in unary encoding by a unary conjunctive grammar.

Let us make some further technical assumptions on the encoding of these languages. Assume that $\mathrm{VALC}(T)$ is defined over an alphabet $\Sigma_k = \{0, \ldots, k-1\}$, for a suitable k, and that $0 \in \Omega$, so that no word in $\mathrm{VALC}(T)$ has a leading zero. Define the language $\mathrm{INVALC}(T)$ as $(\Sigma_k^* \setminus 0\Sigma_k^*) \setminus \mathrm{VALC}(T)$. These elaborations do not affect Proposition 1, so that we can use Theorem 4 to obtain the following result:

Lemma 6. *For every Turing machine T there exist and can be effectively constructed conjunctive grammars G and G' over the alphabet $\{a\}$, such that $L(G) = f_k(\mathrm{VALC}(T))$ and $L(G') = f_k(\mathrm{INVALC}(T))$.*

The undecidability of basic decision problems for unary conjunctive grammars, such as whether a given grammar generates $\varnothing$ or whether a given grammar generates a^*, can be easily inferred from this. Let us, however, establish a more general result:

Theorem 5. *For every fixed unary conjunctive language $L_0 \subseteq a^*$ there is no algorithm to decide whether a given conjunctive grammar over $\{a\}$ generates L_0.*

Proof. Let $G_0 = (\Sigma, N_0, P_0, S_0)$ be a fixed conjunctive grammar generating L_0. Suppose there is an algorithm to check whether $L(G) = L_0$ for any given conjunctive grammar G over $\{a\}$. Let us use this algorithm to solve the emptiness problem for Turing machines. Depending on the form of L_0, there are two cases.

Case I: L_0 contains no subset of the form $a^\ell(a^p)^*$, where $\ell \geqslant 0$ and $p \geqslant 1$. Given a Turing machine T, construct a conjunctive grammar $G_T = (\{a\}, N_T, P_T, S_T)$

for $f_k(\mathrm{VALC}(T))$. On the basis of G_T and G_0, construct a new conjunctive grammar $G = (\{a\}, N_T \cup N_0 \cup \{S, A\}, P_T \cup P_0 \cup P, S)$, where P contains the following four new rules: $S \to S_0$, $S \to S_T A$, $A \to aA$ and $A \to \varepsilon$. Then it can be proved that $L(G) = L_0$ if and only if $L(G_T) = \varnothing$, and the latter is equivalent to $L(T) = \varnothing$.

Case II: L_0 contains a subset $a^\ell(a^p)^*$, where $\ell \geqslant 0$ and $p \geqslant 1$. Assume that p is larger than the cardinality of the alphabet used for $\mathrm{INVALC}(T)$. Define $\mathrm{INVALC}(T)$ over a p-letter alphabet and consider the language $f_p(\mathrm{INVALC}(T) \cdot \mathtt{0})$, which is generated by some conjunctive grammar $G'_T = (\{a\}, N'_T, P'_T, S'_T)$. Using G_0 and G'_T, construct a new grammar $G = (\{a\}, N'_T \cup N_0 \cup \{S, B, C\}, P'_T \cup P_0 \cup P, S)$, where the new rules in P are as follows: $S \to S_0 \& B$, $S \to a^\ell S'_T$, $B \to a^i$ (for all $0 \leqslant i < \ell$), $B \to a^{\ell+i}C$ (for all $1 \leqslant i < p$), $C \to a^p C$ and $C \to \varepsilon$. Note that $L_G(B) = a^* \setminus a^\ell(a^p)^*$. For this grammar one can establish that $L(G) = L_0$ if and only if $\mathrm{INVALC}(T) = \Sigma_p^* \setminus \mathtt{0}\Sigma_p^*$, which in turn holds if and only if $L(T) = \varnothing$.

In each case we have shown that the emptiness problem of Turing machines would be decidable, which forms a contradiction. □

If L_0 is not generated by a conjunctive grammar, then this problem becomes trivial. Hence, the following characterization is obtained:

Corollary 1. *For different constant languages $L_0 \subseteq a^*$, the problem of testing whether a given conjunctive grammar over $\{a\}$ generates L_0 is either Π_1-complete or trivial.*

Theorem 6. *For conjunctive grammars over a unary alphabet there exist no algorithm to decide whether a given grammar generates a finite language (a regular language).*

Proof (a sketch). Given a Turing machine T, construct another TM T' that recognizes $\{\varepsilon\}$ if $L(T) \neq \varnothing$, and $\varnothing$ otherwise. Construct a conjunctive grammar G for $f_k(\mathrm{VALC}(T')) \cdot \{a^{k^n} \mid n \geqslant 0\}$. Then $L(G)$ is either nonregular (if $L(T) \neq \varnothing$) or empty. □

Having seen the above results, it is natural to ask whether unary conjunctive languages have any nontrivial decidable properties. It is known that the membership of a word can be decided in cubic time [10], but nothing besides this problem and its Boolean combinations is known to be decidable. Finding such an example (or perhaps proving its nonexistence) is left as a problem for future study.

6 Growth of Unary Conjunctive Languages

Every infinite unary language $L = \{a^{i_1}, a^{i_2}, \ldots, a^{i_n}, \ldots\}$ where $0 \leqslant i_1 < i_2 < \ldots < i_n < \ldots$, can be regarded as an increasing integer sequence, and it is natural to consider the growth rate of such sequences, represented by a function

$g(n) = i_n$. Obviously, the growth of every regular language is linearly bounded. The example of a conjunctive grammar for the language $\{a^{4^n} \mid n \geqslant 0\}$ [7], see Example 2, shows that the growth of unary conjunctive languages can be exponential, which raises two questions. First, can this growth be superexponential, and is there any upper bound for the growth rate of unary conjunctive languages? Second, can this growth be superlinear but subexponential, e.g., polynomial?

The following theorem gives the strongest possible answer to the first question:

Theorem 7. *For every recursively enumerable set of natural numbers X there exists a conjunctive grammar G over an alphabet $\{a\}$, such that the growth function of $L(G)$ is greater than that of X at any point.*

Proof. Let T be a Turing machine, which recognizes the set $X = \{i_1, i_2, \ldots, i_j, \ldots\}$, where $0 \leqslant i_1 < i_2 < \ldots < i_j < \ldots$, and the numbers are given to it in a binary notation. Consider the language $\mathrm{VALC}(T)$, which contains words $w_n = f_2^{-1}(a^n)\#C_T(n)$. By Lemma 6, there exists a conjunctive grammar G over an alphabet $\{a\}$ that generates $L = f_k(\mathrm{VALC}(T))$ for some $k \geqslant 2$.

Let $g(n)$ be the growth function of L. It is sufficient to show that $g(j) \geqslant i_j$ for each $j \geqslant 1$. To see this, consider the values $g(1), g(2), \ldots, g(j)$. At least one of them describes a computation $w_{j'}$ for $j' \geqslant j$. Since g is an increasing function, we obtain $g(j) \geqslant f_k(w_{j'}) > j' \geqslant j$. □

Note that this quick-growing language is bound to be computationally very easy, as the upper bound of parsing complexity for conjunctive grammars is $DTIME(n^3) \cap DSPACE(n)$ [10,13].

The next example gives a unary conjunctive language of a polynomial growth.

Proposition 2. *There exists a conjunctive grammar G over an alphabet $\{a\}$, such that the growth function of $L(G)$ satisfies is $g(n) = \Theta(n^2)$.*

Proof (a sketch). Consider the set of numbers $X = \{(2^m + 3i) \cdot 2^m \mid m \geqslant 0,\ 2^m \leqslant 2^m + 3i < 2^{m+2}\}$. Let $g(n)$ denote the n-th largest element of X; this is the growth function of the corresponding unary language $L = \{a^n \mid n \in X\}$. The set of binary notations of the numbers in X is

$$f_2^{-1}(L) = \left\{ 1w0^m \;\middle|\; |w| = m-1 \text{ or } |w| = m, \text{ and } f_2(w) \text{ divides by } 3 \right\}.$$

This is clearly a linear context-free language, hence L is conjunctive by Theorem 4. Let us prove that $n^2 \leqslant g(n) \leqslant 4n^2$.

It is not hard to prove that for any number $n = 2^m + j$, where $0 \leqslant j < 2^m$, it holds that $g(2^m + j) = (2^m + 3j)2^m$; in particular, $g(2^m) = 2^{2m}$. Then, to see that $g(n) \geqslant n^2$, consider $g(2^m + j)$ for $0 \leqslant j < 2^m$. We have

$$g(2^m + j) = (2^m + 3j)2^m = 2^{2m} + 3j \cdot 2^m \geqslant 2^{2m} + 2j \cdot 2^m + j^2 = (2^m + j)^2,$$

where the inequality is due to $j \cdot 2^m \geqslant j^2$. On the other hand,

$$g(2^m + j) \leqslant g(2^{m+1}) = 2^{2m+2} \leqslant 4(2^m + j)^2,$$

which proves the upper bound $g(n) \leqslant 4n^2$. □

This construction can be generalized to obtain the following result:

Theorem 8. *For every rational number $p/q \geqslant 1$ there exists a conjunctive grammar G over an alphabet $\{a\}$, such that the growth function of $L(G)$ is $g(n) = \Theta(n^{p/q})$.*

7 Conclusion

We can conclude that though we have established that the growth of unary conjunctive grammars can be as high as theoretically possible, we still have no means of proving that some particular unary languages cannot be represented by conjunctive grammars, and hence the class of conjunctive languages still could not be separated from $NSPACE(n)$. Inventing a method for producing such nonrepresentability results for unary conjunctive grammars is left as an open problem.

References

1. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science 47, 149–158 (1986)
2. Culik II, K., Gruska, J., Salomaa, A.: Systolic trellis automata, I and II. International Journal of Computer Mathematics 15, 195–212 (1984) and 16, 3–22 (1984)
3. Domaratzki, M., Pighizzini, G., Shallit, J.: Simulating finite automata with context-free grammars. Information Processing Letters 84, 339–344 (2002)
4. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. Journal of the ACM 9, 350–371 (1962)
5. Hartmanis, J.: Context-free languages and Turing machine computations. Proceedings of Symposia in Applied Mathematics 19, 42–51 (1967)
6. Ibarra, O.H., Kim, S.M.: Characterizations and computational complexity of systolic trellis automata. Theoretical Computer Science 29, 123–153 (1984)
7. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. In: DLT 2007, Turku, Finland, July 3–6, 2007 (to appear)
8. Jeż, A., Okhotin, A.: Language equations with addition in positional notation, TUCS Technical Report No 824, Turku Centre for Computer Science, Turku, Finland (June 2007)
9. Leiss, E.L.: Unrestricted complementation in language equations over a one-letter alphabet. Theoretical Computer Science 132, 71–93 (1994)
10. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
11. Okhotin, A.: Conjunctive grammars and systems of language equations. Programming and Computer Software 28, 243–249 (2002)
12. Okhotin, A.: On the equivalence of linear conjunctive grammars to trellis automata. Informatique Théorique et Applications 38(1), 69–88 (2004)
13. Okhotin, A.: Nine open problems for conjunctive and Boolean grammars. Bulletin of the EATCS 91, 96–119 (2007)
14. Okhotin, A., Yakimova, O.: On language equations with complementation. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 420–432. Springer, Heidelberg (2006)

Ruling Out Polynomial-Time Approximation Schemes for Hard Constraint Satisfaction Problems

Peter Jonsson[1], Andrei Krokhin[2], and Fredrik Kuivinen[1]

[1] Department of Computer and Information Science
Linköpings Universitet, SE-581 83, Linköping, Sweden
{petej,freku}@ida.liu.se
[2] Department of Computer Science
Durham University, Durham, DH1 3LE, UK
andrei.krokhin@durham.ac.uk

Abstract. The *maximum constraint satisfaction problem* (Max CSP) is the following computational problem: an instance is a finite collection of constraints on a set of variables, and the goal is to assign values to the variables that maximises the number of satisfied constraints. Max CSP captures many well-known problems (such as Max k-SAT and Max Cut) and so is **NP**-hard in general. It is natural to study how restrictions on the allowed constraint types (or constraint language) affect the complexity and approximability of Max CSP. All constraint languages, for which the CSP problem (i.e., the problem of deciding whether all constraints in an instance can be simultaneously satisfied) is currently known to be **NP**-hard, have a certain algebraic property, and it has been conjectured that CSP problems are tractable for all other constraint languages. We prove that any constraint language with this algebraic property makes Max CSP hard at gap location 1, thus ruling out the existence of a polynomial-time approximation scheme for such problems. We then apply this result to Max CSP restricted to a single constraint type. We show that, unless $\mathbf{P} = \mathbf{NP}$, such problems either are trivial or else do not admit polynomial-time approximation schemes. All our hardness results hold even if the number of occurrences of each variable is bounded by a constant.

Keywords: maximum constraint satisfaction, complexity, approximability.

1 Introduction

Many combinatorial optimisation problems are **NP**-hard so there has been a great interest in constructing approximation algorithms for such problems. For some optimisation problems, there exist collections of approximation algorithms known as *polynomial-time approximation schemes* (PTAS). An optimisation problem Π has a PTAS A if, for any fixed rational $c > 1$ and for any instance $\mathcal{I}$ of Π, $A(\mathcal{I}, c)$ returns a c-approximate (i.e., within c of optimum) solution

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 182–193, 2007.

in time polynomial in $|\mathcal{I}|$. There are some well-known **NP**-hard optimisation problems that have the highly desirable property of admitting a PTAS: examples include KNAPSACK, EUCLIDEAN TSP, and INDEPENDENT SET restricted to planar graphs [1]. It is also well-known that a large number of optimisation problems do not admit a PTAS unless some unexpected collapse of complexity classes occurs. For instance, problems like MAX k-SAT and INDEPENDENT SET do not admit a PTAS unless $\mathbf{P} = \mathbf{NP}$ [1]. We note that if Π is a problem that does not admit a PTAS, then there exists a constant $c > 1$ such that Π cannot be approximated within c in polynomial time.

Constraint satisfaction problems (CSP) [19] and its optimisation variants have played an important role in research on approximability. Many combinatorial problems are subsumed by the CSP framework, and examples include problems in graph theory [10], combinatorial optimisation [13], and computational learning [7]. We will focus on a class of optimisation problems known as the *maximum constraint satisfaction problem* (MAX CSP). The most well-known examples in this class are MAX k-SAT and MAX CUT.

Let D be a finite set. A subset $R \subseteq D^n$ is called a *relation* and n is the *arity* of R. Let $R_D^{(k)}$ denote the set of all k-ary relations on D and let $R_D = \cup_{i=1}^{\infty} R_D^{(i)}$. A *constraint language* is a finite subset of R_D.

Definition 1 (CSP(Γ))**.** *The constraint satisfaction problem over the constraint language Γ, denoted* CSP(Γ)*, is defined to be the decision problem with instance (V, C), where*

- *V is a finite set of variables, and*
- *C is a (multi)set of constraints $\{C_1, \ldots, C_q\}$, in which each constraint C_i is a pair $(R_i, \boldsymbol{s_i})$ with $\boldsymbol{s_i}$ a list of variables of length n_i, called the constraint scope, and $R_i \in \Gamma$ is an n_i-ary relation in R_D, called the constraint relation.*

The question is whether there exists an assignment $s : V \to D$ which satisfies all constraints in C. A constraint $(R_i, (v_1, v_2, \ldots, v_{n_i})) \in C$ is satisfied *by an assignment s if the image of the constraint scope is a member of the constraint relation, i.e., if $(s(v_1), s(v_2), \ldots, s(v_{n_i})) \in R_i$.*

For a constraint language $\Gamma \subseteq R_D$, the optimisation problem MAX CSP(Γ) is defined as follows:

Definition 2 (MAX CSP(Γ))**.** MAX CSP(Γ) *is defined to be the optimisation problem with*

Instance: *An instance (V, C) of* CSP(Γ)*.*
Solution: *An assignment $s : V \to D$ to the variables.*
Measure: *Number of constraints in C satisfied by the assignment s.*

We use *multisets* of constraints instead of just sets of constraints as we do not have any weights in our definition of MAX CSP. We chose to use multisets instead of weights because bounded occurrence restrictions are easier to explain in the multiset setting. Note that we prove our hardness results in this restricted setting without weights and with a constant bound on the number of occurrences of each variable.

Example 1. Given a (multi)graph $G = (V, E)$, the MAX k-CUT problem, $k \geq 2$, is the problem of maximising $|E'|$, $E' \subseteq E$, such that the subgraph $G' = (V, E')$ is k-colourable. For $k = 2$, this problem is known simply as MAX CUT. The problem MAX k-CUT is known to be **APX**-complete for any k (it is Problem GT33 in [1]), and so has no PTAS. Let N_k denote the binary disequality relation on $\{0, 1, \ldots, k-1\}$, $k \geq 2$, that is, $(x, y) \in N_k \iff x \neq y$. To see that MAX CSP($\{N_k\}$) is precisely MAX k-CUT, think of vertices of a given graph as of variables, and apply the relation to every pair of variables x, y such that (x, y) is an edge in the graph, with the corresponding multiplicity.

Most of the early results on the complexity and approximability of MAX CSP were restricted to the Boolean case, i.e. when $D = \{0, 1\}$. For instance, Khanna et al. [13] characterise the approximability of MAX CSP(Γ) for all Γ over the Boolean domain. It has been noted that the study of non-Boolean CSP gives a better understanding (when compared with Boolean CSP) of what makes CSP easy or hard: it appears that many observations made on Boolean CSP are special cases of more general phenomena. Recently, there has been some major progress in the understanding of non-Boolean CSP: Bulatov has provided a complete complexity classification of the CSP problem over a three-element domain [3] and also given a classification of constraint languages that contain all unary relations [2]. Corresponding results for MAX CSP have been obtained by Jonsson et al. [11] and Deineko et al. [8].

It has been conjectured [4] that, for all constraint languages Γ, CSP(Γ) is either in **P** or is **NP**-complete (i.e., it cannot be **NP**-intermediate), and the conjecture also specifies the dividing line between the two cases, by means of a certain algebraic condition. Moreover, it was shown in [4] that, for all constraint languages Γ satisfying this condition, the problem CSP(Γ) is indeed **NP**-complete. In this paper we prove that, for such languages Γ, it is **NP**-hard to tell instances of MAX CSP(Γ) in which all constraints are satisfiable from instances where at most an ε-fraction of the constraints are satisfiable (for some constant ε which depends on Γ). In particular, this implies that, for such Γ, the problem MAX CSP(Γ) cannot admit a PTAS.

We then apply this result to study the case when the constraint language Γ consist of a single relation R. We show that, for such Γ, MAX CSP(Γ) is either trivial or else does not admit a PTAS. Finally, we use this last result to strengthen several earlier hardness results obtained in the study of MAX CSP via the algebraic property of supermodularity.

We obtain our results by techniques which are quite different from the ones used in [8,11]. In [11] it was proved that constraint languages over a three element domain which are cores and not *supermodular* (see Section 5 for a definition) give rise to MAX CSP-problems which do not admit a PTAS (it is in fact proved that they are **APX**-hard, which implies that they do not admit a PTAS, unless **P** = **NP**). The technique used in [11] is mainly that of *strict implementations*. With strict implementations, certain new relations can be constructed from old ones in a way that preserve the hardness of the corresponding MAX CSP-problem.

That is, if Γ is a constraint language which strictly implements a relation R, then MAX CSP($\Gamma \cup \{R\}$) is no harder than MAX CSP(Γ). This technique can be used to reduce the huge number of constraint languages to a set of constraint languages which is easier to reason about. Hardness results for this smaller set of constraint languages are then obtained from known results.

Let us fix a finite domain D and let U_D be the set of all unary constraints on D, that is $U_D = \{R \mid R \subseteq D\}$. In [8] it was proved that MAX CSP($\Gamma \cup U_D$) is **APX**-hard (and, therefore do not admit a PTAS) if $\Gamma \cup U_D$ is not supermodular on any chain (a *chain* is a lattice which is a total order) on D, and tractable (in **PO**) otherwise. The proof of this result uses a characterisation of relations which are supermodular on some chain together with strict implementations.

In contrast to the two results described above, we obtain the results in the present paper by quite different means. Our main result is proved by using perfect implementations and the associated theory of universal algebra. Universal algebra have previously been successfully used to classify the complexity of CSPs (we give an overview of this connection in Section 3, see also [4]). By using the notion of *hardness at gap location 1* and working with bounded occurrence MAX CSP-problems (this was also done in [8]) we manage to prove not only **NP**-hardness results, but also the impossibility of a PTAS (unless $\mathbf{P} = \mathbf{NP}$). Proofs of our results are omitted due to space constraints.

2 Preliminaries

A *combinatorial optimisation problem* is defined over a set of *instances* (admissible input data); each instance $\mathcal{I}$ has a set $\mathsf{sol}(\mathcal{I})$ of *feasible solutions* associated with it, and each solution $y \in \mathsf{sol}(\mathcal{I})$ has a value $m(\mathcal{I}, y)$. The objective is, given an instance $\mathcal{I}$, to find a feasible solution of optimum value. The optimal value is the largest one for *maximisation* problems and the smallest one for *minimisation* problems.

Definition 3 (Performance ratio). *A solution $s \in \mathsf{sol}(\mathcal{I})$ to an instance $\mathcal{I}$ of a optimisation problem Π is r-approximate if*

$$\max\left\{\frac{m(\mathcal{I},s)}{\mathrm{OPT}(\mathcal{I})}, \frac{\mathrm{OPT}(\mathcal{I})}{m(\mathcal{I},s)}\right\} \leq r,$$

where $\mathrm{OPT}(\mathcal{I})$ is the optimal value for a solution to $\mathcal{I}$. An approximation algorithm for an optimisation problem Π has performance ratio $R(n)$ *if, given any instance $\mathcal{I}$ of Π with $|\mathcal{I}| = n$, it outputs an $R(n)$-approximate solution.*

Definition 4 (PTAS). *An optimisation problem Π admits a* PTAS *if, for any rational constant $c > 1$, there is an algorithm that, given an instance $\mathcal{I}$ of Π, returns a c-approximate solution in time polynomial in $|\mathcal{I}|$.*

Definition 5 (Hard to approximate). *We say that a problem Π is* hard to approximate *if there exists a constant c such that it is* **NP***-hard to approximate Π with performance ratio c.*

Obviously, any problem that is hard to approximate cannot admit a PTAS. The following notion has been defined in a more general setting in [17].

Definition 6 (Hard gap at location α). MAX CSP(Γ) *has a* hard gap at location $\alpha \leq 1$ *if there exists a constant* $\varepsilon < \alpha$ *such that it is* **NP***-hard to decide if, for a given instance* $\mathcal{I} = (V, C)$ *of* MAX CSP(Γ), $\text{OPT}(\mathcal{I}) \geq \alpha|C|$ *or* $\text{OPT}(\mathcal{I}) \leq \varepsilon|C|$.

Note that if a problem Π has a hard gap at location α (for any α) then Π is hard to approximate. This simple observation has been used to prove inapproximability results for a large number of optimisation problems. See, e.g., [1,20] for surveys on inapproximability results and the related PCP theory.

Petrank [17] gave an informal conjecture which states that for "natural" optimisation problems hardness at gap location 1 can be used to show hardness at all other possible gap locations. Sometimes this can be done by a padding argument. Given an instance $\mathcal{I} = (V, C)$ of MAX CSP(Γ), we can add fresh variables V' and constraints C' to $\mathcal{I}$ such that at most a constant fraction ε of the constraints in C' can be satisfied simultaneously. If we choose C' and V' appropriately then we will obtain a proof of hardness at gap location α (for some α which depends on ε) for MAX CSP(Γ). Hence, in this sense hardness at gap location 1 is a stronger result than hardness at any other gap location. Petrank [17] give further arguments for why hardness at gap location 1 is a natural and interesting hardness notion.

Throughout the paper, MAX CSP(Γ)-k denotes the problem MAX CSP(Γ) restricted to instances where the number of occurrences of each variable is bounded by k. Note that if a variable occurs t times in a constraint which appears s times in an instance, then this would contribute $t \cdot s$ to the number of occurrences of that variable in the instance. The bounded occurrence property is closely related to bounding the degree in graphs. Re-considering Example 1, the problem MAX CSP($\{N_2\}$)-3 would correspond to MAX CUT restricted to (multi)graphs with maximum degree 3. In our hardness results, we will write that MAX CSP(Γ)-B is hard (in some sense) to denote that there is a k such that MAX CSP(Γ)-k is hard in this sense.

3 Hardness at Gap Location 1 for MAX CSP

We will now present the definitions and basic results we need from universal algebra. For a more thorough treatment of universal algebra in general we refer the reader to [5]. The article [4] contains a presentation of the relationship between universal algebra and constraint satisfaction problems.

An *operation* on a finite set D is an arbitrary function $f : D^k \to D$. Any operation on D can be extended in a standard way to an operation on tuples over D, as follows: Let f be a k-ary operation on D. For any collection of k n-tuples, $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in D^n$, the n-tuple $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k})$ is defined as follows:

$$f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) = (f(\boldsymbol{t_1}[1], \boldsymbol{t_2}[1], \ldots, \boldsymbol{t_k}[1]), \ldots, f(\boldsymbol{t_1}[n], \boldsymbol{t_2}[n], \ldots, \boldsymbol{t_k}[n])),$$

where $\boldsymbol{t_j}[i]$ is the i-th component in tuple $\boldsymbol{t_j}$. An operation $f : D^k \to D$ is said to be *idempotent* if $f(d, d, \ldots, d) = d$ for all $d \in D$, and it is called a *projection* if there is $1 \leq i \leq k$ such that $f(\boldsymbol{x}) = x_i$, for all $\boldsymbol{x} = (x_1, x_2, \ldots, x_k) \in D^k$.

Let R_i be a relation in the constraint language Γ. If f is an operation such that for all $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R_i$ we have $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) \in R_i$, then R_i is said to be *invariant* under f. If all relations in Γ are invariant under f, then Γ is said to be invariant under f. An operation f such that Γ is invariant under f is called a *polymorphism* of Γ. The set of all polymorphisms of Γ is denoted $\mathrm{Pol}(\Gamma)$. For a set of operations F, the set of all relations which are invariant under each operation in F is denoted $\mathrm{Inv}(F)$.

Example 2. Let $D = \{0, 1, 2\}$ and let R be the directed cycle on D, i.e., $R = \{(0, 1), (1, 2), (2, 0)\}$. One polymorphism of R is the operation $f : \{0, 1, 2\}^3 \to \{0, 1, 2\}$ defined as $f(x, y, z) = x - y + z \pmod 3$. This can be verified by considering all possible combinations of three tuples from R and evaluating f component-wise.

We continue by defining a closure operator $\langle\cdot\rangle$ on sets of relations: for any set $\Gamma \subseteq R_D$, the set $\langle\Gamma\rangle$ consists of all relations that can be expressed using relations from $\Gamma \cup \{EQ_D\}$ (where EQ_D denotes the equality relation on D), conjunction, and existential quantification. Those are the relations definable by *primitive positive formulae* (pp-formulae) using relations from $\Gamma \cup \{EQ_D\}$. As an example of a pp-formula consider the relations $A = \{(0, 0), (0, 1), (1, 0)\}$ and $B = \{(1, 0), (0, 1), (1, 1)\}$, over the boolean domain $\{0, 1\}$. With those two relations we can construct $I = \{(0, 0), (0, 1), (1, 1)\}$ with the pp-formula $I(x, y) \iff \exists z : A(x, z) \wedge B(z, y)$.

The sets of relations of the form $\langle\Gamma\rangle$ are referred to as *relational clones*, or *co-clones*. An alternative characterisation of relational clones is given in the following theorem.

Theorem 1 ([18])

- *For every set* $\Gamma \subseteq R_D$, $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.
- *If* $\Gamma' \subseteq \langle\Gamma\rangle$, *then* $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Gamma')$.

By using this connection between polymorphisms and relations definable by pp-formulae we obtain the following lemma. This lemma allows us to use some of the algebraic theory, which is commonly used when studying CSP, to get hardness results for Max CSP.

Lemma 1. *Let* Γ *be a constraint language and let* R *be a relation which is definable by a pp-formula using relations from* Γ. *If* Max CSP$(\Gamma \cup \{R\})$*-*k *has a hard gap at location 1, then* Max CSP(Γ)*-*k' *has a hard gap at location 1 for some integer* k'.

The notions of a core and a retraction play an important role in the study of graphs, and they can easily be generalised to constraint languages. A *retraction* of a constraint language Γ is a unary polymorphism $\pi \in \mathrm{Pol}(\Gamma)$ such that

$\pi(x) = x$ for all x in the image of π. We will say that Γ is a *core* if the only retraction of Γ is the identity function. Given a relation $R \in R_D^{(k)}$ and a subset X of D we define the *restriction of R onto X* as follows: $R\big|_X = \{\boldsymbol{x} \in X^k \mid \boldsymbol{x} \in R\}$. For a set of relations Γ we define $\Gamma\big|_X = \{R\big|_X \mid R \in \Gamma\}$. If π is a retraction of Γ with minimal image D', then a core of Γ is the set $\Gamma\big|_{D'}$. As in the case of graphs, all cores of Γ are isomorphic, so one can speak about *the* core of Γ.

The intuition here is that if Γ is not a core, then it has a non-injective retraction π, which implies that, for every assignment s, there is another assignment πs that satisfies all constraints satisfied by s and uses only a restricted set of values. Hence, the problem is equivalent to a problem over this smaller set.

Lemma 2. *If Γ' is the core of Γ, then, for any k,* Max CSP(Γ')*-k is hard at gap location 1 if and only if* Max CSP(Γ)*-k is hard at gap location 1.*

The three definitions below closely follows the presentation in [4].

Definition 7 (Finite algebra). *A* finite algebra *is a pair $\mathcal{A} = (A; F)$ where A is a finite non-empty set and $F = \{f_i^A \mid i \in I\}$ is a set of finitary operations on A.*

We will only make use of finite algebras so we will write *algebra* instead of *finite algebra*. An algebra is said to be *non-trivial* if it has more than one element.

Definition 8 (Homomorphism of algebras). *Given two algebras $\mathcal{A}=(A; F_A)$ and $\mathcal{B} = (B; F_B)$ such that $F_A = \{f_i^A \mid i \in I\}$, $F_B = \{f_i^B \mid i \in I\}$ and both f_i^A and f_i^B are n_i-ary for all $i \in I$, then $\varphi : A \to B$ is said to be an* homomorphism *from $\mathcal{A}$ to $\mathcal{B}$ if*

$$\varphi(f_i^A(a_1, a_2, \ldots, a_{n_i})) = f_i^B(\varphi(a_1), \varphi(a_2), \ldots, \varphi(a_{n_i}))$$

for all $i \in I$ and $a_1, a_2, \ldots, a_{n_i} \in A$. If φ is surjective, then $\mathcal{B}$ is a homomorphic image *of $\mathcal{A}$.*

For an operation $f : D^n \to D$ and a subset $X \subseteq D$ we define $f\big|_X$ as the function $g : X^n \to D$ such that $g(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in X^n$. For a set of operations F on D we define $F\big|_X = \{f\big|_X \mid f \in F\}$.

Definition 9 (Subalgebra). *Let $\mathcal{A} = (A; F_A)$ be an algebra and $B \subseteq A$. If for each $f \in F_A$ and any $b_1, b_2, \ldots, b_n \in B$, we have $f(b_1, b_2, \ldots, b_n) \in B$, then $\mathcal{B} = (B; F_A\big|_B)$ is a* subalgebra *of $\mathcal{A}$.*

The operations in Pol(Inv(F_A)) are the *term operations* of $\mathcal{A}$. If F consists of the idempotent term operations of $\mathcal{A}$, then the algebra $(A; F)$ is called the *full idempotent reduct of $\mathcal{A}$*, and we will denote this algebra by $\mathcal{A}^c$. Given a set of relations Γ over the domain D we say that the algebra $\mathcal{A}_\Gamma = (D; \mathrm{Pol}(\Gamma))$ is *associated* with Γ. An algebra $\mathcal{B}$ is said to be a *factor* of the algebra $\mathcal{A}$ if $\mathcal{B}$ is a homomorphic image of a subalgebra of $\mathcal{A}$. The following theorem concerns the hardness of CSP for certain constraint languages.

Theorem 2 ([4]). *Let Γ be a constraint language and let Γ' be its core. If the algebra $\mathcal{A}^c_{\Gamma'}$ has a non-trivial factor whose term operations are only projections, then* CSP(Γ) *is* **NP***-hard.*

It has been conjectured [4] that, for any other core languages Γ, the problem CSP(Γ) is tractable, and this conjecture has been verified in many important cases (see, e.g., [2,3]).

Our first result, Theorem 3, shows that the problems from the above theorem are not only **NP**-hard, but also the corresponding optimisation problems are hard at gap location 1, which rules out the existence of PTAS for such problems. By Lemma 2, it is sufficient to prove this for core constraint languages.

Theorem 3. *Let Γ be a core constraint language. If $\mathcal{A}^c_\Gamma$ has a non-trivial factor whose term operations are only projections, then* Max CSP(Γ)-B *is hard at gap location 1.*

There are four basic ingredients in the proof of Theorem 3. The first two are Lemma 1 and the rather standard use of expander graphs to bound the number of variable occurrences (see, e.g., Section 8.4.1 of [1]). We also use the following alternative technical characterisation (obtained in the proof of Proposition 7.9 of [4]) of constraint languages satisfying the conditions of the theorem.

The *not all equal* relation contains the tuples $(0,0,1)$, $(0,1,0)$, $(1,0,0)$, $(1,1,0)$, $(1,0,1)$, and $(0,1,1)$, we denote this relation by NAE. We denote set of all singleton unary relations by C_D, that is, for a finite domain D we have $C_D = \{\{(x)\} \mid x \in D\}$.

Lemma 3. *Let Γ be a core constraint language. The following are equivalent:*

- *The algebra $\mathcal{A}^c_\Gamma$ has a non-trivial factor whose term operations are only projections.*
- *There exists a subset B of D and a surjective mapping $\varphi : B \to \{0,1\}$ such that the relational clone $\langle \Gamma \cup C_D \rangle$ contains the relation $\varphi^{-1}(NAE)$ which is the full preimage (under φ) of NAE.*

The final ingredient in the proof of Theorem 3 is that the problem Max Not-All-Equal 3Sat is hard at gap location 1, which was proved in [17].

Note that if CSP(Γ) is tractable, then Max CSP(Γ) cannot be hard at gap location 1. Hence, if the above conjecture from [4] holds, our result describes all problems Max CSP(Γ) that are hard at gap location 1.

It is not hard to see that hardness at gap location 1 rules out the existence of PTAS even when Max CSP(Γ)-B is restricted to satisfiable instances (i.e., those where all constraints can be simultaneously satisfied).

Corollary 1. *Under the assumptions of Theorem 3, there exists a constant c (depending on Γ) such that* Max CSP(Γ)-B *restricted to satisfiable instances cannot be approximated within c in polynomial time (unless* **P** = **NP***).*

The following conjecture has been made by Feder et al. [9].

Conjecture 1. For any fixed Γ such that CSP(Γ) is **NP**-complete, there is an integer k such that CSP(Γ)-k is **NP**-complete.

Under the assumption that the dichotomy conjecture (that all problems CSP(Γ) not covered by Theorem 2 are tractable) holds, an affirmative answer follows immediately from Theorem 3. So, for all constraint languages Γ such that CSP(Γ) is currently known to be **NP**-complete, it is also the case that CSP(Γ)-B is **NP**-complete.

4 Approximability of Single Relation Max CSP

A relation R is said to be *d-valid* if $(d,\ldots,d) \in R$ for $d \in D$, and simply *valid* if it is is d-valid for some $d \in D$. It was proved in [12] that every problem Max CSP($\{R\}$) with R neither empty nor valid is **NP**-hard. We strengthen this result by proving that the problems are not only **NP**-hard but also cannot have a PTAS. Note that for some Max CSP problems such approximation hardness results are known, e.g., for Max Cut and Max DiCut (see Example 1). Our result extends those hardness results to all possible relations.

Theorem 4. *Let $R \in R_D$ be non-empty. If $(d,\ldots,d) \in R$ for some $d \in D$ then* Max CSP($\{R\}$) *is trivial. Otherwise,* Max CSP($\{R\}$)-B *is hard to approximate.*

For a constraint language Γ, let Aut(Γ) denote the permutation group consisting of injective unary polymorphisms of Γ. Recall that a permutation group $\mathcal{G}$ on a set D is called *transitive* if, for every $d, d' \in D$, there exists $g \in \mathcal{G}$ such that $g(d) = d'$. A digraph $G = (V, E)$ is said to be *vertex-transitive* if the permutation group Aut($\{E\}$) is transitive.

After proving a couple of lemmas which reduce the set of relations one needs to consider to prove Theorem 4, the relations which are left are edge relations of vertex-transitive digraphs. In [16] the following characterisation of the complexity of the CSP problem for such relations was given. By deriving an algebraic characterisation of this result we can use Theorem 3 together with certain ideas and techniques (domain restriction, strict implementation) from [12] to prove Theorem 4.

Theorem 5 ([16]). *Let $G = (V, E)$ be a vertex-transitive digraph which is a core. If G is a directed cycle, then* CSP($\{E\}$) *is tractable. Otherwise,* CSP($\{E\}$) *is* **NP***-complete.*

Theorem 4 can be used to classify approximability of Max CSP(Γ) for constraint languages Γ with sufficiently many symmetries. The following result can be derived from Theorem 4.

Corollary 2. *Let Γ be a constraint language such that* Aut(Γ) *is transitive. If Γ contains a non-empty relation R which is not d-valid for all $d \in D$, then* Max CSP(Γ) *is hard to approximate. Otherwise,* Max CSP(Γ) *is trivial.*

Note that the constraint languages considered in Corollary 2 can be seen as a generalisation of vertex-transitive graphs.

5 Max CSP and Supermodularity

In this section, we present two results whose proofs make use of Theorem 4. These results strengthens earlier published results [14,15] in various ways (e.g., they apply to a larger class of constraint languages or they give approximation hardness instead of **NP**-hardness).

Recall that a poset $\mathcal{P} = (D, \sqsubseteq)$ is a *lattice* if, for every $x, y \in D$, there exist a greatest lower bound $x \sqcap y$ and a least upper bound $x \sqcup y$. The algebra $\mathcal{L} = (D; \sqcap, \sqcup)$ is a *lattice*, and $x \sqcup y = y \iff x \sqcap y = x \iff x \sqsubseteq y$. We will write $x \sqsubset y$ if $x \neq y$ and $x \sqsubseteq y$. All lattices we consider will be finite, and we will simply refer to these algebras as *lattices* instead of using the more appropriate term *finite lattices*. The *direct product* of $\mathcal{L}$, denoted by $\mathcal{L}^n$, is the lattice with domain D^n and operations acting componentwise.

Definition 10 (Supermodular function). *Let $\mathcal{L}$ be a lattice on D. A function $f : D^n \to \mathbb{R}$ is called* supermodular *on $\mathcal{L}$ if it satisfies,*

$$f(\boldsymbol{a}) + f(\boldsymbol{b}) \leq f(\boldsymbol{a} \sqcap \boldsymbol{b}) + f(\boldsymbol{a} \sqcup \boldsymbol{b}) \tag{1}$$

for all $\boldsymbol{a}, \boldsymbol{b} \in D^n$.

The *characteristic function* of a n-ary relation R over the domain D is the function $f : D^n \to \{0, 1\}$ such that $f(\boldsymbol{x}) = 1$ iff $\boldsymbol{x} \in R$. Call a relation supermodular if its characteristic function is such. The set of all supermodular relations on a lattice $\mathcal{L}$ will be denoted by $\mathrm{Spmod}_{\mathcal{L}}$ and a constraint language Γ is said to be supermodular on a lattice $\mathcal{L}$ if $\Gamma \subseteq \mathrm{Spmod}_{\mathcal{L}}$.

Supermodularity on lattices plays an important role in the study of Max CSP, as all known tractable cases of Max CSP(Γ) can be explained via this property [6,8,11,15].

The next definition follows [6].

Definition 11 (Generalised 2-monotone). *Given a poset $\mathcal{P} = (D, \sqsubseteq)$, a relation R is said to be* generalised 2-monotone *on $\mathcal{P}$ if*

$$x \in R \iff ((x_{i_1} \sqsubseteq a_{i_1}) \wedge \ldots \wedge (x_{i_s} \sqsubseteq a_{i_s})) \vee ((x_{j_1} \sqsupseteq b_{j_1}) \wedge \ldots \wedge (x_{j_s} \sqsupseteq b_{j_s}))$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $a_{i_1}, \ldots, a_{i_s}, b_{j_1}, \ldots, b_{j_s} \in D$, and either of the two disjuncts may be empty.

It is not hard to verify that generalised 2-monotone relations on some lattice are supermodular on the same lattice. For brevity, we will use the word *2-monotone* instead of generalised 2-monotone.

The following proposition is a combination of results proved in [6] and [14].

Proposition 1

- *If Γ consists of 2-monotone relations on a lattice, then* Max CSP(Γ) *can be solved in polynomial time.*
- *Let $\mathcal{P} = (D, \sqsubseteq)$ be a poset, which is not a lattice. If Γ contains all at most binary 2-monotone relations on $\mathcal{P}$, then* Max CSP(Γ) *is* **NP***-hard.*

We strengthen the second part of the above result as follows:

Proposition 2. *Let $\sqsubseteq$ be a partial order, which is not a lattice order, on D. If Γ contains all at most binary 2-monotone relations on $\sqsubseteq$, then* Max CSP(Γ)-B *is hard to approximate.*

A *diamond* is a lattice $\mathcal{L}$ on a domain D such that $|D|-2$ elements are pairwise incomparable. That is, a diamond on $|D|$ elements consist of a top element, a bottom element and $|D|-2$ elements which are pairwise incomparable. A *chain* is a lattice with total order. The following proposition is a combination of results proved in [6] and [15].

Theorem 6. *Let Γ contain all at most binary 2-monotone predicates on a lattice $\mathcal{L}$ which is either a chain or a diamond. If $\Gamma \not\subseteq \mathrm{Spmod}_{\mathcal{L}}$, then* Max CSP($\Gamma$) *is* **NP**-*hard.*

We can strengthen this result in three ways: our result applies to arbitrary lattices, we obtain hardness of approximation instead of **NP**-hardness, and we get the result for bounded occurrence instances.

Theorem 7. *Let Γ contain all at most binary 2-monotone predicates on an arbitrary lattice $\mathcal{L}$. If $\Gamma \not\subseteq \mathrm{Spmod}_{\mathcal{L}}$, then* Max CSP($\Gamma$)-$B$ *is hard to approximate.*

Acknowledgements. The authors would like to thank Gustav Nordh for comments which have improved the presentation of this paper. Peter Jonsson is partially supported by the *Center for Industrial Information Technology* (CENIIT) under grant 04.01, and by the *Swedish Research Council* (VR) under grant621-2003-3421. Andrei Krokhin is supported by the UK EPSRC grant EP/C543831/1. Fredrik Kuivinen is supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

References

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation: Combinatorial Optimization Problems and their Approximability Properties. Springer, Heidelberg (1999)
2. Bulatov, A.: Tractable conservative constraint satisfaction problems. In: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03), pp. 321–330. IEEE Computer Society Press, Los Alamitos (2003)
3. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1), 66–120 (2006)
4. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the complexity of constraints using finite algebras. SIAM J. Comput. 34(3), 720–742 (2005)
5. Burris, S., Sankappanavar, H.: A Course in Universal Algebra. Springer, Heidelberg (1981)
6. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: Supermodular functions and the complexity of Max CSP. Discrete Appl. Math. 149(1-3), 53–72 (2005)
7. Dalmau, V., Jeavons, P.: Learnability of quantified formulas. Theor. Comput. Sci. 306, 485–511 (2003)

8. Deineko, V., Jonsson, P., Klasson, M., Krokhin, A.: Supermodularity on chains and complexity of maximum constraint satisfaction. In: European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05), volume AE of DMTCS Proceedings, pp. 51–56, 2005. Discrete Mathematics and Theoretical Computer Science, Full version available as The approximability of Max CSP with fixed-value constraints, arXiv.org:cs.CC/0602075 (2005)
9. Feder, T., Hell, P., Huang, J.: List homomorphisms of graphs with bounded degrees. Discrete Math. 307, 386–392 (2007)
10. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press, Oxford (2004)
11. Jonsson, P., Klasson, M., Krokhin, A.: The approximability of three-valued Max CSP. SIAM J. Comput. 35(6), 1329–1349 (2006)
12. Jonsson, P., Krokhin, A.: Maximum H-colourable subdigraphs and constraint optimization with arbitrary weights. J. Comput. System Sci. 73(5), 691–702 (2007)
13. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM J. Comput. 30(6), 1863–1920 (2000)
14. Krokhin, A., Larose, B.: Maximum constraint satisfaction on diamonds. Technical Report CS-RR-408, University of Warwick, UK (2004)
15. Krokhin, A., Larose, B.: Maximum constraint satisfaction on diamonds. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 388–402. Springer, Heidelberg (2005)
16. MacGillivray, G.: On the complexity of colouring by vertex-transitive and arc-transistive digraphs. SIAM J. Discret. Math. 4(3), 397–408 (1991)
17. Petrank, E.: The hardness of approximation: Gap location. Computational Complexity 4, 133–157 (1994)
18. Pöschel, R., Kalužnin, L.: Funktionen- und Relationenalgebren. DVW, Berlin (1979)
19. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
20. Trevisan, L.: Inapproximability of combinatorial optimization problems, arXiv.org:cs.CC/0409043 (2004)

New Bounds for MAX-SAT by Clause Learning

Alexander S. Kulikov[1,⋆] and Konstantin Kutzkov[2]

[1] St. Petersburg Department of Steklov Institute of Mathematics
27 Fontanka, 191023 St.Petersburg, Russia
kulikov@logic.pdmi.ras.ru
[2] Department of Computer Science, University of Munich
Oettingenstr. 67, 80538 München, Germany
kutzkov@gmail.com

Abstract. To solve a problem on a given CNF formula F a splitting algorithm recursively calls for $F[v]$ and $F[\neg v]$ for a variable v. Obviously, after the first call an algorithm obtains some information on the structure of the formula that can be used in the second call. We use this idea to design new surprisingly simple algorithms for the MAX-SAT problem. Namely, we show that MAX-SAT for formulas with constant clause density can be solved in time c^n, where $c < 2$ is a constant and n is the number of variables, and within polynomial space (the only known such algorithm by Dantsin and Wolpert uses exponential space). We also prove that MAX-2-SAT can be solved in time $2^{m/5.88}$, where m is the number of clauses (this improves the bound $2^{m/5.769}$ proved independently by Kneis et al. and by Scott and Sorkin).

1 Introduction

Splitting method is one of the most popular ways of proving upper bounds for SAT and MAX-SAT problems. The simplest form of a splitting algorithm is given in Fig. 1. Obviously, without the simplification phase the worst case running time of such an algorithm is 2^n, as it just considers all possible candidates (i.e., assignments of Boolean values to all variables of a formula) to solutions.

A natural idea for reducing the running time of such an algorithm is introducing the following partial order on candidates: α is stronger than β if β cannot be a solution without α being a solution. Clearly, in such a case there is no need in considering the assignment β and thus the search space is reduced. This is actually what is done at simplification phase in many known splitting algorithms. For example, if l is a pure literal, then any assignment $\alpha \ni l$ is stronger than $(\alpha \backslash \{l\}) \cup \{\neg l\}$. Thus to prove that a given splitting algorithm is correct one has to prove that any possible candidate is either considered by the algorithm or is weaker than some other assignment. However such kind of knowledge is typically used in the same branch. That is, if in some branch an algorithm does not consider an assignment β, then there is an assignment α, which is stronger than β and is considered in the same branch.

⋆ Supported in part by INTAS (grants 04-77-7173, 05-109-5352), RFBR (grants 05-01-00932-a, 06-01-00502-a, 06-01-00584-a) and Russian Science Support Foundation.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 194–204, 2007.

Splitting Algorithm

Input: a CNF formula F.
Method.

1. Simplify F as long as it is possible.
2. If the solution for F is obvious (in particular, if F is empty), then return it.
3. Choose a variable v according to some heuristic.
4. Recursively call for $F[v]$ and $F[\neg v]$ and return the solution according to the solutions returned by both recursive calls.

Fig. 1. Form of a splitting algorithm

In this paper we show how to get rid in the current branch of the candidates considered in another branch. The idea itself is quite natural and not new: it is used, e.g., in theoretical algorithms ([1], [2]), practical solvers ([3], [4], [5]) and even in proof systems ([6]). In SAT algorithms this is usually called clause learning, which means that an algorithm stores partial assignments that make a formula unsatisfiable. In our new algorithms we use a natural extension of this simple idea, i.e., we store partial assignments that cannot be extended to an optimal (satisfying maximal possible number of clauses) assignment. We prove the following bounds (throughout all the paper we ignore polynomial factors):

- c^n for MAX-SAT for formulas with constant clause density, where $c < 2$ is a constant and n is the number of variables;
- $2^{m/5.88}$ for MAX-2-SAT, where m is the number of clauses.

Both algorithms use only polynomial space.

Overview of previous results. We are only aware of one bound better than 2^n for MAX-SAT for formulas with constant clause density. The algorithm is due to Dantsin and Wolpert [7]. This algorithm however uses exponential space.

For MAX-2-SAT, several bounds w.r.t. the number of clauses were obtained since 1999, see Table 1. Note that the bounds by Scott and Sorkin are actually

Table 1. Known bounds for MAX-2-SAT

$2^{m/2.873}$	[9]	Niedermeier and Rossmanith
$2^{m/4.000}$	[10]	Hirsch
$2^{m/5.000}$	[11]	Gramm, Hirsch, Niedermeier, and Rossmanith
$2^{m/5.000}$	[12]	Scott and Sorkin
$2^{m/5.217}$	[13]	Kneis and Rossmanith
$2^{m/5.500}$	[14]	Kojevnikov and Kulikov
$2^{m/5.769}$	[15]	Kneis, Mölle, Richter, and Rossmanith
$2^{m/5.769}$	[16]	Scott and Sorkin

applied to a wider class of problems, namely for MAX-2-CSP. All these bounds explore the splitting method, which seems to be unable to give any non-trivial upper bound for MAX-2-SAT w.r.t. the number of variables. Williams [8] uses a fast matrix multiplication algorithm in a nice way to get a $2^{n/1.261}$ bound.

2 General Setting

2.1 Main Definitions

Let V be a set of Boolean variables. The negation of a variable $v \in V$ is denoted by $\neg v$. A literal is either a variable or its negation. A clause and an assignment are sets of literals that do not contain any variable together with its negation. A formula in conjunctive normal form (CNF) is a multi-set of clauses. A total assignment is an assignment to all variables of a formula. We say that a clause C is satisfied by an assignment α, if $C \cap \alpha \neq \emptyset$; however, we say that C is falsified by α, if $\forall l \in C$, $\neg l \in \alpha$. By $V(\alpha)$ we denote the set of variables of α. By $\mathrm{Cl}(F, \alpha)$ we denote the number of clauses of F satisfied by α and by $\mathrm{MCl}(F)$ we denote the maximal number of simultaneously satisfiable clauses (thus, $\mathrm{MCl}(F) = \max_\alpha \mathrm{Cl}(F, \alpha)$). The SAT problem asks whether there exists an assignment that satisfies all clauses of a given formula. The MAX-SAT problem asks for an assignment that satisfies $\mathrm{MCl}(F)$ clauses.

A d-literal is a literal occurring exactly d times in a formula. A d^+-literal appears at least d times. A (d_1, d_2)-literal appears d_1 times positively and d_2 times negatively. Other literal types are defined similarly.

Let F be a CNF formula, v be a variable of F, α be an assignment to variables of F. We usually write assignments $\{v\}$ and $\{\neg v\}$ just as v and $\neg v$. By $F[\alpha]$ we denote the formula resulting from F by first removing all clauses satisfied by α and then removing all literals l such that $\neg l \in \alpha$ from the remaining clauses. Let also

$$F_v = \{C : C \cup \{v\} \in F\}, \;\; F_{\neg v} = \{C : C \cup \{\neg v\} \in F\}, \;\; F_{-v} = \{C \in F : v \notin C\} \;\; ,$$

then clearly

$$F[v] = F_{\neg v} \cup F_{-v}, \;\; F[\neg v] = F_v \cup F_{-v} \;\; .$$

For assignments α and β such that $V(\beta) \subseteq V(\alpha)$, by α^β we denote the assignment resulting from α by changing the values of $V(\beta)$ in accordance with β. For example, α^a assigns the value True to a.

Let α be a total assignment to a formula F and a be an (i, j)-literal of F. Then,

$$\mathrm{Cl}(F, \alpha^a) = i + \mathrm{Cl}(F[a], \alpha) = i + \mathrm{Cl}(F_{\neg a}, \alpha) + \mathrm{Cl}(F_{-a}, \alpha) \;\; , \tag{1}$$

$$\mathrm{Cl}(F, \alpha^{\neg a}) = j + \mathrm{Cl}(F[\neg a], \alpha) = j + \mathrm{Cl}(F_a, \alpha) + \mathrm{Cl}(F_{-a}, \alpha) \;\; . \tag{2}$$

By $n(F)$ and $m(F)$ we denote, respectively, the number of variables and clauses in a formula F (we usually omit F, when it is clear from the context).

2.2 Splitting Algorithms

A splitting algorithm usually first somehow simplifies an input formula F and then recursively calls for several formulas of the form $F[\alpha]$ (this is called splitting). To estimate the running time of a splitting algorithm one has to specify a formula complexity measure $\gamma(F)$. A splitting number of a splitting $F[\alpha_1]$, $\ldots, F[\alpha_k]$ is defined to be the unique positive root of the equation $1 = \sum_{i=1}^{k} x^{-a_i}$, where $a_i = \gamma(F) - \gamma(F[\alpha_i])$, and is denoted by $\tau(a_1, \ldots, a_k)$. It is known [17] that the running time of a splitting algorithm on a formula F does not exceed $\tau_{\max}^{\gamma(F)}$, where $\tau_{\max}$ is the maximal splitting number of this algorithm. Thus, to prove upper bounds on the running time it suffices to prove upper bounds on $\tau_{\max}$. There are several simple properties of splitting numbers [17]. In particular,

- $\tau(a, a) = 2^{1/a}$;
- $\tau(a, b) \leq 2^{1/\sqrt{ab}}$;
- $\tau(a_1, \ldots, a_p) \leq \tau(b_1, \ldots, b_p)$, if $\forall i, a_i \geq b_i$;
- $\tau(1, 2, \ldots, a) < 2$ and $\tau(1, 2, \ldots, a) < \tau(1, 2, \ldots, a + 1)$;
- $\tau(1, a) \to 1$ as $a \to \infty$.

To prove that a splitting algorithm is correct one has to prove that this algorithm considers all possible candidates to solutions (e.g., if an algorithm for SAT returns the answer "Unsatisfiable", then one has to guarantee that it considered all possible total assignments and none of them is a satisfying assignment). To achieve this goal splitting algorithms usually split on a set of assignments $\alpha_1, \ldots, \alpha_k$, such that any possible total assignment is an extension of one of α_i's. However if we know that a total assignment β cannot be a solution without a total assignment α being a solution, then there is no need to consider β. For example, if l is a pure literal, then there is no sense in considering assignments containing $\neg l$. We formalize this simple idea in the following definition (a similar definition is used in [18]).

Let α_1 and α_2 be assignments to variables of a formula F. We say that α_1 is stronger w.r.t. F than α_2 and write $\alpha_1 \succeq_F \alpha_2$, if for any total assignment β, β^{α_2} can be a solution to a problem for F only if β^{α_1} is a solution (again, we omit F if this does not lead to ambiguity). By a solution for SAT we mean a satisfying assignment, while a solution for MAX-SAT is an assignment satisfying the maximal number of clauses.

3 A New Algorithm for MAX-SAT

In this section we show that MAX-SAT for formulas with at most Δn clauses, where Δ is a constant, can be solved in time c^n, where $c < 2$ is a constant.

First, let us give some informal ideas showing that there exists a SAT algorithm with running time c^n, where $c < 2$, for formulas with constant clause density Δ. The algorithm works as follows. If a formula contains a (d^+, d^+)-literal l, for some big constant $d = d(\Delta)$, it just splits on it. This provides a good splitting number w.r.t. the number of clauses (in both branches at least d clauses are eliminated). Otherwise, the algorithm picks a d^--literal a and checks

the satisfiability of F_a. This can be easily done in polynomial time: at each step we choose a literal (of course, we assume that it is not pure) and split the current formula into two formulas having fewer clauses; since F_a contains at most d clauses, the size of the resulting splitting tree is at most 2^d. If F_a is unsatisfiable, then so is $F[\neg a] = F_a \cup F_{-a}$ and we can just assign the value True to a. Otherwise, we find a satisfying assignment to F_a. It is easy to see that it contains at most d literals. Let us denote it by $\alpha = \{l_1, \ldots, l_k\}$ $(k \leq d)$.

Now we recursively call for $F[\neg a] = F_a \cup F_{-a}$. If the answer is "Satisfiable" we immediately return. Otherwise we can conclude that α cannot be extended to a satisfying assignment of F_{-a}, which means that we do not need to consider extensions of α in the a-branch. Thus, we split $F[a]$ to

$$F[a, \neg l_1], F[a, l_1, \neg l_2], \ldots, F[a, l_1, \ldots, l_{k-1}, \neg l_k] \ .$$

The overall splitting number w.r.t. n is $\tau(1, 2, \ldots, k+1) < 2$. Thus, the algorithm is always able to find a good splitting w.r.t. either n or m. Since $m \leq \Delta n$, we can prove that the overall running time of this algorithm is c^n, where $c < 2$.

The formal description of the algorithm for MAX-SAT is given in Fig. 2. The algorithm first applies the pure literal rule as long as it is applicable. Then it splits on a D^+-literal, if such literal exists. If there is no such literal, the algorithm just selects any literal a and finds optimal assignments α_a and $\alpha_{\neg a}$ for F_a and $F_{\neg a}$, respectively. Since both F_a and $F_{\neg a}$ contain at most D clauses, this can be done in polynomial time. Moreover, we can assume that both α_a and $\alpha_{\neg a}$ contain at most D literals. Finally, the algorithm splits a formula using the literals of one of the found assignments. We do not explain here how the algorithm constructs the answer from the answers returned by the recursive calls, as this is a standard operation (all one needs to do is to count the number of currently satisfied clauses; see, e.g., [10]).

Theorem 1. *For any constant $\Delta > 0$ there exist constants $D > 0$ and $c < 2$ such that MaxSatAlg(D) returns* $\mathrm{MCl}(F)$ *for any formula F with at most $\Delta n(F)$ clauses in time $c^{n(F)}$.*

Proof. If a current formula F contains a D^+-literal, the algorithm just splits on it. This removes one variable and at least one clause in one branch and one variable and at least D clauses in the other branch (remind that F does not contain pure literals). Now consider the case when F consists of D^--literals only. Assume w.l.o.g. that $i + k_{\neg a} \geq j + k_a$ (remind that a is an (i, j)-literal). We claim that $\alpha_{\neg a} \cup a \succeq_F \alpha_{\neg a} \cup \neg a$. Indeed, consider any total assignment β to F. By (1),

$$\mathrm{Cl}(F, \beta^{\alpha_{\neg a} \cup a}) = i + k_{\neg a} + \mathrm{Cl}(F_{-a}, \beta^{\alpha_{\neg a}}) \ ,$$

while by (2),

$$\mathrm{Cl}(F, \beta^{\alpha_{\neg a} \cup \neg a}) \leq j + k_a + \mathrm{Cl}(F_{-a}, \beta^{\alpha_{\neg a}}) \ .$$

So, $\beta^{\alpha_{\neg a} \cup a}$ satisfies at least as many clauses of F as $\beta^{\alpha_{\neg a} \cup \neg a}$ (for any β) and we conclude that $\alpha_{\neg a} \cup a \succeq_F \alpha_{\neg a} \cup \neg a$. Thus, we do not need to consider any extension of $\alpha_{\neg a}$ in the $\neg a$-branch. Thus, we can split as follows:

$$F[a], F[\neg a, \neg y_1], F[\neg a, y_1, \neg y_2], \ldots, F[\neg a, y_1, \ldots, y_{q-1}, \neg y_q] \ .$$

Algorithm MaxSatAlg

Parameters: a positive real number D.
Input: a CNF formula F.
Output: MCl(F).
Method.

1. Assign the value True to all pure literals of F.
2. If F is empty, then return 0.
3. If F contains a D^+-literal a, then recursively call for $F[a]$ and $F[\neg a]$ and return the answer.
4. Let a be any literal of F. Let also $i = d(a)$, $j = d(\neg a)$.
5. Find optimal assignments for F_a and $F_{\neg a}$. Denote them by $\alpha_a = \{x_1, \ldots, x_p\}$ and $\alpha_{\neg a} = \{y_1, \ldots, y_q\}$, respectively, and let

$$k_a = \mathrm{MCl}(F_a) = \mathrm{Cl}(F, \alpha_a) \ ,$$

$$k_{\neg a} = \mathrm{MCl}(F_{\neg a}) = \mathrm{Cl}(F, \alpha_{\neg a}) \ .$$

6. If $i + k_{\neg a} \geq j + k_a$, then recursively call for

$$F[a], F[\neg a, \neg y_1], F[\neg a, y_1, \neg y_2], \ldots, F[\neg a, y_1, \ldots, y_{q-1}, \neg y_q]$$

   and return the answer.
7. Otherwise recursively call for

$$F[\neg a], F[a, \neg x_1], F[a, x_1, \neg x_2], \ldots, F[a, x_1, \ldots, x_{p-1}, \neg x_p]$$

   and return the answer.

Fig. 2. An algorithm for MAX-SAT

If $i + k_{\neg a} < j + k_a$, then by using exactly the same argument one can show that $\alpha_a \cup \neg a \succeq_F \alpha_a \cup a$. This ensures that MaxSatAlg is correct.

Now let us estimate the running time. We use the following complexity measure (for $w = w(\Delta)$ defined later):

$$\gamma(F) = \begin{cases} n(F) + wm(F), & \text{if } F \text{ contains a } D^+\text{-literal,} \\ n(F), & \text{otherwise.} \end{cases}$$

If a formula contains a D^+-literal, the algorithm splits with a splitting number at most $\tau(1 + w, 1 + wD)$. Otherwise, the splitting number is at most $r_D = \tau(1, 2, \ldots, D + 1)$ (as $p, q \leq D$). Now let d be any constant bigger than Δ. Assume now that there exists constants w and D, such that

$$\tau(1 + w, 1 + wD) \leq 2^{\frac{1}{1+wd}} \text{ and } \tau(1, \ldots, D + 1) \leq 2^{\frac{1}{1+wd}} \ .$$

Then the running time of the algorithm is bounded by

$$2^{\frac{\gamma}{1+wd}} \leq 2^{\frac{n+wm}{1+wd}} \leq 2^{n\frac{1+w\Delta}{1+wd}} = c^n \ ,$$

where $c = 2^{\frac{1+w\Delta}{1+wd}} < 2$ is a constant. Below we prove that such w and D exist.

It is easy to see that for any integer $D \geq 1$ there exists $w_D > 0$ such that $\tau(1,\ldots,D+1) = 2^{\frac{1}{1+w_D d}}$ (since $\tau(1,\ldots,D+1) < 2$). Thus, it is sufficient to find an integer D such that $\tau(1+w_D, 1+w_D D) \leq 2^{\frac{1}{1+w_D d}}$. To show this it suffices to show that $(1+w_D d)^2 \leq (1+w_D)(1+w_D D)$ (since $\tau(a,b) \leq 2^{\frac{1}{\sqrt{ab}}}$). The last inequality is equivalent to $w_D d^2 + 2d \leq D+1+w_D D$, which is obviously satisfied for large enough D. □

4 MAX-2-SAT

In this section we present a new algorithm for MAX-2-SAT with a running time $2^{m/5.88} \approx 1.12512^m$. It extends an algorithm by Kojevnikov and Kulikov [14] by using the idea of clause learning. The analysis is mostly the same as in [14] with several cases improved. The algorithm first simplifies an input formula by applying (as long as it is possible) to it *almost common clauses*, *pure literal*, *dominating unit clause*, and *frequently meeting variables* rules. We do not describe here the rules in detail (see [14]), but provide several important properties of simplified formulas in Lemma 1, where by a simplified formula we mean a formula for which no rule described above is applicable. After simplifying the algorithm just splits a resulting formula by assigning Boolean values to some variables.

Before stating some known properties of this algorithm we give some additional definitions. For a literal l, by $d_1(l)$ we denote the number of unit clauses (l) in a formula and by $d_2(l)$ we denote the number of 2-clauses containing the literal l. By weight of a variable x we mean $d_2(x) + d_2(\neg x)$.

Lemma 1 ([14])

- *A simplified formula does not contain variables of weight at most* 2.
- *A simplified formula does not contain clauses of the form* (xy) *and* $(\neg xy)$.
- *If a variable* x *occurs at most one time without a variable* y*, then a formula can be simplified.*
- *If* F *is a simplified formula,* $(xy) \in F$ *and* y *has weight* 3*, then at least in one of the formulas* $F[x]$ *and* $F[\neg x]$ *the simplification rules assign a Boolean value to* y.
- *If a simplified formula* F *contains only variables of weight* 3*, then there is a variable* x*, such that both* $F[x]$ *and* $F[\neg x]$ *(after simplifying) contain at least* 6 *variables fewer than* F.

The running time of the algorithm by Kojevnikov and Kulikov is estimated w.r.t. the following complexity measure:

$$\gamma(F) = N_3 + 1.9 \cdot N_4(F) + \sum_{i \geq 5} \frac{i \cdot N_i(F)}{2} \ ,$$

where $N_i(F)$ is the number of variables of weight i of F. We change the coefficients as follows:

$$\gamma(F) = 0.98 \cdot N_3 + 1.96 \cdot N_4(F) + \sum_{i \geq 5} \frac{i \cdot N_i(F)}{2} \ .$$

Thus, we have to prove that the algorithm always splits a simplified formula with a splitting number at most $\tau(5.88, 5.88)$. Note that reducing the weight of any variable in a simplified formula reduces γ at least by 0.5 (as the difference between any two coefficients before N_i is at least 0.5). First consider the case, when a formula F contains a variable x of weight $w \geq 6$. In this case we can just split on this variable. This reduces γ at least by w in the both branches: eliminating x reduces γ at least by $w/2$, decreasing the weight of neighbors of x further reduces γ at least by $w/2$.

Now consider a case when F contains only variables of weight at most 4. Let x be a variable of weight 4. For $1 \leq i \leq j \leq 4$, let k_{ij} denote the number of neighbors of x that have weight j and occur i times with x. It is easy to see that such a neighbor of x becomes a variable of weight $(j - i)$ after assigning a Boolean value to x. By Lemma 1, $k_{ij} = 0$ for $j \leq 2$ and for $j - i \leq 1$. So, only k_{13}, k_{14} and k_{24} can be positive. Since x is a variable of weight 4, $k_{13} + k_{14} + 2k_{24} = 4$. Now let F' be a formula obtained from F by assigning a Boolean value to x. Then,

$$\begin{aligned}\gamma(F) - \gamma(F') = (0.98 \cdot k_{13} + 1.96 \cdot (k_{14} + k_{24} + 1)) - 0.98 \cdot k_{14} = \\ 0.98 \cdot (k_{13} + k_{14} + 2k_{24}) + 1.96 = 0.98 \cdot 4 + 1.96 = 5.88 \ .\end{aligned}$$

If F contains only variables of weight at most 3, then by Lemma 1 we can always find a $(6, 6)$-splitting w.r.t. n. This gives a $(5.88, 5.88)$-splitting w.r.t. γ, since $\gamma(F) = 0.98 \cdot n(F)$. The only remaining case is when a formula contains at least one variable of weight 5 and all other variables have weight at most 5 and this is the case where we explore clause learning.

First, let us calculate a straightforward splitting number. It is done similarly to the case with a variable of weight 4:

$$k_{13} + k_{14} + k_{15} + 2k_{24} + 2k_{25} + 3k_{35} = 5 \ ,$$

$$\begin{aligned}\gamma(F) - \gamma(F') = \\ (0.98 \cdot k_{13} + 1.96 \cdot (k_{14} + k_{24}) + 2.5 \cdot (k_{15} + k_{25} + k_{35} + 1)) \\ -(0.98 \cdot (k_{14} + k_{25}) + 1.96 \cdot k_{15}) \\ = 0.54 \cdot (k_{13} + k_{14} + k_{15} + 2k_{24} + 2k_{25} + 3k_{35}) + 2.5 \\ +0.44 \cdot (k_{13} + k_{14} + 2k_{24} + k_{25} + 2k_{35}) \\ = 5.2 + 0.44 \cdot (k_{13} + k_{14} + 2k_{24} + k_{25} + 2k_{35}) \ .\end{aligned}$$

Thus, if

$$(k_{13} + k_{14} + 2k_{24} + k_{25} + 2k_{35}) \geq 2 \tag{3}$$

then we already have a required splitting number. This, in particular, means that we do not need to consider cases when either $k_{24} > 0$ or $k_{35} > 0$.

The idea of clause learning is used in the following lemma, which says that for any two neighbors of a we do not need to consider extensions of some assignment to these neighbors in one of the branches.

Lemma 2. *Let F be a simplified 2-CNF formula, a be a variable of weight 5 and l, l' be literals appearing with a. Then either*

$$\{a, l_1, l_2\} \succeq_F \{\neg a, l_1, l_2\} \text{ or } \{\neg a, l_1, l_2\} \succeq_F \{a, l_1, l_2\} \ ,$$

where l_1 is either l or $\neg l$ and l_2 is either l' or $\neg l'$.

Proof. Let F contain i_2 2-clauses with literal a, j_2 2-clauses with $\neg a$, i_1 clauses (a) and j_1 clauses $(\neg a)$. Note that F_a and $F_{\neg a}$ consist of i_2 and j_2 unit clauses, respectively.

Now, from the fact that $5 = (i_2 + j_2) = (i_1 + i_2 - j_1) + (j_1 + j_2 - i_1)$ we conclude that either $(i_1 + i_2 - j_1)$ or $(j_1 + j_2 - i_1)$ is at least 3. W.l.o.g. we assume that $i_1 + i_2 - j_1 \geq 3$.

Let α be a total assignment to F. Then, by (1) and (2),

$$\mathrm{Cl}(F, \alpha^a) = i_1 + i_2 + \mathrm{Cl}(F_{\neg a}, \alpha) + \mathrm{Cl}(F_{-a}, \alpha) \ ,$$

$$\mathrm{Cl}(F, \alpha^{\neg a}) = j_1 + j_2 + \mathrm{Cl}(F_a, \alpha) + \mathrm{Cl}(F_{-a}, \alpha) \ .$$

Thus,

$$\begin{aligned} \mathrm{Cl}(F, \alpha^a) - \mathrm{Cl}(F, \alpha^{\neg a}) &= i_1 + i_2 - j_1 - j_2 + \mathrm{Cl}(F_{\neg a}, \alpha) - \mathrm{Cl}(F_a, \alpha) \\ &\geq 3 - j_2 + \mathrm{Cl}(F_{\neg a}, \alpha) - \mathrm{Cl}(F_a, \alpha) \end{aligned}$$

Note that $\mathrm{Cl}(F_{\neg a}, \alpha) - \mathrm{Cl}(F_a, \alpha) \geq -i_2$, as α satisfies at least 0 clauses of $F_{\neg a}$ and at most i_2 clauses of F_a. If (l) or $(\neg l)$ is in $F_{\neg a}$, we set l_1 to be a literal satisfying this unit clause. Otherwise, if (l) or $(\neg l)$ is in F_a, we set l_1 so that it falsifies the corresponding clause. The literal l_2 is defined similarly. Then, for any $\alpha \ni l_1, l_2$, $\mathrm{Cl}(F_{\neg a}, \alpha) - \mathrm{Cl}(F_a, \alpha) \geq -i_2 + 2$, and hence $\mathrm{Cl}(F, \alpha^a) - \mathrm{Cl}(F, \alpha^{\neg a}) \geq 3 - j_2 - i_2 + 2 \geq 0$. Thus, we have shown that for any total assignment α such that $a, l_1, l_2 \in \alpha$, $\mathrm{Cl}(F, \alpha^a) \geq \mathrm{Cl}(F, \alpha^{\neg a})$, which means that $\{a, l_1, l_2\} \succeq_F \{\neg a, l_1, l_2\}$. □

Let us give an example of using this lemma. Suppose that F contains the following five clauses: $(al_1)(al_2)(al_3)(\neg al_4)(\neg al_5)$ (and does not contain any other clauses with variable a). Lemma 2 implies that $\{a, \neg l_2, l_4\} \succeq_F \{\neg a, \neg l_2, l_4\}$. Indeed, any extension of $\{\neg a, \neg l_2, l_4\}$ satisfies *at most* 4 of the mentioned above clauses (as (al_2) is falsified), while any extension of $\{a, \neg l_2, l_4\}$ satisfies *at least* 4 of these clauses.

Now we consider all the remaining cases. Remind that F contains only variables of weight at most 5 and let a be a variable of weight exactly 5. First consider the case when a has a neighbor b of degree 3. By (3), $k_{13} = 1$ and $k_{15} = 4$. By

Lemma 1, b is assigned a Boolean value in at least one of the branches $F[a]$ and $F[\neg a]$, moreover all three neighbors of b are different variables. Thus, splitting on a provides a splitting number not exceeding $\tau(5.2+0.44, 5.2+0.44+2\cdot 0.54) < 1.12$.

Suppose that a appears with some variable b twice. This means, by (3), that $k_{25} = 1$ and $k_{15} = 3$. Lemma 2 implies that $\{l_a, l_b\} \succeq_F \{\neg l_a, l_b\}$, where l_a and l_b are literals of a and b. Thus, we can assign a Boolean value to b in one of the branches $F[a]$ and $F[\neg a]$. The corresponding splitting number is $\tau(5.2 + 0.44, 5.2 + 0.44 + 0.98) < 1.121$.

Now we know that a has exactly five different neighbors. Moreover, by (3) at most one of them has weight at least 4, while all other have weight exactly 5 (in other words, $k_{14} \leq 1$, $k_{15} \geq 4$, $k_{14} + k_{15} = 5$). W.l.o.g. we can assume that $d_1(a) + d_2(a) - d_1(\neg a) \geq 3$ (otherwise, we consider $\neg a$ instead of a). Let b_1, b_2 be any two literals appearing together with the literal a and c be any literal appearing together with the literal $\neg a$. Lemma 2 then implies that

$$\{a, \neg b_1, \neg b_2\} \succeq_F \{\neg a, \neg b_1, \neg b_2\} \ ,$$

$$\{a, \neg b_1, c\} \succeq_F \{\neg a, \neg b_1, c\} \ .$$

Thus we can split F to $F[a]$, $F[\neg a, b_1]$, $F[\neg a, \neg b_1, b_2, \ldots, b_{i_2}, \neg c_1, \ldots, \neg c_{j_2}]$, where $b_1, \ldots, b_{i_2}$ and $c_1, \ldots, c_{j_2}$ are all literals appearing together with the literals a and $\neg a$, respectively. So all we have to do is to show that this splitting gives good splitting number in all the remaining cases. First, rename the literals so that the splitting above corresponds to a splitting

$$F[a], F[\neg a, l_1], F[\neg a, \neg l_1, l_2, l_3, l_4, l_5] \ .$$

Let l_1, l_2, l_3, l_4 be literals of variables of weight 5 variables and l_5 be literal of a variable of weight d, where $d \in \{4, 5\}$.

Assume that l_1 has at least three neighbors of degree less than 5 in $F[\neg a]$. Then the straightforward splitting number is $\tau(5.2 + 0.44 \cdot (5 - d), 5.2 + 0.44 \cdot (5 - d) + 1.96 + 3 \cdot 0.98 + 0.54, 5.2 + 0.44 \cdot (5 - d) + 5 \cdot 1.96 + (d - 5) \cdot 0.98)$. In both cases it does not exceed 1.1249. Otherwise, the neighborhood of $\{l_1, l_2, l_3, l_4\}$ in $F[\neg a]$ contains at least 8 literals of variables of weight 5 (and hence these are different from l_1, l_2, l_3, l_4, l_5). In this case it is not difficult to show that the splitting number does not exceed $\tau(5.2 + 0.44 \cdot (5 - d), 5.2 + 0.44 \cdot (5 - d) + 1.96 + 4 \cdot 0.54, 5.2 + 0.44 \cdot (5 - d) + 5 \cdot 1.96 + (d - 5) \cdot 0.98 + 2.94)$, hence, 1.1246.

References

1. Williams, R.: On computing k-CNF formula properties. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 330–340. Springer, Heidelberg (2004)
2. Robson, J.: Algorithms for maximum independent sets. Journal of Algorithms 7(3), 425–440 (1986)
3. Marques-Silva, J., Sakallah, K.: Grasp: a search algorithm for propositional satisfiability. IEEE Transactionon Computers 48(5), 506–521 (1999)

4. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, pp. 530–535 (2001)
5. Zhang, H.: Sato: An efficient propositional prover. In: McCune, W. (ed.) Automated Deduction - CADE-14. LNCS, vol. 1249, pp. 272–275. Springer, Heidelberg (1997)
6. Beame, P., Impagliazzo, R., Pitassi, T., Segerlind, N.: Memoization and DPLL: formula caching proof systems. In: Proceedings of 18th IEEE Annual Conference on Computational Complexity, pp. 248–259. IEEE Computer Society Press, Los Alamitos (2003)
7. Dantsin, E., Wolpert, A.: MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 266–276. Springer, Heidelberg (2006)
8. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoretical Computer Science 348, 357–365 (2005)
9. Niedermeier, R., Rossmanith, P.: New upper bounds for MaxSat. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 575–585. Springer, Heidelberg (1999)
10. Hirsch, E.A.: A $2^{K/4}$-time algorithm for MAX-2-SAT: Corrected version. ECCC Report TR99-036, Revision 02 (2000)
11. Gramm, J., Hirsch, E.A., Niedermeier, R., Rossmanith, P.: Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. Discrete Applied Mathematics 130(2), 139–155 (2003)
12. Scott, A., Sorkin, G.: Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 382–395. Springer, Heidelberg (2003)
13. Kneis, J., Rossmanith, P.: A new satisfiability algorithm with applications to Max-Cut. Technical Report AIB2005-08, Dept. of Computer Science, RWTH Aachen University (2005)
14. Kojevnikov, A., Kulikov, A.S.: A new approach to proving upper bounds for MAX-2-SAT. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 11–17. ACM Press, New York (2006)
15. Kneis, J., Moelle, D., Richter, S., Rossmanith, P.: Algorithms based on the treewidth of sparse graphs. In: Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science. LNCS, pp. 385–396 (2005)
16. Scott, A.D., Sorkin, G.B.: Linear-programming design and analysis of fast algorithms for Max 2-SAT and Max 2-CSP. Discrete Optimization 2006 (to appear)
17. Kullmann, O., Luckhardt, H.: Algorithms for SAT/TAUT decision based on various measures. Preprint (1998)
18. Kulikov, A.S.: Automated generation of simplification rules for SAT and MAXSAT. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 430–436. Springer, Heidelberg (2005)

Towards Hierarchical Clustering
(Extended Abstract)

Mark Sh. Levin

Inst. for Information Transmission Problems,
Russian Academy of Sciences, Moscow 127994, Russia
mslevin@acm.org

Abstract. In the paper, new modified agglomerative algorithms for hierarchical clustering are suggested. The clustering process is targeted to generating a cluster hierarchy which can contain the same items in different clusters. The algorithms are based on the following additional operations: (i) building an ordinal item pair proximity ('distance') including the usage of multicriteria approaches; (ii) integration of several item pair at each stage of the algorithms; and (iii) inclusion of the same items into different integrated item pairs/clusters. The suggested modifications above are significant from the viewpoints of practice, e.g., design of systems architecture for engineering and computer systems.

Keywords: Hierarchical clustering, agglomerative algorithm, hierarchy, system architecture, multicriteria analysis.

1 Introduction

Clustering problems is a basic scientific problem and has been studied many years (*e.g.*, [2], [4], [9], [10], [16], [20], [22], [23], [27], [30], [33], [35], [39], [50], [55], [61], [62], [66]). In recent years a set of survey works on clustering algorithms was published ([1], [4], [12], [24], [26], [31], [33], [49], [53], [64]). Generally, the following basic clustering strategies (styles) are pointed out (*e.g.*, [24], [31]): (1) *partitioning* (or *k-clustering*) (*e.g.*, [2], [37]) and graph partitioning using pagerank vector [3]; (2) *hierarchical clustering* including (2.1) single-link, (2.2) average-link, (2.3) complete-link (*e.g.*, [34], [59]); (3) graph-theoretic methods (*e.g.*, via minimal spanning tree [63]) (*e.g.*, [5], [29], [61]); (4) combinatorial optimization methods (*e.g.*, [15], [48], [54]); (5) conceptual clustering (*e.g.*, [8], [14], [45], [46], [60]); and (7) AI methods (e.g., neural networks, knowledge bases) (*e.g.*, [30], [51], [57], [58]); (8) evolutionary approaches, genetic algorithms (*e.g.*, [6], [17], [18], [36], [40], [44]); (8) approximation clustering (based on semidefinite programming methods) [47]); and (9) fixed points approach [20].

In last decade, many studies are targeted to hierarchical clustering, for example: (i) graph-based hierarchical clustering [32], (ii) lattice-based hierarchical clustering [43], (iii) likehood based hierarchical clustering [11], (iv) energy efficient hierarchical clustering [7], and (v) hierarchical clustering using dynamic modeling [38], and (vi) model-based Gaussian hierarchical clustering [25]. In

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 205–215, 2007.

addition, it is reasonable to point out contemporary studies multiple criteria classification problems based on multicriteria decision making methods (*e.g.*, [13], [65])

This paper describes new modifications of agglomerative algorithm for hierarchical clustering. The suggested algorithms are based on the following additional operations: (i) building an ordinal item pair proximity ('distance') including the usage of multicriteria approaches; (ii) integration of several item pair at each stage of the algorithms; and (iii) inclusion of the same items into different integrated item pairs/clusters. Thus the clustering process is targeted to generating a cluster hierarchy which can contain the same items in different clusters. The clustering process above (*i.e.*, hierarchy) can correspond to a structure (or architecture) of a modular (composite) hierarchical system and this structure is a significant result in system analysis/design of engineering and computer systems. Complexity of basic considered algorithms is polynomial.

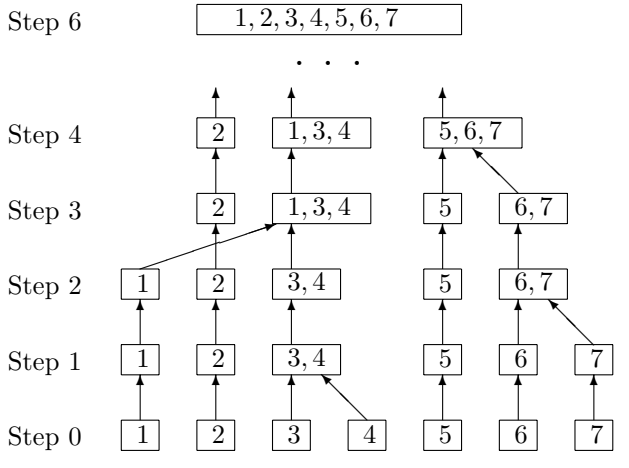


Fig. 1. Example of hierarchical clustering

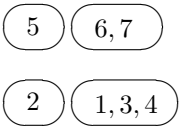


Fig. 2. Clusters for *Step 3*

2 Agglomerative Algorithm

2.1 Basic Algorithm

There is a set of n elements $A = \{A_1, ..., A_i, ..., A_n\}$ and a corresponding vector estimate of m attributes/parameters $(T_1, ..., T_j, ..., T_m)$ for each element i: $z_i = (z_{i,1}, ..., z_{i,j}, ..., z_{i,m})$. The basic agglomerative algorithm (polynomial, *algorithm 1*) is as follows (Bottom-Up element pair integration process):

Stage 1. Computing the matrix of element pair $\forall(A(i_1), A(i_2))$, $A(i_1) \in A$, $A(i_2) \in A$, $i_1 \neq i_2$ "distances" (a simple case, metric l_2):

$$d_{i_1 i_2} = \sqrt{\sum_{j=1}^{m} (z_{i_1,j} - z_{i_2,j})^2}.$$

Stage2. Revelation of the smallest pair "distance" and integration of the corresponding two elements into a resultant "integrated" element.

Stage 3. Stopping process or re-computing the matrix of pair "distances" and *Go To Stage 2.*

As result, a tree-like structure for the element pair integration process (Bottom-Up) is obtained (one element pair integration at each integration step). A basic procedure for aggregation of items (aggregation as average values) is as follows ($J_{i_1,i_2} = A_{i_1} \& A_{i_2}$): $\forall j \quad z_{J_{i_1,i_2},j} = \frac{z_{i_1,j}+z_{i_2,j}}{2}$. The item pair aggregation process can be based on other functions (*e.g.*, max, min). Integration of several items can be considered analogically. An illustrative example of hierarchical clustering is depicted in Fig. 1. Evidently, the number of integration steps equals $O(n)$ (complexity of *algorithm 1* equals $O(n^3m)$). At each step of the algorithm clusters without intersection are obtained. Fig. 2 depicts clusters for step 3 in Fig. 1.

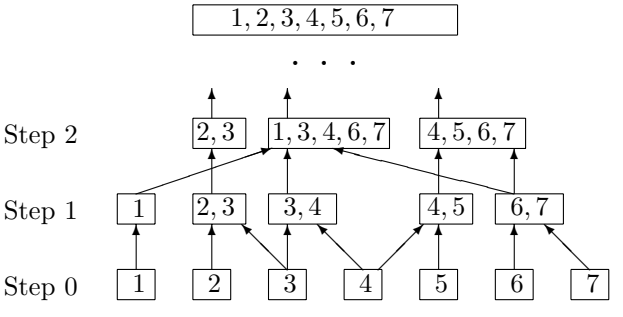


Fig. 3. Illustration for hierarchy

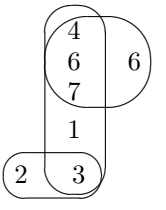


Fig. 4. Clusters for *Step 2*

2.2 Properties and Improvement Ideas

Let us point out some properties of the considered clustering process as follows:

1. The matrix of unit pair "proximity" can contain several "minimal" elements. Thus there are problems as follows: (i) selection of the "best" unit pair for integration; (ii) possible integration of several unit pair at each algorithm stage.

2. Computing the matrix of element pair "distances" often does not correspond to the problem context and it is reasonable to consider a "softer" approach for computing element pair "proximity".

3. The obtained structure of the clustering process is a tree. Often the clustering problem is used to get a system structure that corresponds to the abovementioned clustering process (*e.g.*, evolution trees, system architecture). Thus, it is often reasonable to organize the clustering process as a hierarchy, *e.g.*, for modular systems in which the same modules can be integrated into different system components/parts.

Fig. 3 illustrates concurrent integration of unit pairs at the same step of the algorithm when some units can be integrated into different system components parts, i.e., obtaining a hierarchical system structure (common modules/parts, *e.g.*, 3 and 4). In this case, obtained clusters can have intersections (Fig. 4.) Fig. 4 depicts clusters for step 2 in Fig. 3.

3 Modifications

3.1 Ordinal Unit Pair Proximity

Computing the item "distance" based on metric l_2 is a 'simplified' mathematical approach. Now let us consider two methods based on an ordinal scale $[0, 1, ..., k]$, 0 corresponds to "equal" items pair. The first method **interval dividing scheme** is the following. The mapping process is: $\forall(i_1, i_2) \;\; d_{i_1,i_2} \Rightarrow r_{i_1,i_2}$, where $r_{i_1,i_2} \in [0, ..., k]$. Fig. 5 illustrates the method. Let $a, b, v, w, p, q, e, f \in A$. Interval $(\mu = \min_{(i_1,i_2)}\{d_{i_1,i_2}\}, M = \max_{(i_1,i_2)}\{d_{i_1,i_2}\})$ is divided into k+1 subintervals $0, 1, 2, ..., k$. As a result, $r_{a,b} = 0$ if $d_{a,b} \in [0, \frac{M-\mu}{k+1}]$, $r_{p,q} = 1$ if $d_{p,q} \in [\frac{M-\mu}{k+1}, \frac{2(M-\mu)}{k+1}]$, etc. It is reasonable to consider versions of the method: (i) equal subintervals, (ii) non-equal subintervals (dividing process is based on a context).

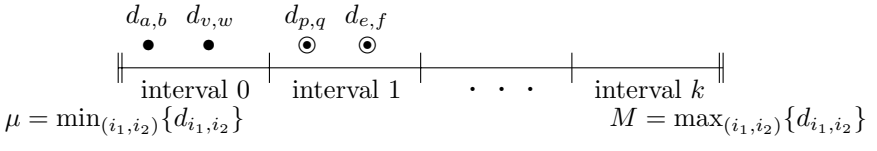


Fig. 5. Illustration of "interval dividing"

Thus, it is possible to select the smallest ordinal "proximity" and integrate corresponding item pairs. The problem may exist in the case when items can be integrated into different item pairs because the number of pairs will be very large. In this case a special *limitation algorithmic rule* can be used:

limitation of integrated pairs number at each algorithm stage, e.g., $\leq \lambda \leq n$.

Now let us consider **multicriteria approach**, *e.g.*, Pareto approach [52]. The following m criteria are considered for item pair proximity $\{ C_1, ..., C_j, ..., C_m \}$. As a result, we get a space of vector proximity for item pair (i_1, i_2):

$$\delta_{(i_1,i_2)} = \{\delta_{(i_1,i_2),1}, ..., \delta_{(i_1,i_2),j}, ..., \delta_{(i_1,i_2),m}\},$$

where $\delta_{(i_1,i_2),j} = |z_{i_1,j} - z_{i_2,j}|$ corresponds to criterion C_j.

Fig. 6 illustrates the method (the depicted points correspond to item pair proximity $\{\delta_{(i',i'')}\}$): (i) if item pair proximity corresponds to ideal point then $r_{(i_1,i_2)} = 0$, (ii) if item pair proximity corresponds to Pareto-effective points

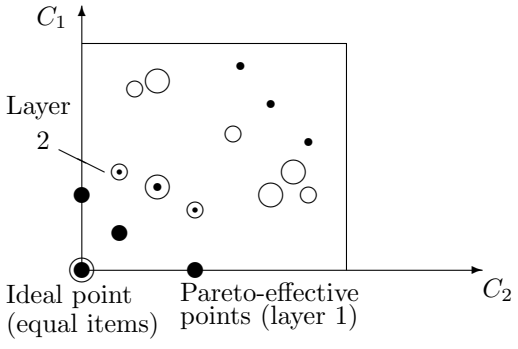


Fig. 6. Illustration for Pareto-approach

then $r_{(i_1,i_2)} = 1$, etc. Here the *limitation algorithmic rule* can be used as well. Clearly, other methods of multicriteria ranking can be used, *e.g.*, outranking techniques [56].

Multicriteria approaches (e.g., Pareto approach) to computing item pair proximity are more justified and correct than the usage of metric (*e.g.*, l_2) by the following reasons: *1.* from the viewpoint of using various scales (i.e., quantitative, nominal, ordinal) for assessment of the initial items, *2.* from the viewpoint of processing the obtained item estimates.

3.2 Multi-pair Integration Process

On the other hand, the clustering process can be organized as a series of clique problems [19]. Let $G = (A, E)$ be a graph over set of elements A and set of edges E corresponds to item pair with the minimal value of proximity. The solution of "maximal clique problem" in G leads to selection of a subgraph $H = (B, E')$ ($B \subset A$) where elements of B can be integrated into an aggregated item. Here a *Top-Down* process is obtained to build a "system hierarchy": revelation of maximal clique(s) in initial graph G, revelation of maximal clique(s) in subgraph(s) as H, etc. Evidently, the above-mentioned solving scheme is based on series of NP-hard problems. Note improvement ideas from the previous section (and corresponding polynomial operations) are more preferable from the viewpoint of complexity.

4 Improved Algorithms

Let us consider algorithm improvements.

Improvement 1 (algorithm 2):

Stage 1. Computing an ordinal "distance"/proximity (0 corresponds to equal or the more similar elements). Here it is possible to compute the pair "distance" via the previous approach and mapping the pair "distance" to the ordinal scale.

Stage 2. Revelation of the smallest pair distance and integration of the corresponding elements.

Note 1. It is possible to reveal several close element pairs and execution several pair integration.

Note 2. It is possible to include the same element into different integrated pairs.

Stage 3. The stage corresponds to stage 3 in *algorithm 1.*

Here, a hierarchical structure for the element pair integration (Bottom-Up) is obtained (several element pair integration at each integration step). The complexity of the problem may consist in revelation of many subcliques (in graph over elements and their proximity). In the process of computing the ordinal proximity it is reasonable to use a limited number of element pairs for each level of the proximity ordinal scale. As a result, the limited number of integrated element pairs (or complete subgraphs or cliques) will be revealed at each integration stage. This provides polynomial complexity of the algorithm (number of operations, volume of required memory) $O(m\ n^2)$.

Improvement 2 (algorithm 3): This algorithm is close to *algorithm 2*, but the computing process for the ordinal element pairs proximity is based on multicriteria analysis, *e.g.*, Pareto-approach or outranking technique (*i.e.*, Electre-like methods). Complexity of the algorithm is $O(m\ n^4)$.

The algorithms 2 and 3 implement the following trend:

from tree-like structure (of clustering process) to hierarchy.

An analysis of obtained clique(s) can be included into the algorithms as well.

5 Application

5.1 Education

The described algorithms are used as a special student laboratory work (MatLab-environment) in the author's course "Design of systems: structural approach", Moscow Inst. of Physics and Technology (State Univ.) ([41], [42]). Each student has to prepare the program and an example (a numerical example or a real world application from his/her professional domain).

Table 1. Estimates

	Attributes			
	T_1	T_2	T_3	T_4
A_1	2	3	300	2.4
A_2	54	5	100	5.0
A_3	11	3	300	2.4
A_4	54	3	300	2.4
A_5	55	1	100	2.4
A_6	0.25	1	10	2.4
A_7	54	3	200	5.0

Table 2. Distances/Proximity

	A_2	A_3	A_4	A_5	A_6	A_7
A_1	5.9(4)	1.4(1)	5.0(3)	5.9(4)	5.0(3)	5.9(4)
A_2		5.0(3)	3.0(2)	4.1(3)	6.5(4)	2.2(1)
A_3			4.0(3)	4.9(3)	3.8(2)	3.2(2)
A_4				3.0(2)	6.1(4)	1.4(1)
A_5					5.0(3)	2.4(1)
A_6						5.9(4)

5.2 Design of System Hierarchy

Let us consider clustering of wireless standards: 802.11 (A_1), 802.11a (A_2), 802.11b (A_3), 802.11g (A_4), 802.15.3 (A_5), 802.15.4 (A_6), and HyperLAN2 (A_7). This numerical example is partially based on laboratory work of student Dmitry Yu. Mikhin (Fall 2005). The set of attributes is as follows: Maximal speed (T_1), Power requirements (T_2), Distance (m) (T_3), and Frequency (GHz) (T_4). Table 1 contains estimates of the items, Table 2 contains initial "distances" and corresponded ordinal proximity (4 level). Fig. 7 depicts clustering hierarchy.

Algorithm 2 (**interval dividing scheme**) was used. Here after *Step 1* the item estimates and "distance"/proximity matrix were re-computed and 3-item clique (initial items $\{2, 4, 5, 7\}$) was revealed.

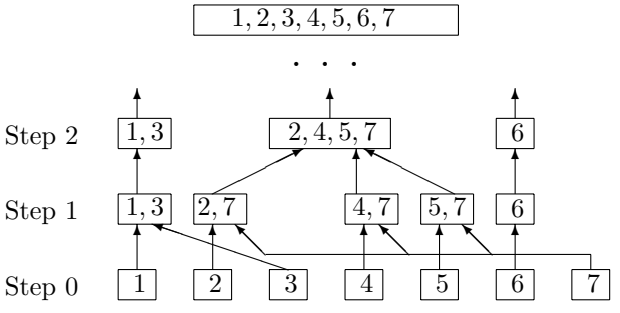


Fig. 7. Clustering hierarchy

6 Towards Quality of Clustering

Generally, clustering procedures can be evaluated from the following viewpoints: (a) complexity (*e.g.*, computational complexity, required memory, information complexity of various kinds) and (b) quality of the results. Let us consider basic goals of clustering processes as follows: (1) a set of clusters, (2) hierarchy of clusters (e.g., trees, hierarchies), (3) fuzzy clustering (e.g., intersection of clusters).

A traditional clustering evaluation measure (for a cluster set) has the follows kind: (*e.g.*, [21], [32]):

$$ClusteringQuality = \frac{InterClusterDistance}{IntraClusterDistance}$$

In the case of hierarchical clustering another metric has to be applied (*i.e.*, other goals) (*e.g.*,[32], [43]). In [32] three basic measures are analyzed and new measures are suggested: *1.* three basic measures: (i) *the greatest coverage by the smallest possible clusters*, (ii) *the greater cluster's inferential power*, and (iii) minimal overlap between clusters; *2.* new metrics: *Quality*, *Diversity*, and *Coverage*. Fuzzy clustering procedures require special kinds of quality metrics (*e.g.*, [28]).

In our case (as in many applications), clustering procedures have an investigation character and are targeted to an analysis of data under various research situations. A procedural possibility to change some parameters (*e.g.*, proximity intervals), modes, to analyze intermediated results, etc. is a result. From this viewpoint, complexity, easy to use and to understand, changeability, results presentability, reproducibility of results can be considered as basic goals and characteristics. Finally, quality of hierarchical clustering procedures and performance analysis of the resultant hierarchical clustering require special additional studies (including usage of multicriteria approaches).

7 Conclusion

In the paper, we have described our modified agglomerative algorithms for hierarchical clustering which involve the following additional operations: (i) building an ordinal item pair proximity ('distance') including the usage of multicriteria approaches; (ii) integration of several item pair at each stage of the algorithms; and (iii) inclusion of the same items into different integrated item pairs. Multicriteria approaches and ordinal item pair proximity are more reasonable for the following two viewpoints: (a) using various scales for item assessment and (b) processing the item estimates. The suggested modifications above are significant from the viewpoints of practice, *e.g.*, design of systems structure/architecture for engineering and computer systems.

Future research directions are the following: *1.* evaluation and comparative study of the suggested clustering algorithms *2.* design and analysis of a man-machine clustering procedure that can involve expert judgment; *3.* examination of various real world applications; *4.* usage of fuzzy set approaches; and *5.* special studies of hierarchical clustering procedures and performance analysis of the resultant hierarchies.

The author would like to thank the anonymous referees whose comments highly contributed to the revision of this paper.

References

1. Agrawal, P.K., Procopiuc, C.M.: Exact and approximation algorithms for clustering. Algorithmica 33, 201–226 (2002)
2. Anderberg, M.R.: Cluster Analysis for Applications. Academic Press, New York (1973)
3. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vector. In: 47th Annual IEEE Symp. on Foundations of Compter Science FOCS 2006, Berkeley, CA, pp. 475–486 (2006)
4. Arabie, P., Hubert, L.J., De Soete, G. (eds.): Clustering and Classification. World Scientific, Singapore (1996)
5. Augustson, J.G., Minker, J.: Analysis of some graph-theoretical cluster techniques. J. of the ACM 17(4), 575–588 (1970)
6. Babu, G.P., Murty, M.N.: Clustering with evolution styrategies. Pattern Recognition 27(2), 321–329 (1994)

7. Bandyopadhyay, S., Coyle, E.J.: An energy efficient hierarchical clustering algorithm for wireless sensor network. In: INFOCOM 2006 (2006)
8. Biswas, G., Weinberg, J., Li, C.: Conceptual Clustering Method for Knowledge Discovery in Databases. Editions Technip. (1995)
9. Bournaud, I., Ganascia, J.-G.: Conceptual clustering of complex objects: A generalization space based approach. In: Ellis, G., Rich, W., Levinson, R., Sowa, J.F. (eds.) ICCS 1995. LNCS, vol. 954, pp. 173–187. Springer, Heidelberg (1995)
10. Capoyleas, V., Rote, G., Woeginger, G.: Geometric clustering. J. of Algorithms 12, 341–356 (1991)
11. Castro, R.M., Coates, M.J., Nowak, R.D.: Likelihood based hierarchical clustering. IEEE Trans. on Signal Processing 52(8), 2308–2321 (2004)
12. Chatterjee, M., Das, S.K., Turgut, D.: WCA: a weghted clustering algorithm for mobile Ad Hoc networks. Cluster Computing 5, 193–204 (2002)
13. Chen, Y.: Multiple Criteria Decision Analysis: Classification Problems and Solutions. PhD Thesis, University of Waterloo, Ontarion (2006)
14. Cheng, Y., Fu, K.S.: Conceptual clustering in knowledge organization. IEEE Trans. PAMI 7, 592–598 (1985)
15. Cheng, C.H.: A branch and bound custering algorithm. IEEE Trans. SMC 25, 895–898 (1995)
16. Du, Q., Faber, V., Gunzburger, M.: Centroidal Voronoi tesselations: applications and algoithms. SIAM Rev. 41, 637–676 (1999)
17. Estvill-Castro, V., Murray, A.T.: Spatial clustering for data mining with genetic algorithms, Technical Report, Queensland Univ. of Technology, Australia (1997)
18. Garay, G., Chaudhuri, B.B.: A novel genetic algorithm for automatic clustering. Pattern Recognition 25(2), 173–187 (2004)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability. The Guide to the Theory of NP-Completeness. W.H.Freeman and Company, San Francisco (1979)
20. Genkin, A.V., Muchnik, I.B.: Fixed points approach to clustering. J. of Clustering 10, 219–240 (1993)
21. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theoret. Comput. Sci. 38, 293–306 (1985)
22. Gordon, A.D.: Classification, 2nd edn. Chapman&Hall/CRC (1999)
23. Guha, S., Rastogi, R., Shim, K.: ROCK: a robust clustering algorihtm for categorial atributes. Information Systems 25(5), 345–366 (2000)
24. Fasulo, D.: An Analysis of Recent Work on Clustering Algorithms. Technical Report # 01-03-02, Univ. of Washington (2001)
25. Fraley, C.: Algorithms for model-based Gaussian hierarchical clustering. SIAM J. on Scientific Computing 20(1), 270–281 (1998)
26. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: On clustering validation techniqes. J. of Intelligent Information Systems 17(2-3), 107–145 (2004)
27. Hartigan, J.A.: Clustering Algorithms. Wiley, New York (1975)
28. Hoppner, F., Klawonn, F., Kruse, R., Runkler, T.: Fuzzy Cluster Analysis. Wiley, New York (1999)
29. Hubert, L.J.: Some applications of graph theory to clustering. Psychomerica 39, 283–309 (1974)
30. Jain, A.K., Dubes, R.C.: Algorithms for Custering Data. Prentice Hall, Englewood Cliffs, NJ (1988)
31. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Computing Surveys 31(3), 264–323 (1999)
32. Jonyer, I., Cook, D.J., Holder, L.B.: Discovery and evaluation of graph-based hierarchical conceptual clustering. J. of Machine Learning Research 2, 19–43 (2001)

33. Jajuqa, K., Sokolovski, A., Bock, H.-H. (eds.): Classification, Clustering and Data Analysis. Recent Advances and Applications. Springer, Heidelberg (2002)
34. Jardine, N., Sibson, R.: Mathematical Taxonomy. Wiley, London (1971)
35. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika 2, 241–254 (1967)
36. Jones, D., Beltramo, M.A.: Solving partitioning problems with genetic algorithms. In: Proc. of 4th Int. Conf. on Genetic Algorithms, pp. 442–449 (1991)
37. Kanungo, T., Mount, D.M., Natanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans. PAMI 24(7), 881–892 (2002)
38. Karypis, G., Han, E., Kuma, V.: Chameleon: hierarchical clustering using dynamic modeling. IEEE Computer 32, 68–75 (1999)
39. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduciton to Cluster Analysis. John Wiley & Sons, New York (1990)
40. Kivijarvi, J., Franti, P., Nevalainen, O.: Self-adaptive genetic algorithm for clustering. J. of Heuristics 9(2), 113–129 (2003)
41. Levin, M.Sh.: Course 'Design of Systems: structural approach (2004...2007), http://www.iitp.ru/mslevin/SYSD.HTM
42. Levin, M.Sh.: Course 'System design: structural approach. In: 18th Int. Conf. Design Methodology and Theory DTM2006, DETC2006-99547 (2006)
43. Markov, Z.: A lattice-based approach to hierarchical clustering. In: Proc. of the Florida Artificial Intelligence Research Symposium, pp. 389–393 (2001)
44. Maulik, U., Bandyopadhyay, S.: Genetic algorithm-based clusterign technique. Pattern Recognition 33, 1445–1465 (2000)
45. Michalski, R.S., Stepp, R.: Revealing conceptual structure in data by inductive inference. In: Hayes, J.E., Michie, D., Pao, Y.-H. (eds.) Machine Intelligence 10, pp. 173–196. Wiley, New York (1982)
46. Michalski, R.S., Stepp, R.: Learning from observation. In: Michalski, R.S., carbonell, J.C., Mitchell, T.M. (eds.) Machine Learning: An Artificial Intelligence Approach, Palo Alto, CA, Tioga, pp. 163–190 (1983)
47. Mirkin, B.: Approximation clustering: a mine of semidefinite programming problems. In: Pardalos, P., WOlkowich, H. (eds.) Topics in Semidefinite and Interior Point Methods. Fields Institute Communication Series, AMS, Providence, pp. 167–180 (1997)
48. Mirkin, B., Muchnik, I.: Combinatorial optimization in clustering. In: Du, D.-Z., Pardalos, P. (eds.) Handbook on Combinatorial Optimization, vol. 2, pp. 261–329. Kluwer, Boston, MA (1998)
49. Mirkin, B.G.: Clustering for Data Mining: A Data Recovery Approach. Chapman & Hall / CRC (2005)
50. Murtagh, F.: A survey of recent advances in hierarchical clustering algorithms. The Computer Journal 26, 354–359 (1993)
51. Pal, N.R., Bezdek, J.C., Tsao, E.C.-J.: Generalized clsutering networks are Kohonen's self-organizing scheme. IEEE Trans. Neural Networks 4, 549–557 (1993)
52. Pareto, V.: Manual of Political Economy. Reprint ed., New York (1971)
53. Pavan, M., Pellilo, M.: Dominant sets and pairwise clustering. IEEE Trans. on PAMI 29(1), 167–172 (2007)
54. Ramanathan, K., Guan, S.U.: Clustering and combinatorial optimization in recursive supervised learning. J. of Combinatorial Optimization 13(2), 137–152 (2007)
55. Rasmussen, E.: Clustering algorithm. In: Information Retrieval. Printice Hall, Englewood Cliffs (1992)
56. Roy, B.: Multicriteria Methodology for Decision Aiding. Kluwer Academic Publishers, Dordrecht (1996)

57. Sethi, I., Jain, A.K. (eds.): Artificial Neural Networks and Pattern Recognition: Old and New Connections. Elsevier, New York (1991)
58. Shekar, B., Murty, N.M., Krishna, G.: A knowledge-based clustering scheme. Pattern Recognition Letters 5(4), 253–259 (1987)
59. Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy. Freeman, San Francisco (1973)
60. Stepp, R.E., Michalski, R.S.: Conceptual clustering of structured objects: a goal oriented approach. Artificial Intelligence 28(1), 43–69 (1986)
61. Van Ryzin, J. (ed.): Classification and Clustering. Academic Press, New York (1977)
62. Willet, P.: Recent trends in hierarchical document clustering: a critical review. Information Processing & Management 24(5), 577–597 (1988)
63. Zahn, C.T.: Graph-theoretic method for detecting and describing gestalt clusters. IEEE Trans. Comput. 20, 68–86 (1971)
64. Zait, M., Messatfa, H.: A comparative study of clustering methods. Future Generation Computer Systems 13, 149–159 (1997)
65. Zaponunidis, C., Doumpos, M.: Multicriteria classification and sorting methods: A literacy review. Eur. J. of Oper. Res. 138(2), 229–246 (2002)
66. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: a new data clustering algorithm and its applications. Data Mining and Knowledge Discovery 1(2), 141–182 (1997)

Estimation of the Click Volume by Large Scale Regression Analysis

Yury Lifshits[1,⋆] and Dirk Nowotka[2]

[1] Steklov Institute of Mathematics St.Petersburg, Russia
yura@logic.pdmi.ras.ru
[2] FMI, Universität Stuttgart, Germany
nowotka@fmi.uni-stuttgart.de

Abstract. How could one estimate the total number of clicks a new advertisement could potentially receive in the current market? This question, called the *click volume estimation problem* is investigated in this paper. This constitutes a new research direction for advertising engines. We propose a model of computing an estimation of the click volume. A key component of our solution is the application of linear regression to a large (but sparse) data set. We propose an iterative method in order to achieve a fast approximation of the solution. We prove that our algorithm always converges to optimal parameters of linear regression. To the best of our knowledge, it is the first time when linear regression is considered in such a large scale context.

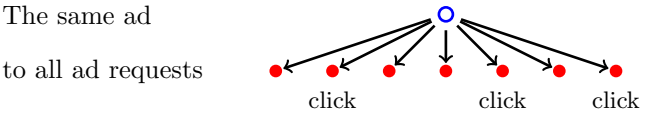
1 Introduction

In general, an *advertising engine* (AE) (1) maintains a database of advertisements, (2) receives ad requests "some person is accessing some media", and (3) returns several ads that are most relevant to this request. Google AdWords, Yahoo! Search Marketing, and Microsoft adCenter are the most prominent advertising engines for sponsored search. Google AdSense is an example of an AE for contextual advertisements. Finally, the Amazon.com recommendation system [8] is a particular case for an e-commerce recommendations AE. We expect that specialized advertising engines will be introduced very soon for blogspace, social networks, computer games and virtual reality, and even supermarket bills.

In this paper we start a new research direction for advertising engines. Consider the following question: How could one estimate the total number of clicks a new advertisement can potentially receive in the current market? We call this the **click volume estimation problem** and use $CV(a)$ to denote the click volume of an advertisement a. Knowledge about advertisements (the *ad space*), requests for ads (the *request space*), and *historical information* can be used to calculate an estimation of $CV(a)$. The click volume estimation problem has not yet been investigated in the literature to the best of our knowledge.

⋆ The first author acknowledges support from the grants NSh-8464.2006.1, INTAS N 04-77-7173, and INTAS YSF 1000014-6233.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 216–226, 2007.



There are plenty of reasons to be interested in the click volume of an ad. Let's consider some of them.

- Maintainers of advertising engines might wish to understand how many clicks they can (approximately) sell for any given advertisement. Click volume information can be useful for setting optimal prices.
- The click volume can measure the current effectiveness of advertising engines.
- Advertising engines might use different strategies for the cases when the click volume is smaller than the demand from advertisers and when it is larger. In the latter case an AE can *skip* some ad request even if advertisers mark it as belonging to their target group.
- The real goal of an AE is to recognize the whole interested audience for the given ad and to display it *only* to these people. Hence, estimating the volume of that audience is the first step towards the recognition problem.
- Using click volume estimation, an advertiser can predict the necessary resources needed to cover a given fraction of market.
- Comparison of overall click volume and click volume restricted to the target subspace of ad requests provides a kind of "target coverage" value. It can help advertisers to understand whether their target description is good enough.
- Assuming we have a purchase history table. Then applying similar techniques we can estimate the purchase market for a new product.

Results. This paper is the first step towards setting a formal definition of click volume and constructing efficient algorithms for computing reliable CV estimations. Our main contributions are (1) a general model of an advertising engine and its history table, (2) a methodology for calculating the click volume using linear regression, (3) a fast iterative algorithm for solving the linear regression problem on large and sparse data sets based on [11,18], (4) a complexity bound for one round of iterations, and (5) a proof of convergence for the algorithm. Finally, we pose a series of open problems and suggest directions for further research in Section 4.

Let us describe our solution for the click volume problem in an informal way. We take a history table and transform it into the list of pairs: event vector, empirical value of click-through rate. Here, an event vector characterizes a displayed ad, an ad request and their relationship. It belongs to a high dimensional euclidean space, but only few of its components are nonzero. We make an assumption that the click-through rate (more precisely, *logit* of click-through rate) can be calculated as a scalar product between the event vector and some unknown vector α. Then we have to find the α that minimizes the prediction error

over the whole history table. This is a well-known linear regression problem. Unfortunately, classical methods, like direct method and SVD-based method [15] are infeasible in our settings. As far as we know, all previous algorithms have time complexity $\Omega(mn)$ where mn is the size of underlying matrix. In order to get better complexity, we have to use the sparseness of the underlying matrix.

In essence, we need just to compute the projection of one vector to the linear hull of some family of *sparse* vectors. While solving systems of linear equations and computing eigenvectors are well studied in sparse settings [13], no particular method was suggested for the *projection* problem. In this paper we propose an iterative algorithm based on componentwise descent method [18]. In [18] only one component of the current approximation is updated in every turn. Inspired by the sequential minimal optimizations method [11] used for training support vector machines, we make a modification to componentwise descent method. That is, we calculate the optimal shift value *analytically*. We show that shift computation is linear just to the number of *nonzero* components of the corresponding term vector.

In this paper we just start the investigation of solving the large scale regression problem on sparse data. But we are convinced that this method can find much more applications in web computing than just click volume estimation.

Related research. The algorithm for predicting the click-through rate in [12] was the main inspiration source for our research. However, there are some important differences. In [12] the click-through rate (hence, click volume) is estimated for all ads shown for a given search term (ad request in our terminology). We present a solution for the dual problem: fixing an ad and assuming that it is displayed on all ad requests. Moreover, we estimate the click through rate for any pair of ad and ad request. In our case the problem of insufficient history can not be solved *solely* by the assumption that two content-similar ads have a similar click volume. Indeed, a collection "ad-to-ad-request" is far from the completeness for *every* ad and even for every ad cluster. Next, we present a model which captures various industrial solutions at the same time, while [12] addresses only sponsored search technology.

Let us compare our regression based solution with other possible approaches. For every newcomer event vector one can locate the nearest events from our history table and use their click-through rates as the basis of estimation. However, the nearest neighbors algorithms either are (asymtotically) too slow [6], or use assumptions (like a non-euclidean space) [17] that clearly do not hold for our case. Alternatively, one can, for example, treat ads as people, ad requests as books and empirical click-through rates as ratings. Than we can apply the well developed theory of collaborative filtering [10]. Unfortunately, one needs at least few appearances of the newcomer ad in the history table to apply this method. Also, collaborative filtering does not use term similarity between ads and ad requests.

We refer to the papers [4,16] for a general introduction to sponsored search. Algorithms for optimal ad choosing are constructed in [9]. Auction design for sponsored search is well presented in [3,5]. Many open problems around advertising engines were posed on the SSA'06 panel discussion [1]. Finally, various versions of similarity for search terms (i.e. ad requests in our terminology) are investigated in the paper [2].

2 Basic Model: Advertising Engine and History Table

The general setting discussed in this paper is an advertising engine which, given some historical information about placing advertisements and a new advertisement from an advertiser, has to estimate the number of clicks (click volume) this new ad would get over a certain period of time. Let's define our notation.

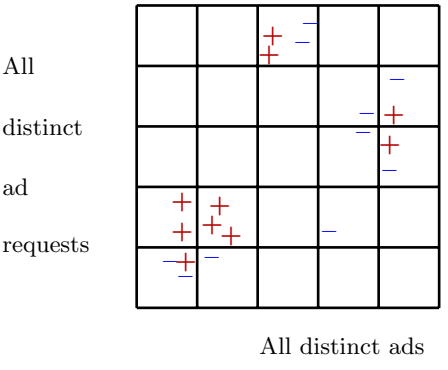
Let A be the set of all *advertisements*. By convention we denote elements of A by a and its subscripted versions. Let $a_{new} \in A$ denote the advertisement whose click volume will be estimated. In principle an element of A might be defined by several properties like the content of the advertisement (for example text with a link and a phone number), a target audience description, some keywords describing the ad, and so on. However, all we assume here is that an ad is represented by a vector of reals. For example, an entry of a might denote the membership of the ad in some property where 1 means that a has that property and 0 means that a does not have that property.

Furthermore, we need a notion of ad request. The AE is contacted every time a person is contacting some media (in most cases websites). Let R be the set of all *ad requests* whose elements are denoted by r and its subscripted versions in the following. Requests, like advertisements, could be described by a number of parameters like a person (or rather IP address), media (for example a search web page), and an action (for example an entered search query). But again, all we require is r being a vector of reals, like advertisements.

Advertisements and requests together form an exposition *event*. An event should be represented by features of the advertisement (for example, keywords, language, target group), features of the ad request (for example, location, search phrase), and the relation between ad and ad request (for example, whether or not the language of the ad is spoken at the location the request came from). Let us assume that an event is represented as a vector such that each component denotes whether or not that event possesses that feature. Let E denote the set of all events. Given an ad a and a request r, let $e(a, r)$ denote the event representing the ad, the ad request, and the relation between them.

The data we use for our estimation of the click volume is called *history table* over $E \times \{0, 1\}$ where $(e(a, r), b)$ denotes an advertisement a shown on ad request r which received a click if $b = 1$ and no click if $b = 0$. Let us fix an arbitrary history table HT with n entries for the rest of this paper. We have

$$HT = (e(a_1, r_1), b_1), (e(a_2, r_2), b_2), \ldots, (e(a_n, r_n), b_n) \ .$$



We can visualize the history table as a set of + and − symbols in the table whose columns correspond to ads and rows correspond to ad request. Actually every cell represents the set of all possible unique exposition events that are observed at the same pair (a, r). Our history table is a collection of "click-occurrence" values on some points in the table. We want to stress here that there is an important difference between columns and rows. Namely, in our history ad requests are taken randomly from standard daily (weekly, annual) distribution. But ads were chosen by the advertising engine and they are by no means random. Typically, for every row our history covers only few cells. Therefore, we can assume that only a linear (more precisely, proportional to the number of all requests) number of cells is covered by the history.

The *click volume* $CV(a)$ is the total number of clicks over a fixed period of time (for example one week) that we would expect for a to get if shown at *all* ad requests taken from the same distribution as in HT. The click-through rate is a standard term for internet advertising. We recall it in the setting of our model. Given an event e the *click-through rate* (CTR) $CTR(e)$ is the ratio between the number of clicks for e and the total number of times e occurs (impressions), or in other words, the probability that a is clicked on in conjunction with the ad request r when $e = e(a, r)$.

3 Reduction to the Regression Problem

The history table HT records for each event e either a click or no click. What we really want to know, though, is the probability of an event being clicked, that is, we need the actual CTR of e. The CTR of an event can in principle be calculated from the number of times e occurs in HT and the number of times when e received a click. However, this simple approach does not work well, since most of the vectors in HT are unique.

The history table HT is assumed to be our given input. Although it is a natural way to present the input data, it is not suitable for calculating the click volume of a new ad quickly. Therefore, HT is pre-processed. The first step in the pre-processing phase is a *dimensionality reduction* of HT. There exist a number of different approaches to reduce the dimensionality of E; see [14] for a survey. We suggest dimensionality reduction by some term extraction method, for example *term clustering*. Latent semantic indexing is another method for term extraction. However, we do not suggest latent semantic indexing here since it will turn out that sparseness of the event vector set is a desired property for applying the linear

regression method to our problem as suggested below; see the next section in particular. Let us assume that DR denotes the dimensionality reduction function of our choice, that is, $DR(e)$ denotes the event derived from e by reducing its dimensionality.

The function DR is now used to transform the history table HT into a *reduced history table* RHT. Firstly, we replace every entry (e, b) of the history table HT by $(DR(e), b)$, and secondly, all entries (e', b_1), ..., (e', b_k) of the same reduced event e' are combined to one entry (e', b') where b' denotes the click through rate $(\sum_{1 \leq i \leq k} b_i)/k$ for e'. Let the set of all reduced events in RHT be denoted by a matrix T and the set of all click through rates by a vector β, that is $RHT = T\beta$.

Given an event e_{new}, the problem of estimating the click through rate of e_{new} can be formulated as the problem of fitting the function that relates reduced events to their click through rate. The *linear regression* analysis is a standard method for curve fitting where a linear function is used to describe the relation between two variables; in our case, these are reduced events and click through rate.

However, it is not guarantied that the function resulting from the linear regression analysis yields a value between 0 and 1 for e_{new}. To avoid this problem, we apply a one-to-one mapping from $[0, 1]$ to $[-\infty, +\infty]$ to all click through rate values in RHT. Such a mapping is for example logit where $\text{logit}(p) = \log(p/(1-p))$. Let now $\gamma = \text{logit}(\beta)$, and we perform a linear regression analysis on $T\gamma$. The regression analysis yields a vector α such that $\|T\alpha - \gamma\|$ is minimal, that is, α approximates the relation between reduced events and the (logit of the) click through rate such that the sum of the quadratic error of all entries is minimal. The click through rate of e_{new} is now estimated by $\text{logit}^{-1}(\alpha \cdot DR(e_{new}))$, that is (1) the dimensionality of e_{new} is reduced, (2) this reduced event is combined with α, and (3) the resulting value is mapped into the interval $[0, 1]$ by logit^{-1}.

We would like to know the sum of all clicks a new ad a_{new} should get when exposed to all requests of the history (for a given time period). That is, a query consists of an ad a_{new} which needs to be combined with all requests in HT. We get the following formula.

$$CV(a_{new}) = \sum_{1 \leq i \leq n} \text{logit}^{-1}(\alpha \cdot DR(e(a_{new}, r_i)))$$

All these steps are summarized in the following methodology.

Methodology for Click Volume Estimation

Pre-processing:

1. Dimensionality reduction of E, e.g. by term clustering
2. Calculation of $RHT = T\gamma$
3. Approximate calculation of α such that $\|(\alpha T) - \gamma\|$ is minimal (see next section)

Query: Calculate $CV(a_{new}) = \sum_{1 \leq i \leq n} \text{logit}^{-1}(\alpha \cdot DR(e(a_{new}, r_i)))$

4 Solving the Large Scale Regression Problem

The methodology of estimating the click volume of a_{new} of the previous section employs the linear regression analysis for calculating $CV(a_{new})$. Consider the history table entries of reduced dimensionality, let $e = DR(e(a_{new}, r_i))$. The direct calculation of parameters of linear regression requires inverting a huge and non-sparse matrix. Since we assume to work with large data sets and under strict time constraints this method is not available to us. Instead, we suggest an alternative method to estimate α which is iteratively approximating the optimal solution and which also has good convergence properties for sparse settings as the ones we consider for click volume. Intuitively, α is learned by using the reduced history table as a training set.

The formal setting. We consider the history table of reduced dimensionality RHT where we have a logit value for click through rate assigned to every event entry e. Let m denote the dimension of e, and let T denote the $n \times m$ matrix of entries in RHT, and let γ be an n vector where γ_i denotes the logit value of the click through rate the i-th entry in RHT. Let t_j denote the j-th column vector (of dimension n) of T for all $1 \leq j \leq m$. The goal is to find the α such that $\|T\alpha - \gamma\|$ is minimal. Certainly, $\alpha = (T \cdot T^*)^{-1}T^*\gamma$. However, calculating α directly this way is too expensive for our purpose. Therefore we describe an iterative method in the next paragraph which generates a sequence $\alpha^{(1)}$, $\alpha^{(2)}$, ... converging towards α.

The iterative algorithm. Let $\alpha^{(1)}$ be the zero vector. Assume $\alpha^{(k)}$ is known and we would like to calculate $\alpha^{(k+1)}$. First, one component j of $\alpha^{(k)}$ is chosen. Then $\alpha^{(k+1)}$ is such that $\alpha_i^{k+1} = \alpha_i^{(k)}$ for all $i \neq j$ and

$$\alpha_j^{(k+1)} = \alpha_j^{(k)} + \frac{(\alpha^{(k)}T - \gamma) \cdot t_j}{\|t_j\|^2} \ .$$

Intuitively, we minimize the error of the "prediction" of all CTR values in our history table by $\alpha^{(k)}$ by adjusting only the j-th component $\alpha^{(k)}$ while all other components are left unchanged.

A geometric interpretation. Let us reformulate our algorithm in terms of *discrepancy* vector φ:

$$\varphi^{(k)} = \gamma - \alpha^{(k)}T$$

$$\varphi^{(0)} = \gamma, \quad \varphi^{(k+1)} = \varphi^{(k)} - \frac{\varphi^k \cdot t_j}{\|t_j\|^2} t_j$$

Consider the linear hull $span(T)$ of T. We would like to come as close to γ as possible, while all our estimations belong to $span(T)$. Hence, our optimal estimation is the orthogonal projection of γ to $span(T)$, let us denote it by φ. A minimal error achieved by a vector that is equal to $span(T)$-orthogonal

component of γ. In every step of our algorithm we do the following. Take our current estimation $\varphi^{(k)} \in span(T)$, draw a line parallel to t_j and going through $\varphi^{(k)}$, then project γ (or, with the same result, φ) to this line. This projection point is our new estimation $\varphi^{(k+1)}$.

Algorithm analysis

Theorem 1 (Convergence theorem). *Consider a euclidean space, a vector $\varphi^{(0)}$, a family of vectors $T = \{t_1, \ldots, t_m\}$, and a fixed infinite order of updates $J : j_1, j_2, \ldots$ that contains every index infinitely many times. Let us construct the sequence $\varphi^{(k)}$ by the following rule:*

$$\varphi^{(k+1)} = \varphi^{(k)} - \frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k}$$

Then the sequence $(\varphi^{(k)})$ converges to φ, where φ is the $span(T)$-orthogonal component of $\varphi^{(0)}$.

Proof. Note that $\varphi^{(k+1)} \perp t_{j_k}$. Indeed,

$$\varphi^{(k+1)} \cdot t_{j_k} = \varphi^{(k)} \cdot t_{j_k} - \frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k} \cdot t_{j_k} = 0.$$

Since $\varphi^{(k)} = \varphi^{(k+1)} + \frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k}$ and $\varphi^{(k+1)} \perp t_{j_k}$, we have $\|\varphi^{(k)}\| \geq \|\varphi^{(k+1)}\|$.

Assume now that $(\varphi^{(k)})$ does not converge to φ. Since $\|\varphi^{(k)}\|$ is bounded, we can choose a subsequence $(\varphi^{(k_i)})$ converging to some $\psi \neq \varphi$. Let us divide all T family to those vectors that are orthogonal to ψ (subfamily T_1) and those that are not orthogonal (subfamily T_2). Then T_2 is nonempty, otherwise ψ coincides with φ.

Let $c = \min_{j \in T_2} \frac{\psi \cdot t_j}{\|t_j\|}$. Since there exists a subsequence of $(\varphi^{(k)})$ converging to ψ, there are infinitely many members, that belong to the $c/2$ neighborhood of ψ. Consider one such visit $\varphi^{(k)}$. Let us look at the next updates. If we use $j_k \in T_1$ we can come only closer to ψ. Indeed, as was shown above, $\varphi^{(k+1)} \perp t_{j_k}$, moreover $\psi \perp t_{j_k}$. Since

$$\varphi^{(k)} - \psi = (\varphi^{(k+1)} - \psi) + \frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k}$$

and the first and second terms are orthogonal, $\|\varphi^{(k)} - \psi\| \geq \|\varphi^{(k+1)} - \psi\|$.

Since every index occurs infinitely often we will finally apply an update for some $j_k \in T_2$, still being in the $c/2$ neighborhood of ψ. Let us estimate the "size" of this step.

$$\frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k} = \frac{\psi \cdot t_{j_k} + (\varphi^{(k)} - \psi) \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k} \geq \frac{c\|t_{j_k}\| - \frac{c}{2}\|t_{j_k}\|}{\|t_{j_k}\|^2} t_{j_k} \geq \frac{c}{2\|t_{j_k}\|} t_{j_k}.$$

Since the shift was orthogonal to $\varphi^{(k+1)}$, we have

$$\|\varphi^{(k+1)}\|^2 \leq \|\varphi^{(k)}\|^2 - \frac{c^2}{4}.$$

Let us summarize our observations. Sequence $(\varphi^{(k)})$ visits $c/2$ neighborhood of ψ infinitely often, once coming inside it cannot escape by T_1 updates, while T_2 update leads to a substantial decrease of $\|\varphi^{(k)}\|$. We get a contradiction with the facts that $\|\varphi^{(k)}\|$ is nonnegative and monotonically decreasing. □

Lemma 1 (Round complexity). *It is possible to make a single update for every j (one round of updates) in time linearly depending from the number of nonzero elements in the history table.*

Proof. At the beginning, we precompute norms of all t_j. We will maintain both current regression vector $\alpha^{(k)}$ and current discrepancy vector $\varphi^{(k)} = \gamma - \alpha^{(k)}T$. We start from $\alpha^{(k)} = 0$ and $\varphi^{(0)} = \gamma$. Consider some j and let q_j be the number of nonzero components in t_j. At first we update $\alpha_j^{(k)}$ in $\mathcal{O}(q_j)$ time. Indeed, we need to calculate the scalar product between t_j and $\varphi^{(k)}$ and this can be done by few corresponding look-ups. Then we update the discrepancy vector by the rule $\varphi^{(k+1)} = \varphi^{(k)} - \frac{\varphi^{(k)} \cdot t_{j_k}}{\|t_{j_k}\|^2} t_{j_k}$. Again, scalar product can be computed in $\mathcal{O}(q_j)$ time and only q_j components of $\varphi^{(k)}$ should be modified. Summing over all j we get the required round complexity. □

Some other complexity remarks:

1. A vector $\alpha^{(k)}$ can be safely updated in two components j_1 and j_2 *in parallel* if we have $t_{j_1} \cap t_{j_2} = \emptyset$. Again the sparseness assumption would allow for a high degree of parallelism in practice.
2. Note that in the case of orthogonal column vectors t_i α is reached by updating every component exactly once.
3. Also the joint update of two components i and j is not expensive. In order to calculate

$$\left(\alpha_i^{(k+1)}\alpha_j^{(k+1)}\right)^* = \left(\alpha_i^{(k)}\alpha_j^{(k)}\right)^* - \left((t_i t_j)\cdot(t_i t_j)^*\right)^{-1}(t_i t_j)^*(\alpha^{(k)}T - \gamma)$$

we need to invert $(t_i t_j)\cdot(t_i t_j)^*$ which is only a 2×2 matrix.

5 Further Work

In this paper we state the click volume problem, show how one can reformulate it as the large scale regression problem and propose an iterative algorithm for solving the latter. It is the first time this has been done and many questions follow from that. How do we estimate the convergence speed of our algorithm? How should we choose the next component for update? Assume that we allow the output "cannot predict CTR for this event". Can we improve the accuracy by solving the problem with this relaxation? Can we combine the regression approach with other methods, e.g. clustering?

Experimental validation. Of course, one wants to verify our regression based approach in industrial applications (e.g. Google AdWords system). Forthcoming experiments should answer the following questions. What is the convergence speed of our iterative algorithm in practice? What is the overall error of linear regression estimation on the history table? What event vector representation and dimensionality reduction routines lead to the most accurate CTR prediction? Finally, it is interesting to apply the large scale linear regression algorithm to other problems, e.g. predicting news article popularity on `digg.com`.

Related problems for on-line advertisements. The dual problem for click volume estimation is the *ad volume estimation*. Namely, to estimate the total amount of advertisements that could get a positive response on the given ad request. *Click volume for the whole market*: What is the fastest way to estimate the click volume for every advertisement in the system? More precisely, can we do it faster than doing a separate click volume estimation for every ad?

Finally, the click volume problem is just a single member of our list of web-related algorithmic problems. In [7], one can find more theoretical questions in web computing.

Acknowledgments. The authors would like to thank Javier Esparza, Stefan Göller, Benjamin Hoffmann, Stefan Kiefer, Mikhail Lifshits, Markus Lohrey, Hinrich Schütze, Kirill Shmakov, Jason Utt and anonymous referees of the previous version of this paper for their useful comments and fruitful discussions.

References

1. Models for sponsored search: What are the right questions? Panel Discussion at SSA'06 (2006), `http://research.microsoft.com/~hartline/papers/panel-SSA-06.pdf`
2. Bartz, K., Murthi, V., Sebastian, S.: Logistic regression and collaborative filtering for sponsored search term recommendation. In: SSA'06 (2006)
3. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. In: SSA'06 (2006)
4. Fain, D., Pedersen, J.: Sponsored search: a brief history. In: SSA'06 (2006)
5. Feng, J., Bhargava, H., Pennock, D.: Implementing sponsored search in web search engines: Computational evaluation of alternative mechanisms. Informs Journal on Computing (2006)
6. Kleinberg, J.M.: Two algorithms for nearest-neighbor search in high dimensions. In: STOC '97, pp. 599–608 (1997)
7. Lifshits, Y.: A Guide to Web Research. Materials of mini-course at Stuttgart University (2007), Available at `http://logic.pdmi.ras.ru/~yura/webguide.html`
8. Linden, G., Smith, B., York, J.: Amazon.com recommendations item-to-item collaborative filtering. Internet Computing (2003)
9. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: AdWords and generalized on-line matching. In: FOCS'05. IEEE, New York (2005)
10. O'Connor, M., Herlocker, J.: Clustering items for collaborative filtering. In: SIGIR '01, Workshop on Recommender Systems (2001)

11. Platt, J.C.: Fast training of support vector machines using sequential minimal optimization. In: Advances in kernel methods: support vector learning, pp. 185–208. MIT Press, Cambridge, MA, USA (1999)
12. Regelson, M., Fain, D.: Predicting clickthrough rate using keyword clusters. In: SSA'06 (2006)
13. Saad, Y.: Iterative methods for sparse linear systems, 2nd edn. SIAM (2003)
14. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys 34(1), 1–47 (2002)
15. Seber, G.A.F., Lee, A.J.: Linear Regression Analysis. Wiley, Chichester (2003)
16. Tuzhilin, A.: The Lane's Gifts v. Google report (2006)
17. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: SODA '93, pp. 311–321 (1993)
18. Zhdanov, V.: The componentwise descent method. Mathematical Notes 22(1), 566–569 (1977)

Maximal Intersection Queries in Randomized Graph Models

Benjamin Hoffmann[1], Yury Lifshits[2,⋆], and Dirk Nowotka[1]

[1] FMI, Universität Stuttgart, Germany
{hoffmann,nowotka}@fmi.uni-stuttgart.de
[2] Steklov Institute of Mathematics at St.Petersburg, Russia
yura@logic.pdmi.ras.ru

Abstract. Consider a family of sets and a single set, called query set. How can one quickly find a member of the family which has a maximal intersection with the query set? Strict time constraints on the query and on a possible preprocessing of the set family make this problem challenging. Such maximal intersection queries arise in a wide range of applications, including web search, recommendation systems, and distributing on-line advertisements. In general, maximal intersection queries are computationally expensive. Therefore, one needs to add some assumptions about the input in order to get an efficient solution. We investigate two well-motivated distributions over all families of sets and propose an algorithm for each of them. We show that with very high probability an almost optimal solution is found in time logarithmic in the size of the family. In particular, we point out a threshold phenomenon on the probabilities of intersecting sets in each of our two input models which leads to the efficient algorithms mentioned above.

1 Introduction

The *nearest neighbor* problem is the task to determine in a general metric space a point that is closest to a given query point. This kind of queries appear in a huge number of applied problems: text classification, handwriting recognition, recommendation systems, distributing on-line advertisements, near-duplicate detection, and code plagiarism detection.

In this paper we consider the nearest neighbor problem in a "binary" form. Namely, every object is described as a set of its features and similarity is defined as the number of common features. For some cases, like recommending a person who has a maximal number of joint friends with you but is not your direct friend, this formalization is quite natural. On the other hand, weighted models could be simply reduced to the binary form. Let us illustrate this reduction by example. Assume that we are working with documents and their terms, and one particular term is "Ekaterinburg". Then we can introduce 8 new artificial terms Ekaterinburg1, ... Ekaterinburg8. For an object having the largest weight for

⋆ Partially supported by grants INTAS YSF 1000014-6233, INTAS 04-77-7173 and NSh-8464.2006.1.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 227–236, 2007.

Ekaterinburg in weighted representation we simply put all eight new terms, while for objects with small weights we put only Ekaterinburg1. Thus, an intersection similarity for the new representation is somehow reflecting the classical scalar product similarity for the vector model.

In order to construct an efficient solution some assumptions should be added to the problem. In our paper we assume that the input is taken from some predefined distribution. Then we construct an algorithm and show that the time complexity and/or the accuracy are reasonably good *with high probability*. Here we use the probability over the input distribution, not over random choices of the algorithm. This probabilistic approach was inspired by the recent survey of Newman [7]. He gives a comprehensive survey about random models of graphs that agree well with many real life networks, including Web graph, friendship graphs, co-authorship graphs, and many others. Hence, we can attack the nearest neighbor problem in already "verified" random models.

The Maximal Intersection Problem. Consider a family of sets and a single set. We ask for a member of the set family which has a maximal intersection with the query set.

The Maximal Intersection Problem (MaxInt)

Database: A family $\mathcal{F}$ of n sets such that $|f| \leq k$ for all $f \in \mathcal{F}$.
Query: Given a set f_{new} with $|f_{new}| \leq k$, return $f_i \in \mathcal{F}$ with maximal $|f_{new} \cap f_i|$.
Constraints: Preprocessing time $n \cdot polylog(n) \cdot poly(k)$ or $n^{1+o(1)}$. Query time $polylog(n) \cdot poly(k)$ or at most $o(n)$.

Let us restate the MaxInt problem in a graph theoretical notation. A database is a bipartite graph with vertex set partition (V, V') such that $|V| = n$ and the degree of every $v \in V$ is at most k. A query is a (new) vertex v (together with edges connecting v with V') of degree at most k. The query task is to return a vertex $u \in V$ with a maximal number of paths of length 2 from v to u.

Results. Inspired by [4] we present for the first time a theoretical investigation of MaxInt. In Section 2 and Section 3 we propose two new randomized models of bipartite graphs, called the Zipf model and the hierarchical schema. Assume that the terms of a query document are ordered by their frequency in the document collection. Now consider the probability curves for the two following events with parameter q. *Any q-match*: there is a document in the random (according to our models) collection that has at least q common terms with the query document. *Prefix q-match*: there is a document in the random collection that has at least q "top" terms of the query document. Both curves have the similar structure: the probability is close to 1 for small q, but suddenly, at some "magic level", it falls to nearly zero and remains so till the end. Our main observation is that these magic levels for prefix match and any match are very close to each other. And this is extremely important for solving MaxInt. Indeed, finding the best prefix

match is computationally feasible, but we don't know the general solution for MaxInt. We show that closeness of magic levels for any match and prefix match with high probability allows to find an approximate solution for MaxInt.

Related Work. MaxInt is a special case of the nearest neighbor problem. Indeed, one just needs to define the distance between two vertices in a bipartite graph as the inverse of the number of 2-step paths between these vertices. Yianilos proposed in [9] *vp*-trees (vantage point trees), a data structure that solves nearest neighbor problem on general metric spaces. The preprocessing time of his algorithms is in $\mathcal{O}(n \cdot \log n)$ and expected query time is in $\mathcal{O}(\log n)$. However, he uses a strong anti-discrete assumption about the metric that does not hold for our similarity-by-intersection-size model.

Nearest neighbors are particularly well studied in vector models with a similarity function based on the scalar product [3]. Actually, we can interpret a document as a vector of 0s and 1s (1 means a term is contained in a document). Then, the scalar product is equal to the size of the intersection. Unfortunately, the first algorithm from [3] needs quadratic space and the second one has linear query time, so none of them preserve our constraints. Both algorithms return a γ-approximate nearest neighbor and are based on random projections of the database vectors onto several randomly chosen directions.

Closely related to MaxInt is *text search*. Finding documents that most fit to some given search terms can also be considered as a problem on a bipartite graph. The documents and terms are the nodes and edges are drawn when a term occurs in a document. Basically the task is to find all documents containing *every* query term and rank these documents by relevance. The key technique in this area is inverted files (inverted indexing). A comprehensive survey of the topic can be found in [10].

A problem similar to MaxInt called *collaborative filtering* is considered by O'Connor and Herlocker in [8]. Collaborative filtering can be seen as a bipartite graph problem where nodes represent people and objects and weighted edges between these people and objects are defined by ratings. The task is to estimate the weight of a new edge.

2 MaxInt in the Zipf Model

Throughout the following sections we use a documents-terms notation, that is, vertices from $\mathcal{D} = \{d_1, \ldots, d_n\}$ represent documents and vertices from $\mathcal{T} = \{t_1, \ldots, t_m\}$ represent terms. Let $n \geq 3$, and $m \leq poly(n)$. By log we always mean $\log_2$, while ln denotes $\log_e$.

We now describe a probabilistic mechanism for generating a document collection called the *Zipf model*. Every document is generated independently. Terms occurrences are also independent. A document contains term t_i with probability $\frac{1}{i}$. Hence, the expected number of terms in a document is approximately equal to $\ln m$ in our model. This model is similar to the *configuration model* (see [7]) with Zipf's law for distribution of term degrees and constant document degrees.

Zipf's law states that in natural language texts the frequency of a word is approximately inversely proportional to its rank in the frequency table[1]. For more details about Zipf's law see [6].

Remark 1. The frequency of a term t in a collection $\mathcal{D}$ of documents is defined as

$$\frac{|\{d \in \mathcal{D} \mid t \in d\}|}{|\mathcal{D}|}.$$

The expected frequency of the term t_i is equal to $\frac{1}{i}$. At the same time, the expected frequency rank for t_i is exactly the i-th value among those of all terms. So the Zipf model reflects in a natural way Zipf's law. Since some of our motivating applications also deal with natural language texts, we can state that the Zipf model agrees with real life at least by degree distribution.

Remark 2. In the following proofs we will use two inequalities ($a, b > 0$):

$$\left(1 - \frac{a}{b}\right)^b < 2^{-a}, \qquad a \le b \qquad (*)$$

$$\left(1 - \frac{1}{ab}\right)^a > 1 - \frac{1}{b}, \qquad a, b \ge 2 \qquad (**).$$

These inequalities follow from the well-known fact $\lim_{n\to\infty}(1 - \frac{x}{n})^n = e^{-x}$.

For further considerations we partition the set of terms in the following way:

$$\underbrace{t_1\ t_2}_{P_1} \quad \underbrace{t_3\ t_4\ t_5\ t_6\ t_7}_{P_2} \quad \dots$$

Here the group P_i includes terms from $t_{\lceil e^{i-1} \rceil}$ to $t_{\lfloor e^i \rfloor}$ A document that contains $\ln m$ terms $p_1 \dots p_{\ln m}$, $p_i \in P_i$, will be called *regular*.

We now introduce a magic level to give statements about the most probable size of maximal intersection

$$q = \sqrt{2 \ln n}.$$

Theorem 1 (Magic Level for the Zipf Model). *Let $3 \le \gamma < q - 1$ be a positive integer. Fix n, m and a regular query document d_{new}. Then for a document collection following the Zipf model the following holds:*

1. *The probability that there exists a document $d \in \mathcal{D}$ that contains the first $q - \gamma$ terms ("prefix match") of d_{new} is greater than $1 - 2^{-e^{\frac{q(\gamma+1)}{2}}}$.*
2. *The probability that there exists a document $d \in \mathcal{D}$ that contains at least $q + \gamma$ arbitrary terms ("any match") of d_{new} is smaller than $\frac{1}{e^{(\gamma-2)q}}$.*

[1] In the *frequency table*, the most frequent term is at rank 1, the second most frequent term at rank 2 and so on.

Proof. 1. The probability that a document contains the prefix of length $q-\gamma$ of d_{new} is equal or greater than

$$\frac{1}{e}\cdot\ldots\cdot\frac{1}{e^{q-\gamma}} > \frac{1}{e^{\frac{(q-\gamma+1)^2}{2}}} > \frac{e^{\frac{q(\gamma+1)}{2}}}{e^{\frac{q^2}{2}}} = \frac{e^{\frac{q(\gamma+1)}{2}}}{n}.$$

This means the probability that there exists no document in $\mathcal{D}$ that contains the $(q-\gamma)$-prefix of d_{new} is equal to or smaller than

$$\left(1-\frac{e^{\frac{q(\gamma+1)}{2}}}{n}\right)^n < 2^{-e^{\frac{q(\gamma+1)}{2}}},$$

which follows from inequality $(*)$ (note that $e^{\frac{q(\gamma+1)}{2}} < n$ for $\gamma < q-1$). So with probability greater than $1-2^{-e^{\frac{q(\gamma+1)}{2}}}$ there exists a document $d \in \mathcal{D}$ that has all terms from the $(q-\gamma)$-prefix of d_{new}.

2. Let $d_{new} = \{a_1, \ldots, a_{\lceil \ln m \rceil}\}$, $s = q+\gamma$. We now estimate the probability that a random document has a large overlap with d_{new}:

$$\begin{aligned}
Pr(|d_{new}\cap d| \geq s) &\leq \sum_{j_1<\cdots<j_s} Pr(a_{j_1},\ldots,a_{j_s}\in d)\\
&\leq \sum_{j_1<\cdots<j_s} \exp(-\sum_{k=1}^{s}(j_k-1))\\
&= \sum_{\Delta_1\geq 0,\ \Delta_2,\ldots\Delta_s>0} \exp(-s\Delta_1-(s-1)\Delta_2-\cdots-\Delta_s)\\
&\leq \sum_{\Delta_1=0}^{\infty}\exp(-s\Delta_1)\cdot\prod_{k=2}^{s}\sum_{\Delta_k=1}^{\infty}\exp(-\Delta_k(s-k+1))\\
&= \frac{1}{1-\exp(-s)}\prod_{k=2}^{s}\frac{\exp(-(s-k+1))}{1-\exp(-(s-k+1))}\\
&\leq \exp(1-\sum_{k=2}^{s}s-k) \leq \exp\left(\frac{(s-2)^2}{2}\right) \leq \frac{1}{e^{\frac{q^2}{2}}\cdot e^{(\gamma-2)q}} \leq \frac{1}{n\cdot e^{(\gamma-2)q}}.
\end{aligned}$$

The probability that not a single document in $\mathcal{D}$ contains at least $q+\gamma$ terms of d_{new} is equal to or greater than

$$\left(1-\frac{1}{n\cdot e^{(\gamma-2)q}}\right)^n > 1-\frac{1}{e^{(\gamma-2)q}},$$

which follows from inequality $(**)$ (note that $e^{(\gamma-2)q} \geq 2$ for $3 \leq \gamma < q$). Therefore, we proved that the probability that any document matches d_{new} at $q+\gamma$ arbitrary positions is smaller than $\frac{1}{e^{(\gamma-2)q}}$.

By Theorem 1 we can conclude that with very high probability there exists a document in $\mathcal{D}$ that matches a prefix of length $q-\gamma$, whereas with quite small probability there exists a document that has at least $q+\gamma$ common terms with d_{new} (at arbitrary positions). We get the following algorithm:

MaxInt Algorithm in the Zipf Model

Preprocessing:

1. For every document: Sort the term list according to the position of the term in the frequency table.
2. For every document: Generate the set of all possible *regular* $(q-\gamma)$-*lists*, i.e. generate all possible term subsets of size $(q-\gamma)$ containing one term from every group $P_1, \ldots, P_{q-\gamma}$.
3. Sort these regular lists and store for every list a pointer to the corresponding document.

Query: Find a regular $(q-\gamma)$-list having the maximal common prefix with the query document by binary search. Return the document corresponding to this $(q-\gamma)$-list.

The running time is as follows (for *average* case[2] analysis we assume that the length of term lists is $\log m$, for *worst* case analysis the length is m):

	average	worst
Step 1	$\mathcal{O}(n \cdot \log m \cdot \log\log m)$	$\mathcal{O}(n \cdot m \cdot \log m)$
Step 2	$n^{1+o(1)}$	$\mathcal{O}(n^2 \cdot \log n)$
Step 3	$\mathcal{O}(\log m \cdot n \cdot \log n)$	$\mathcal{O}(m \cdot n \cdot \log n)$
Query	$\mathcal{O}(\log m \cdot \log n)$	$\mathcal{O}(m \cdot \log n)$

Let us explain the estimations from the second line. The number of all possible regular $(q-\gamma)$-lists is equal to

$$|P_1| \cdot \cdots \cdot |P_{q-\gamma}| \le \prod_{k=1}^{q-\gamma} e^k < e^{q^2/2} = n$$

Therefore, a single document can generate at most n different regular $(q-\gamma)$-lists, the $\log n$ factor arises from the size of a single list. Let us prove the bound for the average case. The probability of containing some fixed regular $(q-\gamma)$-list is $n^{-1+o(1)}$. Summing over all possible lists we see that the expected number of generated regular lists per document is at most $n^{-1+o(1)} \cdot n = n^{o(1)}$. Therefore, the expected time for the indexing stage is $n^{1+o(1)}$.

One can try to improve the accuracy of our algorithm by finding a "maximal prefix with at most one difference to the query document". A famous technique called "indexing with errors" [2,5] might be useful for such an extension.

[2] Only for the average case our constraints from Section 1 are preserved.

3 MaxInt in the Hierarchical Schema

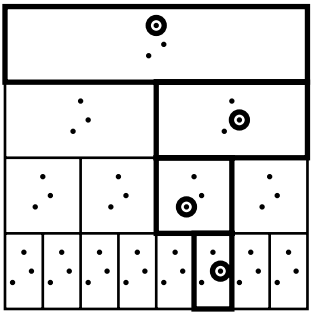


Fig. 1. Hierarchical Schema

Let $|\mathcal{D}| = 2^k$, $k \in \mathbb{N}$, $k \geq 3$, and $|d| = k$ for every $d \in \mathcal{D}$. Let $|\mathcal{T}| = (2^k - 1) \cdot k$ be the number of different terms. A *hierarchical schema* is a table with k levels, level 1 to level k. Level i is divided to 2^{i-1} cells, $1 \leq i \leq k$. Every cell contains k terms. A document collection based on this schema can be generated as follows: Choose a random cell on level k. Then mark all cells that are above it and choose one random term in every marked cell. Now each document corresponds to a path from the top cell to a bottom cell (see Figure 1).

Remark 3. We claim that the hierarchical schema follows *Zipf's law*. To be more precise, the following holds: For every level the product of expected frequency and expected frequency rank of a term is the same. Indeed, the expected frequency of a term on level i is calculated by the formula $\frac{2^k}{2^{i-1} \cdot k}$. The expected rank of a term is calculated by the formula $(2^{i-1} - 1) \cdot k + 2^{i-2} \cdot k$. Hence, the product between frequency and frequency rank (divided by 2^k) is equal to

$$\frac{2^k}{2^{i-1} \cdot k} \cdot (\frac{1.5}{2^k} \cdot 2^{i-1} - 1) \cdot k \approx \frac{3}{2}$$

and hence Zipf's law applies.

Again we introduce *magic levels* to give statements about the most probable size of maximal intersection. The magic levels are

$$q = \frac{k}{1 + \log k} \qquad \text{and} \qquad q' = \frac{k}{\log k}.$$

Theorem 2 (Magic Levels for Hierarchical Schema). *Let $k \geq 4$ and $2 \leq \gamma < q$ be a positive integer. Fix a query document d_{new} following hierarchical schema and take a randomly generated document collection $\mathcal{D}$:*

1. *The probability that there exists a document $d \in \mathcal{D}$ that matches the same terms from top $q-\gamma$ levels ("prefix match") of d_{new} is greater than $1-2^{-(2k)^\gamma}$.*
2. *The probability that there exists a document $d \in \mathcal{D}$ that matches at least $q' + \gamma$ arbitrary terms ("any match") of d_{new} is smaller than $\frac{2}{k^{\gamma-1}}$.*

Proof. 1. The number of different prefixes of length $q - \gamma$ is equal to

$$k(2k)^{q-\gamma-1} < 2^{(1+\log k)(q-\gamma)} = 2^{(1+\log k)(\frac{k}{1+\log k}-\gamma)} = 2^k \cdot (2k)^{-\gamma}.$$

So the probability that a new document does not match a prefix of length $q-\gamma$ with any document from $\mathcal{D}$ is smaller than

$$\left(1-\frac{(2k)^\gamma}{2^k}\right)^{2^k} < 2^{-(2k)^\gamma}.$$

Since $(2k)^\gamma < 2^k$, this inequality follows from inequality $(*)$. We get that the probability that there exists a document with the same prefix as d_{new} of length $q-\gamma$ is greater than $1-2^{-(2k)^\gamma}$.

2. Let $t \geq q'+\gamma$ be the last position where the terms of d and d_{new} match. We want to estimate the probability that d_{new} matches at least $q'+\gamma$ terms at arbitrary positions with d. The probability that the first t terms (beginning at the top) of d and d_{new} are all in the same cells is 2^{1-t}. The probability that at least $q'+\gamma$ terms are matched on some fixed positions is equal or smaller than $\left(\frac{1}{k}\right)^{q'+\gamma} \cdot \left(\frac{k-1}{k}\right)^{t-q'-\gamma}$. An upper bound for the number of different possibilities of matching at least $q'+\gamma$ out of t terms is 2^t. Since the factor $\left(\frac{k-1}{k}\right)^{t-q'-\gamma}$ is smaller than 1, overall we get that the probability that d_{new} matches at least $q'+\gamma$ terms at arbitrary positions with d is equal to or smaller than

$$k \cdot 2^t \cdot \left(\frac{1}{k}\right)^{q'+\gamma} \cdot 2^{1-t} = 2 \cdot k \cdot \left(\frac{1}{k}\right)^{q'+\gamma} = 2 \cdot \left(\frac{1}{k}\right)^{q'+\gamma-1}.$$

The factor k is due to the fact that we need to consider all possible levels for the last matched position t. Now the probability that no document matches at $q'+\gamma$ arbitrary positions with d_{new} is greater than

$$\left(1-2\cdot\left(\frac{1}{k}\right)^{q'+\gamma-1}\right)^{2^k} = \left(1-\frac{1}{2^k \cdot \frac{k^{\gamma-1}}{2}}\right)^{2^k} > 1-\frac{2}{k^{\gamma-1}},$$

which follows from inequality $(**)$, since $\gamma \geq 2, k \geq 4$. So the probability that we get any match in $\mathcal{D}$ is smaller than $\frac{2}{k^{\gamma-1}}$.

By Theorem 2 we get an analogue algorithm as the one for the Zipf model:

MaxInt Algorithm in the Hierarchical Schema

Preprocessing:

1. For every document: Sort the term list according to the hierarchical schema, i.e. according to the levels in which the terms appear.
2. Sort the documents according to their corresponding cell paths, i.e. documents that correspond to the leftmost path in the schema are at the beginning of the sorted list. Documents that correspond to the same cell path are sorted lexicographically.

Query: Find a document having the maximal common prefix with the query document by binary search.

Step 1 of preprocessing needs time $\mathcal{O}(2^k \cdot k \cdot \log k)$. Step 2 needs time $\mathcal{O}(2^k \cdot k^2)$. So the overall running time of the preprocessing is in $\mathcal{O}(2^k \cdot k^2)$. The query time is in $\mathcal{O}(k^2)$.

4 Further Work

In this paper we have shown that assumptions on the random nature of the input can lead to *provable* time and accuracy bounds for MaxInt. Also, we have discovered a MaxInt threshold phenomenon in two randomized models.

The next step is to understand it better. Does it hold for other randomized models from [7], especially the preferential attachment model? Is it still true when we replace the Zipf distribution by a power law distribution? Does it hold in the real life networks? Can we introduce randomized models for sparse vector collections and find a similar effect there? Of course, the most challenging problem is to find an exact algorithm for MaxInt or to prove its hardness. What are other particular cases or assumptions that have efficient MaxInt solutions? On the other hand, we have a very particular subcase for which we still do not believe in a positive solution. Hence, we ask for a hardness proof for the following *on-line inclusion problem*. Note that we have a constraint on space for preprocessing, not time. A related problem but with a much stronger restriction on preprocessing space was proven to be hard by Bruck and Naor [1].

On-line Inclusion Problem

Database: A family $\mathcal{F}$ of 2^k subsets of $[1 \dots k^2]$.
Query: Given a set $f_{new} \subseteq [1 \dots k^2]$, decide whether there exists an $f \in \mathcal{F}$ such that $f_{new} \subseteq f$.
Constraints: Space for preprocessed data $2^k \cdot poly(k)$.
Query time $poly(k)$.

Our algorithm in Section 3 uses polylogarithmic time (in the number of documents) but it returns only an approximate solution with high probability (not every time). Can we get an optimal solution or at least a *guaranteed* approximation by relaxing the time constraint to *expected* polylogarithmic time?

Acknowledgments. The authors would like to thank Holger Petersen, Hinrich Schütze, Kirill Shmakov and Jason Utt for their useful comments and fruitful discussions.

References

1. Bruck, J., Naor, M.: The hardness of decoding linear codes with preprocessing. IEEE Transactions on Information Theory 36(2), 381–385 (1990)
2. Cole, R., Gottlieb, L.-A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: STOC '04, pp. 91–100 (2004)

3. Kleinberg, J.M.: Two algorithms for nearest-neighbor search in high dimensions. In: STOC '97, pp. 599–608 (1997)
4. Lifshits, Y.: A Guide to Web Research. Materials of mini-course at Stuttgart University (2007), Available at
http://logic.pdmi.ras.ru/~yura/webguide.html
5. Maa ß, M.G., Nowak, J.: A new method for approximate indexing and dictionary lookup with one error. Inf. Process. Lett. 96(5), 185–191 (2005)
6. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (2002)
7. Newman, M.: The structure and function of complex networks. SIAM Review 45(2), 167–256 (2003)
8. O'Connor, M., Herlocker, J.: Clustering items for collaborative filtering. In: SIGIR '01, Workshop on Recommender Systems (2001)
9. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: SODA '93, pp. 311–321 (1993)
10. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. 38(2), 6 (Article No.) (2006)

A Note on Specialization of Interpreters

Alexei Lisitsa[1] and Andrei P. Nemytykh[2,⋆]

[1] Department of Computer Science, The University of Liverpool
alexei@csc.liv.ac.uk
[2] Program Systems Institute of Russian Academy of Sciences
nemytykh@math.botik.ru

Abstract. Given a program with two arguments `p(x,y)`. Let the first argument x_0 be fixed. The aim of program specialization with respect to the known x_0 is to construct an optimized program p_{x_0}`(y)` such that p_{x_0}`(y) = p(`x_0`,y)`. Specialization of interpreters with respect to programs is well known problem. In this paper we argue that specialization of interpreters with respect to data may be seen as program verification.

Keywords: Program specialization, supercompilation, program verification, cache coherence protocols.

1 Specialization of Interpreters with Respect to Data

Given two programming languages `L, M` and the semantics of `L` described by an interpreter `int(p,d)` written in `M`, where the first argument stands for the source `L`-programs and the second ranges over the data of `L` language. There is a famous task for automated specialization of the interpreter with respect to the first argument `int(`p_0`,d)`, i.e. the program p_0 is known while the data `d` is unknown. Specialization has to generate a residual program `q` such that `q(d) = int(`p_0`,d)`, where the equality holds whenever the pair `(`p_0`,d)` belongs to the domain of the interpreter. Certainly the `q` is written in `M`: consequently `q` can be seen as a result of compilation of p_0 from `L` to `M`. The goal is to construct an *optimal* `q`. The formulated problem is both undecidable (of course) and interesting. A lot of work was devoted to approximation of the problem (see [8,9,11,12,20,26] for examples).

In this paper we show that specialization of interpreters with respect to data may be reasonable and leads to interesting applications in verification. We consider the following specialization problem `int(p,d,`d_0`)`, where the known part of the data is separated from the unknown part. Firstly, for the sake of simplicity, let us think of the languages `L,M` as predicative languages, i.e. the languages defining only (partial) predicates rather than arbitrary recursive functions. Another assumption is that the interpreter `int` terminates for all possible values of

⋆ The second author is supported by the Program for Basic Research of the Presidium of Russian Academy of Sciences (as a part of "Development of the basis of scientific distributed informational-computing environment on the base of GRID technologies"), and the Russian Ministry of Sciences and Education (contract 2007-4-1.4-18-02-064).

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 237–248, 2007.

its arguments, but for some values it may terminate with abnormal stop. The abnormal stop indicates the input values of the arguments are outside of the domain. Let us have a robust specializer generating a residual program `q` defining an extension of the partial predicate defined by the problem `int(p,d,`d_0`)`. Assume that `q` is a partial constant function `TRUE` or `FALSE` and this property is expressed explicitly in syntax of `q`. For example, `q` does not contain any syntactic construction with the semantics `return FALSE` (in the case of the `TRUE` partial constant). Thus, we assume that specializer was weak enough not to be able to optimize the predicate `int` as

```
q(p,d) { return TRUE; }
```

but was strong enough to eliminate all constructors of the form `return FALSE;`. In such a case, the result of specialization can be considered as a proof of the (partial) constant property. The termination property of `int` mentioned above guarantees that the domain of the original partial predicate is not empty. Notice that we assume that the specializer is allowed to extend the domain of the original partial predicate. This provides additional important possibilities for specialization (see [21]) and distinguishes supercompilation, the technology of specialization we use (see Section 2.1), from other well known specialization technologies (e.g. partial evaluation [11]).

Consider now a more complicated interpreter. Let `int` be a composition $\varphi \circ$ `fint` of a functional language interpreter `fint` (i.e. not only predicative) and a predicate-postcondition φ testing the result of `fint`-interpretation. Now the `TRUE`-constant property of the residual program `q` means all source programs `p` satisfy the post-condition φ (in the given context of specialization). In such a case we conclude the specializer solved a verification problem. The composition $\varphi \circ$ `fint` can be encoded in various ways.

The idea of using supercompilation for proving that `p` implies `q` by composing a filter for `p` (a function `f` such that `f(x) = x if p(x)` and undefined if not) with a predicate for `q` is not new and is originated by V.F. Turchin (see [27,28,29]). The following section is devoted to a non-trivial application of the idea: to verification of a *parameterized* distributed system.

2 An Interesting Example

G. Delzanno [4] specifies a class of *parameterized* cache coherence protocols with global correctness conditions in terms of Extended Finite State Machines (EFSM) [3], which are essentially transition systems with data variables ranging over non-negative numbers. EFSM-transitions are linear transformations with the guards expressed as linear constraints. We find it convenient to consider such models as non-deterministic pebble games, where variables and their values are represented by locations (baskets) and numbers of pebbles in baskets, respectively. Then evolution of EFSM can be thought of as non-deterministic movement of pebbles between baskets and safety properties of the protocols are expressed as systems of linear inequalities imposed on the numbers of the pebbles in the baskets.

Given a initial state `d` of such a game and a finite sequence `p` of the moves, the `p` determines the game state reached as the result of the last move. Thus, inherently, the `p(d)` is a call for a program transforming the states of the game and the result of the program `p` is the last state. We treat the `fint`-interpreter as an interpreter of the programming language defined by the game rules: each program is a *finite* sequence of the moves. Notice that not all moves are applicable to all game states; an attempt of execution of non-applicable move leads to an abnormal stop of the program. The post-condition φ encodes the safety property of the corresponding protocol. Hence, in the case of strong specialization (as described above) of the composition $\texttt{int} = \varphi \circ \texttt{fint}$ with respect to a part of the data `d`, a `TRUE`-constant residual program means successful verification of the protocol (in the given context) was happened.

As an example, consider specification of MESI protocol given G. Delzanno [4].

```
(RH) modified + shared + execlusive ≥ 1 → .
(RM) invalid ≥ 1 → invalid' = invalid - 1, exclusive' = 0, modified' = 0,
                   shared' = shared + exclusive + modified + 1 .
(WH1) modified ≥ 1 → .
(WH2) exclusive ≥ 1 → exclusive' = exclusive - 1, modified' = modified + 1 .
(WH3) shared ≥ 1 → shared' = 0, exclusive' = 1, modified' = 0,
                   invalid' = invalid + modified + exclusive + shared - 1 .
(WM) invalid ≥ 1 → shared' = 0, exclusive' = 1, modified' = 0,
                   invalid' = invalid + modified + exclusive + shared - 1 .
```

Here *modified, exclusive, shared, invalid* are non-negative integer variables of EFSM model, which represent *counting abstraction* of original parameterised automata model: the names denote various states of the automaton (cache) and the values of the variables keep track of the number of automata in corresponding states. The rules `(RH)-(WM)` define the dynamic of EFSM model. Starting with some initial evaluation of the variables, the system may apply non-deterministically any of the rules. In the case the quard of a rule (its left-hand part) is satisfied in a current state (evaluation of all variables, i.e integer vector), the update expressed by the right-hand side of the rule is executed. Primed variable names are used in updates to denote updated values.

The parameterized initial configuration of MESI protocol, i.e. the start state of the MESI game, is expressed as

$$\texttt{invalid} \geq 1,\ \texttt{modified} = 0,\ \texttt{shared} = 0,\ \texttt{exclusive} = 0.$$

The potentially *unsafe* states are expressed with the two following constraints

$$\texttt{--}\ (C_1)\ \texttt{invalid} \geq 0,\ \texttt{modified} \geq 1,\ \texttt{shared} \geq 1,\ \texttt{exclusive} \geq 0;$$
$$\texttt{--}\ (C_2)\ \texttt{invalid} \geq 0,\ \texttt{modified} \geq 2,\ \texttt{shared} \geq 0,\ \texttt{exclusive} \geq 0.$$

The rule names are the move names of the game corresponding to this protocol. Finite sequences of the names describe possible evolutions of the non-deterministic protocol MESI.

We did a number of successful experiments with verification of such kinds of protocols by means of automatic specialization, as we explained above. The following subsection is devoted to some principal technical details of our experiments.

2.1 On Properties of a Program Encoding of the Protocols

We implemented such a MESI-interpreter `fint` in a functional programming language REFAL-5 [30,31], wherein the main syntactical expressing tools are pattern matching and term rewriting. (See the Appendix 3 for additional information on the language.) Nevertheless, we explain some properties of the encoding (as many of them as it is possible) in language independent terms. The encoding follows straightforward the specification. Its *logical* structure looks as follows:

```
fint(ps, <invs,mods,shars,excs>) {
 if null(ps) then return <invs,mods,shars,excs>;
    else fint( cdr(ps), move( car(ps), <invs,mods,shars,excs> ) );
}

move(p, <invs,mods,shars,excs>) {
// RH - trivial. Nothing to do.
if p =? RM && !null(invs)
   then return <cdr(invs),[],[], I : append(mods, append(shars, excs))>;
// WH1 - trivial. Nothing to do.
else if p =? WH2 && !null(excs)
   then return <invs, I : mods, shars, cdr(excs)>;
else if p =? WH3 && !null(shars) then
   return <append(invs, append(mods,append(cdr(shars), excs))),[],[],I>;
else if p =? WM && !null(invs) then
   return <append(cdr(invs), append(mods,append(shars, excs))),[],[],I>;
}
```

Here we use capital letters to name constants and the angle brackets to denote the tuples. Numbers are represented in the unary system, i.e. `0 ::= [ ], n+1 ::= I : n`. Note the last `else`-alternative is undefined. The φ predicate testing the data consistency is

```
phi(<invs,mods,shars,excs>) {
 if !null(mods) && !null(shars) then  return FALSE;
 else if !null(cdr(mods)) then  return FALSE;
 else return TRUE;
}
```

We incorporate a call for the `phi` into the body of `fint` (in its `return` statement) to organize the composition `int` $= \varphi \circ$ `fint`:

```
int(ps, <invs,mods,shars,excs>) {
 if null(ps) then return phi(<invs,mods,shars,excs>);
    else fint( cdr(ps), move( car(ps), <invs,mods,shars,excs> ) );
}
```

Such a kind of the encoding of the *composition* is crucial for successful automatic verification of the protocol. Note for any values of the parameters `ps,invs,mods,shars,excs` the call `int(ps, <invs,mods,shars,excs>)` terminates (possibly it falls into an abnormal stop).

2.2 On Properties of the Specializer

In our experiments we used the SCP4 supercompiler [22,13,25,24] as a specializer. In this section we consider a number of properties of SCP4 concerning to specialization of the protocol class described by G. Delzanno [4]. To facilitate this, we critically simplify some real mechanisms of the SCP4 and get rid of some of them at all by making an explicit assumption that the `append` function is associative and represented by concatenation[1]. Symbolically that can be expressed as follows.

```
append(xs, ys) {
     return xs : ys;
}
```

Every call for the `append` has to be replaced with such its result in our encoding. An expression is defined by the following grammar:

`expr ::= [ ] | SYMBOL : expr | VARIABLE : expr`,

where the variables range over the set of the expressions and the symbols range over identifiers.

Definition 1. *A configuration is any syntactical composition of functions and the tuple constructor:*

conf ::= expr | <conf$_1$, ..., conf$_n$> | FuncName(conf$_1$, ..., conf$_n$).

Definition 2. *Given two configurations* `conf`$_1$ *and* `conf`$_2$*, a configuration* `gconf` *=* gen(`conf`$_1$`,conf`$_2$) *is a generalization of* `conf`$_1$`, conf`$_2$ *iff there exist substitutions* ϑ_1, ϑ_2 *of the variables of* `gconf` *such that the results of the substitutions in* `gconf` *coincide correspondingly with* `conf`$_1$`, conf`$_2$*.*

Definition 3. *A set* `MSG` *of the generalizations of* `conf`$_1$`, conf`$_2$ *is said to be the set of most specific generalizations of* `conf`$_1$`, conf`$_2$ *iff for any generalization* `gconf` *=* gen(`conf`$_1$`,conf`$_2$) *there exists a substitution* ϑ *of the variables of* `gconf` *such that the result of the substitution in* `gconf` *coincides with an element of* `MSG`*.*[2]

The task to be specialized is the following:

`int(ps, <I : invs, [], [], []>)`

where `ps` and `invs` are parameters. Firstly, we suppose the SCP4 is correct itself. The specializer is a general tool (for specialization) knowing nothing in advance

[1] In fact, the REFAL concatenation constructor is associative and there exists another constructor allowing imitate a LISP style `append`. See [30] and Sections 3, 3 for additional explanations.

[2] The associativity of the concatenation causes `MSG` can contain more than one element. E.g. both the following generalizations are most specific: gen(`I , I : I`) `= I : xs` and gen(`I , I : I`) `= ys : I`.

on the specific goal of our specialization that is the hypothesis we have to prove. The hypothesis states the correctness property of the MESI protocol. SCP4 successfully proves the hypothesis, i.e. it automatically verifies the protocol (see Section 1). Using its general mechanisms, SCP4 discovers that the hypothesis can be proved simultaneously with another hypothesis

```
int(ps, <invs, [], I : shars, []>)
```

by induction on the length of the `ps` program. And the specializer proves both hypotheses *simultaneously*, i.e. referring one to another in the course of induction. It is important to stress the second hypothesis is generated fully automatically on the base of analysis and generalizing the possible parameterized configurations of the MESI protocol: they are configurations that appear during unfolding of the potentially infinite tree of all possible computations (of the program encoding the protocol) starting with the initial configuration. It is well known that more general hypotheses can often be proved by induction more easily than the hypotheses from which they were generated.

Let two different configurations $\texttt{conf}_1$, $\texttt{conf}_2$ be given on a branch originating from the root of the tree of all possible computations and ending in $\texttt{conf}_2$. Let the SCP4 decide to generalize $\texttt{conf}_1$, $\texttt{conf}_2$. The result of such a decision is cancellation of the subtree with the root $\texttt{conf}_1$, excluding $\texttt{conf}_1$ itself, and replacement of the upper configuration $\texttt{conf}_1$ with $gen(\texttt{conf}_1,\texttt{conf}_2)$. That is to say, the generalization takes into account only configurations belonging to the same branch of the directed tree. Consequently, in general, the result of specialization depends on the strategy choosing an order of analysis of the branches, which in its turn depends on the order of the branches of the program being analyzed.

Another crucial problem must be solved by the specializer is how to separate the protocol configurations corresponding the base cases of the induction from its inductive cases. In the given example the SCP4 decides four configurations are base cases[3]. In fact, the crucial separation once again happens full automatically and it is a consequence of the generalization procedure used by SCP4. The decision procedure is based on a well-quasi-ordering $\prec$ of the configurations modulo variables' names, which is a variant of the Higman-Kruskal relation [10,14,15]: all syntactical constructors are monotone with respect to $\prec$ and they are matched with the $\prec$. Let two configurations $\texttt{conf}_1$, $\texttt{conf}_2$ belonging to the same branch, such that the first configuration is upper than the second along the branch, be given.

Generalization. *decides to generalize* $\texttt{conf}_1$, $\texttt{conf}_2$ *iff*

- $\texttt{conf}_1 \prec \texttt{conf}_2$, *i.e. strictly greater;*
- *and there is a configuration* `conf` *which is a most specific generalization of* $\texttt{conf}_1$ *and* $\texttt{conf}_2$, *such that there are no occurences of* `[]` *in* $\texttt{conf}_1$ *and* `SYMBOL` *in* $\texttt{conf}_2$, *generalized to the same variable in* `conf`.

[3] Some additional explanations are given in Section 3.

The second condition allows to separate of the base cases from the inductive cases. For example, the generalization does not generalize the following two configurations $\texttt{conf}_1 = \texttt{()}$ and $\texttt{conf}_2 = \texttt{(I)}$. The necessity of generalization of the substructures `[ ]` and `I` to a variable causes that. The second example demonstrates a successful most specific generalization: $gen(\texttt{(), (I : I)}) = \texttt{(xs)}$. Here `xs` is a variable. See [18,22] for the details.

2.3 A Parameter of the Encoding

Our encoding of protocols (see the MESI example above) is not unique and it has a natural parameter - *the order* in which clauses corresponding to different moves in protocol games are expressed. For the MESI we have 24 (i.e. 4!) different permutations for these clauses and thus 24 different specialization tasks which one may try in order to verify the protocol.

The following statement shows that the part of the result of specialization (i.e. the proof of the hypothesis, in which we are interested) does not depend on the order in which moves are expressed in the program.

Proposition 1. *Let σ be a permutation of the cases* `RM,WH2,WH3,WM`[4] *of the MESI specification and* $\texttt{int}_\sigma$ *be the interpreter corresponding to σ. Consider the initial configuration*

$$\texttt{start = int}_\sigma\texttt{(ps, <I : invs,[],[],[]>).}$$

Let there exists a permutation σ_0 such that the SCP4 specializer verifies the MESI protocol encoded with $\texttt{int}_{\sigma_0}$*. Then for any σ the SCP4 starting with* `start` *and transforming the* $\texttt{int}_\sigma$ *constructs a residual program* $\texttt{p}_\sigma\texttt{(ps,invs)}$ *(the result of specialization) not containing* `return FALSE`*.*

Proof. (Sketch) The existence of the automatic verification proof for the $\texttt{int}_{\sigma_0}$ encoding implies that *no* non-parameterized configuration `conf` of $\texttt{int}_{\sigma_0}$ violates the safety property and moreover that the generalization algorithm does not break the safety property of `conf`. On the other hand the unknown value of `ps` and the properties of our encoding guarantee that the SCP4 considers all possible evolutions of the protocol and tests the MESI safety condition at *each step* (each clock cycle) of its development.

Now the proposition is immediately implied by the following observation: for any σ the set of all possible evolutions of $\texttt{int}_\sigma$ coincides with the set of all possible evolutions of $\texttt{int}_{\sigma_0}$. Once again the safety property is not violated and this is checked at *each clock cycle*. Thus the residual program $\texttt{p}_\sigma\texttt{(ps,invs)}$ never returns `FALSE`. The termination property of the original program implies $\texttt{p}_\sigma\texttt{(ps,invs)}$ contains a `return TRUE` operator such that the operator is reachable. □

At first glance, the necessity of such a proof gives rise to doubts: it is not a problem to specialize the 24 different tasks by the SCP4. We did that and the

[4] We exclude the trivial cases.

SCP4 proved the proposition. But the MESI example is very small and has been chosen just for demonstration purposes. In fact, the same proposition holds for a number of protocols (see Section 3) being experimented. But for some of them the number of permutations does not allow hope that such a proof can be done by experiments. For example, the MSI protocol will take about two months for such experiments even though specialization of a single fixed MSI encoding is quick. The arguments in the proof given above hold for all protocols mentioned in the following section.

3 Discussion and Conclusions

On the structure of the automatic proof. The graph given in Figure 1 represents the structure of the inductive proof of the MESI safety condition. The graph was constructed from the residual program and reflects its syntactical properties. The `[1]` node corresponds to the main theorem proved by the SCP4 supercompiler, the `[3g]` node corresponds to the generalized hypothesis (see Section 2.2), the `[5]` lemma stands for an additional induction hypothesis encountered in regular development of the tree of all possible computations, that was not constructed with the generalization. The `TRUE` cases are the base cases of the induction. The solid arrows denote choice between different cases, while the dashed arrows mean the induction steps. The proofs of the `[3g]` generalized hypothesis and the `[5]` lemma refer one to the other, i.e. these two statements become true *simultaneuosly*.

On the encoding. The graph given in Figure 1 represents the proof of the simplified encoding as explained in Section 2.2. The general encoding with the recursive `append` function such that it is not known *in advance* to be associative

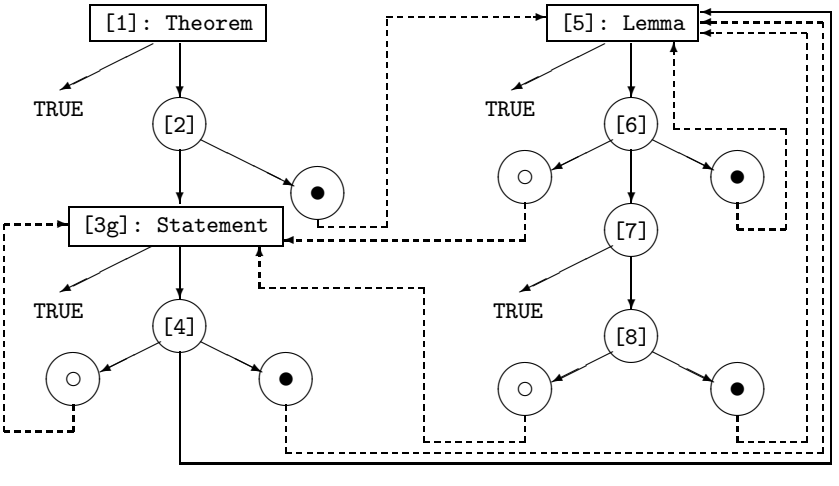


Fig. 1. Graph of the authomatic proof

was also specialized with the SCP4. The corresponding residual program once again does not contain `return FALSE` operators. Thus the MESI protocol was successfully verified under this general encoding. The corresponding automatic proof is much more complicated than the given above. Proposition 1 still holds, but introduction of additional SCP4 tools are needed to elaborate on the proof of such a proposition.

Other protocols. In addition to the MESI protocol we successfully verified (by specialization of interpreters with respect to data) the following cache coherence protocols with conditions of global correctness: IEEE Futurebus+, MOESI, MSI, "The University of Illinois", DEC Firefly, "Berkeley", Xerox PARC Dragon [4,5]. All these protocols are specified analogously to description given in Section 2. Safety properties for a formal model of Steve German's directory-based consistency protocol specified in a slightly more complicated fashion [7] was successfully verified by the SCP4 as well.

First steps in constructing of a theoretical model of supercompilation process in the particular context of parameterized testing by specialization with respect to data, were done in [19], but still there is much work to be done. Safety properties of majority of systems we have considered so far are expressed by systems of linear inequalities and these properties are *non-recursive* (i.e can be tested by non-recursive programms). It is certainly, recursive correctness properties will lead to considerable complication of the specialization process of the programs encoding the composition of the corresponding interpreters and testing predicates. It would be very interesting to find an example of a protocol with such a recursive correctness property, which can be verified by a specializer using our approach. The Alternating Bit Protocol is an example of an infinite-states system, for which the SCP4 fails to construct a proof of correctness of this protocol specified as in [1].

Acknowledgements. The authors thank anonymous referees for several insightful comments that led to a substantial improvement of the paper.

References

1. Abdulla, P., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
2. Bundy, A.: The Automation of Proof by Mathematical Induction. Handbook of Automated Reasoning, 845–911 (2001)
3. Cheng, K.T., Krishnakumar, A.S.: Automatic Generation of Functional Vectros Using the Extended Finite State Machine Model. ACM Transactions on Design Automation of Electronic Systems 1(1), 57–79 (1996)
4. Delzanno, G.: Automatic Verification of Parameterized Cache Coherence Protocols. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 53–68. Springer, Heidelberg (2000)
5. Delzanno, G.: Automatic Verification of Cache Coherence Protocols via Infinite-state Constraint-based Model Checking, `http://www.disi.unige.it/person/DelzannoG/protocol.html`

6. Delzanno, G.: Verification of Consistency Protocols via Infinite-state Symbolic Model Checking, A Case Study. In: Proc. of FORTE/PSTV, pp. 171–188 (2000)
7. Delzanno, G., Bultan, T.: Constraint-Based Verification of Client-Server Protocols. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 286–301. Springer, Heidelberg (2001)
8. Futamura, Y.: Partial Evaluation of Computation Process — An Approach to a Compiler-Compiler. In: Systems. Computers. Controls. 2(5), 45–50 (1971)
9. Futamura, Y., Nogi, K., Takano, A.: Essence of generalized partial computation. Theoretical Computer Science 90, 61–79 (1991)
10. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. 2(7), 326–336 (1952)
11. Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Prentice Hall International, Englewood Cliffs (1993)
12. Jones, N.D.: What not to do when writing an interpreter for specialization. In: Danvy, O., Thiemann, P., Glück, R. (eds.) Partial Evaluation. LNCS, vol. 1110, pp. 216–237. Springer, Heidelberg (1996)
13. Korlyukov, A.V.: User manual on the Supercompiler SCP4 (in Russian) (1999), http://www.refal.net/supercom.htm
14. Kruskal, J.B.: Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. Trans. Amer. Math. Society 95, 210–225 (1960)
15. Leuschel, M.: Homeomorphic Embedding for Online Termination. In: Flener, P. (ed.) LOPSTR 1998. LNCS, vol. 1559, pp. 199–218. Springer, Heidelberg (1999)
16. Lisitsa, A., Nemytykh, A.P.: Verification of parameterized systems using supercompilation. A case study. In: Hofmann, M., Loidl, H.W. (eds.) Proc. of the Third Workshop on Applied Semantics (APPSEM05), Fraunchiemsee, Germany. Ludwig Maximillians Universitat Munchen (2005), Accessible via: ftp://www.botik.ru/pub/local/scp/refal5/appsem_verification2005.ps
17. Lisitsa, A.P., Nemytykh, A.P.: Work on errors (in Russian). In: Program Systems: Theory and Applications (Programmnye systemy: teoriya i prilozheniya) Fizmatlit, Moscow, vol. 1, pp. 195–232 (2006), Accessible via: ftp://www.botik.ru/pub/local/scp/refal5/psta_errors2006.pdf
18. Lisitsa, A.P., Nemytykh, A.P.: Verification as a Parameterized Testing (Experiments with the SCP4 Supercompiler). Programmirovanie. 1 (2007) (In Russian). English translation in J. Programming and Computer Software, 33(1), 14–23 (2007)
19. Lisitsa, A.P., Nemytykh, A.P.: Reachability Analysis in Verification via Supercompilation. Accepted by the Workshop on Reachability Problems - RP07
20. Mogensen, T.: Evolution of Partial Evaluators: Removing Inherited Limits. In: Danvy, O., Thiemann, P., Glück, R. (eds.) Partial Evaluation. LNCS, vol. 1110, pp. 303–321. Springer, Heidelberg (1996)
21. Nemytykh, A.P.: A Note on Elimination of Simplest Recursions. In: Proc. of the ACM SIGPLAN Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation, pp. 138–146. ACM Press, New York (2002)
22. Nemytykh, A.P.: The Supercompiler SCP4: General Structure (extended abstract). In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 162–170. Springer, Heidelberg (2004)
23. Nemytykh, A.P.: Playing on REFAL. In: Proc. of the International Workshop on Program Understanding, A.P. Ershov Institute of Informatics Systems, Syberian Branch of Russian Academy of Sciences, pp. 29–39 (2003), Accessible via: ftp://www.botik.ru/pub/local/scp/refal5/nemytykh_PU03.ps.gz
24. Nemytykh, A.P.: The Supercompiler SCP4: General Structure. Moscow, URSS (A book in Russian, to appear)

25. Nemytykh, A.P., Turchin, V.F.: The Supercompiler SCP4: sources, on-line demonstration (2000), http://www.botik.ru/pub/local/scp/refal5/
26. Pettorossi, A., Proietti, M.: Transformation of logic programs: Foundations and techniques. J. of Logic Programming 19, 20, 261–320 (1994)
27. Turchin, V.F.: The use of metasystem transition in theorem proving and program optimization. In: de Bakker, J.W., van Leeuwen, J. (eds.) Automata, Languages and Programming. LNCS, vol. 85, pp. 645–657. Springer, Heidelberg (1980)
28. Turchin, V.F.: The language Refal - The Theory of Compilation and Metasystem Analysis. Courant Computer Science Report, New York University, vol. 20 (February 1980)
29. Turchin, V.F.: The concept of a supercompiler. ACM Transactions on Programming Languages and Systems 8, 292–325 (1986)
30. Turchin, V.F.: Refal-5, Programming Guide and Reference Manual. New England Publishing Co. Holyoke, Massachusetts (1989), (electronic version: http://www.botik.ru/pub/local/scp/refal5/,2000)
31. Turchin, V.F., Turchin, D.V., Konyshev, A.P., Nemytykh, A.P.: Refal-5: sources, executable modules (2000), http://www.botik.ru/pub/local/scp/refal5/

Appendix

In this section we give an introduction to REFAL-5 and some additional information on supercompilation [29,22]. Emphasize that here the angle brackets have different semantics than it used in the previous sections.

Language REFAL-5. The language is a first-order strict functional language. A fragment of REFAL-5 [30] can be defined by the grammar given below.

```
program       ::= $ENTRY definition+
definition    ::= function-name { sentence;+ }
sentence      ::= pattern = expression
expression    ::= empty | term expression | function-call expression
function-call ::= <function-name expression>
pattern       ::= empty | term pattern
term          ::= SYMBOL | variable | (expression)
variable      ::= e.variable-name | s.variable-name
empty         ::=                        /* nothing */
```

REFAL data are defined by the following grammar:

$$d ::= (d_1) \mid d_1\ d_2 \mid \mathtt{SYMBOL} \mid \mathtt{empty}$$

Roughly speaking, a program in the fragment of REFAL is a term rewriting system. The semantics of the language is based on pattern matching. As usually, the rewriting rules are ordered for matching from the top to the bottom. The terms are generated using two constructors. The first is concatenation. It is binary, associative and is used in infix notation, which allows us to drop its parenthesis. The blank is used to denote concatenation. The second constructor is unary. It is syntactically denoted with its parenthesis only (that is without a name). The unary constructor is used for constructing tree structures. Every

function is unary. Empty sequence is a special basic ground term. This term is denoted with nothing and is called the "empty expression". It is the neutral element (both left and right) of concatenation. Below we use the meta-symbol `[]` for the empty expression. All other basic ground terms are named as "symbols". There exist two types of variables - *e.name* and *s.name*. An *e*-variable can take any expression as its value, an *s*-variable can take any symbol as its value. For every sentence its set of variables from the left side includes its set of variables from the right side; there are no other restrictions on the variables. Associativity of concatenation may cause pattern matching to be ambiguous on some patterns. For example, the following equation `e.1 e.2 = A B` has three solutions: 1) `e.1 = [], e.2 = A B`; 2) `e.1 = A, e.2 = B`; 3) `e.1 = A B, e.2 = []`; . In such cases the pattern matching chooses the solution with the minimal length of the datum assigned to the first *e*-variable (from the left to the right) and so on by induction. In our case the first solution `e.1 = [], e.2 = A B` will be chosen. The function `append` can be defined in REFAL as:

```
$ENTRY append { (e.xs) (e.ys) = e.xs e.ys; }
```

The supercompiler SCP4. Let us consider a program in a language, as well as a parameterized input entry of the program. Then such pair defines a partial mapping. By definition, a supercompiler is a transformer of such pairs. It unfolds a potentially infinite tree of all possible computations. The computations can depend on the values of the parameters that can be unknown during transformation. The supercompiler reduces in the process the redundancy that could be present in the original program. It folds the tree into a finite graph of states and transformations between possible configurations of the computing system. The folding procedure called generalization sometimes is able to lose some information of the structure of the arguments of the configurations. The arguments are generalized with the goal of producing the finite graph of states. Let us note the main differences between this technique and conventional partial evaluation [11]: 1) supercompilation works online, 2) exploits negative information on the ranges of parameters and on intersection of the ranges; 3) it is allowed to extend the domains of the partial functions defined by programs being transformed; 4) the original program definition is thrown away unchanged, i.e. the resulting definition is constructed solely based on the meta-interpretation of the source program rather than by a step-by-step transformation of the original program.

Efficient Computation in Groups Via Compression

Markus Lohrey[1] and Saul Schleimer[2]

[1] Universität Stuttgart, FMI, Germany
[2] School of Mathematics and Statistics, Rutgers University, Mathematics Department, New Brunswick, New Jersey, USA
lohrey@informatik.uni-stuttgart.de, saulsch@math.rutgers.edu

Abstract. We study the *compressed word problem*: a variant of the word problem for finitely generated groups where the input word is given by a context-free grammar that generates exactly one string. We show that finite extensions and free products preserve the complexity of the compressed word problem. Also, the compressed word problem for a graph group can be solved in polynomial time. These results allow us to obtain new upper complexity bounds for the word problem for certain automorphism groups and group extensions.

1 Introduction

The *word problem for finitely generated groups* is a fundamental computational problem linking group theory, topology, mathematical logic, and computer science. For a group G, finitely generated by Σ, it is asked whether a word over Σ and the inverses of Σ represents the identity element of G. The word problem was introduced in the pioneering work of Dehn from 1910 in relation with topological questions. It took about 45 years until Novikov and later independently Boone proved the existence of a finitely presented group with an undecidable word problem, see [22,31] for references. Despite this negative result, many natural classes of groups with decidable word problems were found. Prominent examples are finitely generated linear groups, automatic groups [12], and one-relator groups. With the advent of computational complexity theory the complexity of word problems became an active research area. For instance, it was shown that for a finitely generated linear group the word problem can be solved in logarithmic space [20,30], that automatic groups have quadratic time word problems [12], and that the word problem for a one-relator group is primitive recursive [5].

Group theoretic operations, which preserve (or moderately increase) the complexity of the word problem are useful for constructing groups with efficiently solvable word problems. An example of such a construction is the free product: it is not hard to see that the word problem for a free product $G * H$ can be reduced in polynomial time to the word problem for G and H. In this paper, we introduce a new technique for obtaining upper complexity bounds for word problems. This technique is based on data compression. More precisely, we use compressed representations of strings — so called *straight-line programs*, briefly SLPs — which are able to achieve exponential compression rates for strings with repeated subpatterns. Formally, an SLP $\mathbb{A}$ is a context-free grammar which generates exactly one string $\mathrm{eval}(\mathbb{A})$. Recently, SLPs turned out to be a very flexible compressed representation of strings, which is well-suited for studying algorithms on compressed data. For instance, several polynomial time algorithms for the

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 249–258, 2007.

pattern matching problem on SLP-compressed input strings were developed [13,19,23]. In [21], the first author started to investigate the *compressed word problem* for a finitely generated group G with finite generating set Σ. For a given SLP $\mathbb{A}$ that generates a string over Σ and the inverses of Σ it is asked whether $\mathrm{eval}(\mathbb{A})$ represents the identity element in G (actually, in [21] the compressed word problem for finitely generated monoids was studied). This problem is equivalent to the well-known circuit evaluation problem, where we ask whether a circuit over a finitely generated group G (i.e., an acyclic directed graph with leafs labeled by generators of G and internal nodes labeled by the group multiplication) evaluates to the identity element of G. In [3] this problem was investigated for finite groups, and it was shown that there exist finite groups, for which the circuit evaluation problem is complete for P (deterministic polynomial time).

In [3,21] the main motivation for studying the compressed word problem came from computational complexity theory. Since the input in the compressed word problem is given in a more compact form than in the ordinary word problem it can be expected that the compressed word problem is, in general, more difficult than the ordinary word problem. For instance, whereas the word problem for a finitely generated free group belongs to the class L (deterministic logspace) [20], the compressed word problem for a finitely generated free group of rank at least two is P-complete [21].[1]

In [28], the second author used the polynomial time algorithm for the compressed word problem for a free group in order to present a polynomial time algorithm for the ordinary word problem for the automorphism group of a free group, which answered a question from [17]. Hence, the compressed word problem is used in order to obtain better algorithms for the ordinary word problem. In this paper, we will continue this program and obtain efficient algorithms for a variety of word problems. In order to achieve this goal, we proceed in two steps:

In the first step (Section 3) we give connections between the compressed word problem for a group G and the word problem for some group derived from G. We prove three results of this kind:

- If H is a finitely generated subgroup of the automorphism group of a group G, then the word problem for H is logspace reducible to the compressed word problem for G (Prop. 2). This result is a straight-forward extension of Thm. 5.2 from [28].
- The word problem for the semidirect product $K \rtimes_\varphi Q$ of two finitely generated groups K and Q is logspace reducible to (i) the word problem for Q and (ii) the compressed word problem for K (Prop. 3).
- If K is a finitely generated normal subgroup of G such that the quotient G/K is an automatic group, then the word problem for G is polynomial time reducible to the compressed word problem for K (Prop. 4).

In the second step (Section 4) we concentrate on the compressed word problem. We prove the following results:

- If K is a finitely generated subgroup of G such that the index $[G : K]$ is finite, then the compressed word problem for G is polynomial time reducible to the compressed word problem for K (Thm. 1).

[1] It is believed, although not proven, that L is a proper subclass of P.

- The compressed word problem for a free product $G_1 * G_2$ is polynomial time reducible (under Turing reductions) to the compressed word problem for G_1 and G_2 (Thm. 2). This result even holds for the more general graph product construction [14] (Thm. 4).
- The compressed word problem for a graph group [11] can be solved in polynomial time (Thm. 3). In a graph group, every defining relation is of the form $ab = ba$ for generators a and b.
- The compressed word problem for a finitely generated linear group belongs to the complexity class coRP (Thm. 5), which is the complementary class of randomized polynomial time. See Section 4.4 for the definition.

We end this paper with a few direct applications of the above results. Let us mention one of them concerning topology, see [31] for definitions: Crisp and Wiest [7] have shown shown that the fundamental group of any orientable surface (and of most non-orientable surfaces) embeds in a graph group. This gives a new proof that, for all closed surfaces, the word problem for the automophism group of the fundamental group can be solved in polynomial time.

A long version containing all proofs can be obtained from the authors.

2 Preliminaries

Let Σ be a finite alphabet. Let ε denote the empty word. We use $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$ to denote a disjoint copy of Σ. Let $\Sigma^{\pm 1} = \Sigma \cup \Sigma^{-1}$. For background in complexity theory see [24]. For languages K, L we write $K \leq_m^P L$ (resp. $K \leq_m^{\log} L$) if there exists a polynomial time (resp. logspace) many-one reduction from K to L. We write $K \leq_T^P L$ if there exists a polynomial time Turing reduction from K to L, which means that K can be solved in deterministic polynomial time on a Turing machine with oracle access to the language L. Let $\preceq \in \{\leq_m^P, \leq_m^{\log}, \leq_T^P\}$. In case $K \preceq L_1 \times \cdots \times L_n$ we write $K \preceq (L_1, \ldots, L_n)$. Clearly, if $L_1, \ldots, L_n$ belong to the class P (deterministic polynomial time) and $K \leq_T^P (L_1, \ldots, L_n)$, then K belongs to P as well.

2.1 Groups

For background in combinatorial group theory see [22,31]. Let G be a *finitely generated group* and let Σ be a finite *group generating set* for G. Hence, $\Sigma^{\pm 1}$ is a finite *monoid generating set* for G and there exists a canonical monoid homomorphism $h : (\Sigma^{\pm 1})^* \to G$, which maps a word $w \in (\Sigma^{\pm 1})^*$ to the group element represented by w. For $u, v \in (\Sigma^{\pm 1})^*$ we will also say that $u = v$ in G in case $h(u) = h(v)$.

The *word problem* for G with respect to Σ is the following decision problem:

INPUT: A word $w \in (\Sigma^{\pm 1})^*$.
QUESTION: $w = 1$ in G, i.e., $h(w) = 1$?

It is well known that if Γ is another finite generating set for G, then the word problem for G with respect to Σ is logspace many-one reducible to the word problem for G with respect to Γ. This justifies one to speak just of the word problem for the group G. The

word problem for G is also denoted by $\mathrm{WP}(G)$. The *free group* $F(\Sigma)$ generated by Σ can be defined as the quotient monoid

$$F(\Sigma) = (\Sigma^{\pm 1})^*/\{aa^{-1} = a^{-1}a = \varepsilon \mid a \in \Sigma\}.$$

As usual, the *free product* of two groups G_1 and G_2 is denoted by $G_1 * G_2$. The *automorphism group* of a group G is denoted by $\mathrm{Aut}(G)$. For the standard definition of *automatic groups*, see [12]. Every automatic group G is finitely presented and its word problem can be solved in time $O(n^2)$.

2.2 Trace Monoids and Graph Groups

In the following we introduce some notions from trace theory, see [8,10] for more details. This material will be only needed in Section 4.3. An *independence alphabet* is just a finite undirected graph (Σ, I) without loops. Hence, $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric relation. The *trace monoid* $\mathbb{M}(\Sigma, I)$ is defined as the quotient monoid

$$\mathbb{M}(\Sigma, I) = \Sigma^*/\{ab = ba \mid (a, b) \in I\}.$$

It is a cancellative monoid. Elements of $\mathbb{M}(\Sigma, I)$ are called *traces*. The trace represented by the word $s \in \Sigma^*$ is also denoted by $[s]_I$. The graph group $\mathbb{G}(\Sigma, I)$ is defined as the quotient group

$$\mathbb{G}(\Sigma, I) = F(\Sigma)/\{ab = ba \mid (a, b) \in I\}.$$

Note that $(a, b) \in I$ implies $a^{-1}b = ba^{-1}$ in $\mathbb{G}(\Sigma, I)$. Thus, the graph group $\mathbb{G}(\Sigma, I)$ can be also defined as the quotient monoid

$$\mathbb{G}(\Sigma, I) = \mathbb{M}(\Sigma^{\pm 1}, I)/\{[aa^{-1}]_I = [a^{-1}a]_I = [\varepsilon]_I \mid a \in \Sigma\}.$$

Here, we implicitly extend $I \subseteq \Sigma \times \Sigma$ to $I \subseteq \Sigma^{\pm 1} \times \Sigma^{\pm 1}$ by setting $(a^\alpha, b^\beta) \in I$ if and only if $(a, b) \in I$ for $a, b \in \Sigma$ and $\alpha, \beta \in \{1, -1\}$.

Free groups and free abelian groups arise as special cases of graph groups; note that $\mathbb{G}(\Sigma, \emptyset) = F(\Sigma)$ and $\mathbb{G}(\Sigma, (\Sigma \times \Sigma) \setminus \mathrm{id}_\Sigma) = \mathbb{Z}^{|\Sigma|}$. Graph groups were studied e.g. in [11]; they are also known as *free partially commutative groups* [9,32], *right-angled Artin groups* [4,7], and *semifree groups* [1].

2.3 Grammar Based Compression

In this section we introduce straight-line programs, which are used as a compressed representation of strings with reoccuring subpatterns. Following [26], a *straight-line program (SLP) over the alphabet* Γ is a context-free grammar $\mathbb{A} = (V, \Gamma, S, P)$, where V is the set of *nonterminals*, Γ is the set of *terminals*, $S \in V$ is the *initial nonterminal*, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of *productions*, such that (i) for every $X \in V$ there is exactly one $\alpha \in (V \cup \Gamma)^*$ with $(X, \alpha) \in P$ and (ii) there is no cycle in the relation $\{(X, Y) \in V \times V \mid \exists \alpha : (X, \alpha) \in P, Y \text{ occurs in } \alpha\}$. A production (X, α) is also written as $X \to \alpha$. The language generated by the SLP $\mathbb{A}$ contains exactly one word that is denoted by $\mathrm{eval}(\mathbb{A})$. More generally, every nonterminal $X \in V$ produces

exactly one word that is denoted by $\mathrm{eval}_{\mathbb{A}}(X)$. We omit the index $\mathbb{A}$ if the underlying SLP is clear from the context. The size of $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(X,\alpha)\in P} |\alpha|$. The length of $\mathrm{eval}(\mathbb{A})$ may be exponentially larger than $|\mathbb{A}|$; hence $\mathbb{A}$ may be seen as a compressed representation of $\mathrm{eval}(\mathbb{A})$. Every SLP can be transformed in polynomial time into an equivalent SLP that is in *Chomsky normal form* (as a context-free grammar). This means that all productions have the form $A \to BC$ or $A \to a$ for nonterminals A, B, and C and a terminal a.

In recent years, the complexity of many decision problems on strings, when the input is represented by SLPs, was investigated, see e.g. [13,19,21,23,25]. A seminal result of Plandowski [25] states that for given SLPs $\mathbb{A}$ and $\mathbb{B}$ it can be checked in polynomial time whether $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{B})$. The currently best known algorithm for this problem has a cubic running time [19].

The *compressed word problem* for the finitely generated group G with respect to the finite generating set Σ is the following problem:

INPUT: An SLP $\mathbb{A}$ over the terminal alphabet $\Sigma^{\pm 1}$.
QUESTION: Does $\mathrm{eval}(\mathbb{A}) = 1$ hold in G?

Here, the input size is $|\mathbb{A}|$. Also, it is easy to see that the complexity of the compressed word problem does not depend on the chosen generating set. This allows one to speak of the compressed word problem for the group G. The compressed word problem for G is also denoted by $\mathrm{CWP}(G)$. The following fact is trivial:

Proposition 1. *Assume that H is a finitely generated subgroup of the finitely generated group G. Then $\mathrm{CWP}(H) \leq_m^{\log} \mathrm{CWP}(G)$.*

3 Connections Between the Word Problem and the Compressed Word Problem

The three propositions from this section establish a link between the word problem and the compressed word problem. For their proofs, the following fact is crucial: Let Σ be a finite generating set for the group G and let $\varphi_1, \ldots, \varphi_n \in \mathrm{Aut}(G)$ be automorphisms of G which are taken from some fixed finite subset of $\mathrm{Aut}(G)$. Then, for every $a \in \Sigma^{\pm 1}$, we can construct an SLP $\mathbb{A}$ over the terminal alphabet $\Sigma^{\pm 1}$ such that (i) $\mathrm{eval}(\mathbb{A})$ is a word that represents the group element $\varphi_1 \cdots \varphi_n(a)$ and (ii) $|\mathbb{A}| \in O(n)$; see [28].

Proposition 2 (cf [28]). *Let G be a finitely generated group and let H be a finitely generated subgroup of $\mathrm{Aut}(G)$. Then $\mathrm{WP}(H) \leq_m^{\log} \mathrm{CWP}(G)$.*

Proposition 3. *Let K and Q be finitely generated groups and let $\varphi : Q \to \mathrm{Aut}(K)$ be a homomorphism. Then, for the semidirect product $K \rtimes_\varphi Q$ we have $\mathrm{WP}(K \rtimes_\varphi Q) \leq_m^{\log} (\mathrm{WP}(Q), \mathrm{CWP}(K))$.*

The semidirect product $G = K \rtimes_\varphi Q$ is an extension of K by Q, i.e., K is a normal subgroup of G with quotient $G/K \simeq Q$. A reasonable generalization of Prop. 3 would be $\mathrm{WP}(G) \leq_m^{\log} (\mathrm{WP}(G/K), \mathrm{CWP}(K))$. But this cannot be true: there exist finitely generated groups G, Q, K such that (i) $Q = G/K$, (ii) Q and K have a decidable

word problem, and (iii) G has an undecidable word problem [2]. On the other hand, if we require additionally, that Q is finitely presented (in fact, Q recursively presented suffices), then G must have a decidable word problem [6]. For the special case that the quotient $Q = G/K$ is automatic (and hence finitely presented), we can prove the following:

Proposition 4. *Let K be a finitely generated normal subgroup of G such that the quotient $Q = G/K$ is an automatic group. Then* $\mathrm{WP}(G) \leq_m^P \mathrm{CWP}(K)$.

4 Upper Bounds for Compressed Word Problems

4.1 Finite Extensions

Since every finite group is automatic, Prop. 4 applies to the case that the quotient Q is finite. In this situation, we even obtain a polynomial time reduction from the *compressed* word problem of G to the compressed word problem of K.

Theorem 1. *Assume that K is a finitely generated subgroup of the group G such that the index $[G : K]$ is finite. Then* $\mathrm{CWP}(G) \leq_m^P \mathrm{CWP}(K)$.

For the proof of Thm. 1 one proceeds in two steps. For a given SLP $\mathbb{A}$ over the generators of G one first checks whether $\mathrm{eval}(\mathbb{A})$ represents an element of the subgroup K. This is possible in polynomial time using the coset automaton (whose states are the cosets of K) and the fact that it can be checked in polynomial time whether a given finite automaton accepts $\mathrm{eval}(\mathbb{A})$ for a given SLP $\mathbb{A}$ [26]. Then, in a second step one transforms $\mathbb{A}$ in polynomial time into a new SLP $\mathbb{B}$ over generators for K such that $\mathrm{eval}(\mathbb{A})$ and $\mathrm{eval}(\mathbb{B})$ represent the same group element.

The reducibility relation $\leq_m^P$ in Thm. 1 cannot be replaced by the stronger relation $\leq_m^{\log}$ (unless P = L) because there exists a finite group G with a P-complete compressed word problem [3] (take $K = 1$ in Thm. 1).

4.2 Free Products

Our main result for free products is:

Theorem 2. *Assume that $G = G_1 * G_2$. Then* $\mathrm{CWP}(G) \leq_T^P (\mathrm{CWP}(G_1), \mathrm{CWP}(G_2))$.

Let Σ_i be a finite generating set for G_i ($i \in \{1, 2\}$), where $\Sigma_1 \cap \Sigma_2 = \emptyset$. In order to reduce $\mathrm{CWP}(G)$ to $\mathrm{CWP}(G_1)$ and $\mathrm{CWP}(G_2)$, we follow the strategy for free groups [21], where *composition systems* were used. Composition systems extend SLPs by allowing also productions of the form $A \to B[i : j]$ for nonterminals A and B and $i, j \in \mathbb{N}$. Then $\mathrm{eval}(A)$ is the substring of $\mathrm{eval}(B)$ from position i to j. Hagenah [15] has shown that a given composition system can be transformed in polynomial time into an equivalent SLP. For our proof, we use a special form of composition systems, so called *2-level composition systems*. Such a system is a tuple of the form $\mathbb{A} = (\mathbb{B}, (\mathbb{B}_C)_{C \in W})$, where $\mathbb{B}$ is a composition system, which generates a word over the alphabet W. Moreover, for each $C \in W$, $\mathbb{B}_C$ is an SLP, either over the terminal

alphabet $\Sigma_1^{\pm 1}$ or over the terminal alphabet $\Sigma_2^{\pm 1}$. Thus, $\mathbb{A}$ defines in a natural way a string $\mathrm{eval}(\mathbb{A}) \in (\Sigma_1^{\pm 1} \cup \Sigma_2^{\pm 1})^*$.

We transform a given input SLP $\mathbb{A}$ over the terminal alphabet $(\Sigma_1^{\pm 1} \cup \Sigma_2^{\pm 1})^*$ into a 2-level composition system $\mathbb{A}' = (\mathbb{B}, (\mathbb{B}_C)_{C \in W})$ having three additional properties:

(1) $\mathrm{eval}(\mathbb{A}) = \mathrm{eval}(\mathbb{A}')$ in the group $G_1 * G_2$.
(2) for every $C \in W$, $\mathrm{eval}(\mathbb{B}_C) \neq 1$ (either in G_1 or in G_2).
(3) for every nonterminal A of $\mathbb{B}$, if $C \in W$ and $D \in W$ are two consecutive symbols in $\mathrm{eval}_{\mathbb{B}}(A)$, then either $\mathrm{eval}(\mathbb{B}_C) \in (\Sigma_1^{\pm 1})^*$ and $\mathrm{eval}(\mathbb{B}_D) \in (\Sigma_2^{\pm 1})^*$ or $\mathrm{eval}(\mathbb{B}_C) \in (\Sigma_2^{\pm 1})^*$ and $\mathrm{eval}(\mathbb{B}_D) \in (\Sigma_1^{\pm 1})^*$.

Properties (2) and (3) ensures that $\mathrm{eval}(\mathbb{A}')$ is irreducible in the free product $G_1 * G_2$ and hence $\mathrm{eval}(\mathbb{A}) = 1$ in $G_1 * G_2$ if and only if $\mathrm{eval}(\mathbb{A}') = \varepsilon$. In order to enforce (2), we have to solve instances of $\mathrm{CWP}(G_1)$ and $\mathrm{CWP}(G_2)$. Enforcing (3) is the main difficulty. Here we follow the bottom-up procedure for free groups from [21] in order to determine maximal cancellation between strings which are concatenated on the right-hand side of some production of the SLP $\mathbb{A}$.

Again, the reducibility relation $\leq_T^P$ in Thm. 2 cannot be replaced by the stronger relation $\leq_m^{\log}$ (unless P = NC, where NC is Nick's class — the class of all problems that can be solved with polynomially many processors in polylogarithmic time) because the compressed word problem for $\mathbb{Z} * \mathbb{Z}$ is P-complete [21], whereas the compressed word problem for $\mathbb{Z}$ is easily seen to be in NC.

4.3 Graph Groups and Graph Products

The word problem for a graph group can be solved in linear time on a RAM [9,32]. In order to solve the compressed word problem for a graph group in polynomial time, we follow again the strategy for free groups [21]. For this, it is crucial that there exists a normal form mapping $\mathrm{NF} : \mathbb{M}(\Sigma^{\pm 1}, I) \to \mathbb{M}(\Sigma^{\pm 1}, I)$ on the trace monoid $\mathbb{M}(\Sigma^{\pm 1}, I)$ such that for all $t \in \mathbb{M}(\Sigma^{\pm 1}, I)$: (i) $t = \mathrm{NF}(t)$ in the graph group $\mathbb{G}(\Sigma, I)$ and (ii) the trace $\mathrm{NF}(t)$ cannot be factorized in $\mathbb{M}(\Sigma^{\pm 1}, I)$ as $u[aa^{-1}]_I v$ or $u[a^{-1}a]_I v$ for some $u, v \in \mathbb{M}(\Sigma^{\pm 1}, I)$ and $a \in \Sigma$ [9]. Then, for a given SLP $\mathbb{A}$ over the terminal alphabet $\Sigma^{\pm 1}$ we compute in polynomial time an SLP $\mathbb{B}$ over the terminal alphabet $\Sigma^{\pm 1}$ such that $[\mathrm{eval}(\mathbb{B})]_I = \mathrm{NF}([\mathrm{eval}(\mathbb{A})]_I)$. This calculation is again based on a bottom-up process similarly to [21], but determining the maximal amount of cancellation between composed strings of $\mathbb{A}$ becomes more involved in the presence of partial commutation. Since for every $t \in \mathbb{M}(\Sigma^{\pm 1}, I)$ we have $t = 1$ in $\mathbb{G}(\Sigma, I)$ if and only if $\mathrm{NF}(t) = [\varepsilon]_I$, we obtain:

Theorem 3. *Let (Σ, I) be a fixed independence alphabet. Then $\mathrm{CWP}(\mathbb{G}(\Sigma, I))$ belongs to P (deterministic polynomial time).*

Let us end this section with a generalization of both Thm. 2 and 3. A *graph product* is given by a triple $(\Sigma, I, (G_v)_{v \in \Sigma})$, where (Σ, I) is an independence alphabet and G_v is a group, which is associated with the node $v \in \Sigma$. W.l.o.g. assume that $\Sigma = \{1, \ldots, n\}$. The group $\mathbb{G}(\Sigma, I, (G_v)_{v \in \Sigma})$ defined by this triple is the quotient

$$\mathbb{G}(\Sigma, I, (G_v)_{v \in \Sigma}) = (G_1 * G_2 * \cdots * G_n)/\{xy = yx \mid x \in G_u, y \in G_v, (u, v) \in I\},$$

i.e., we take the free product $(G_1 * G_2 * \cdots * G_n)$, but let elements from adjacent groups commute. Note that $\mathbb{G}(\Sigma, I, (G_v)_{v\in\Sigma})$ is the graph group $\mathbb{G}(\Sigma, I)$ in case every G_v is isomorphic to $\mathbb{Z}$. Moreover, free products and direct products appear as special cases of the graph product construction. Graph products were first studied by Green [14]. By combining ideas from the proof of Thm. 2 and Thm. 3, one can show:

Theorem 4. *Assume that G is a graph product of finitely generated groups $G_1, \ldots, G_n$. Then* $\mathrm{CWP}(G) \leq_T^P (\mathrm{CWP}(G_1), \ldots, \mathrm{CWP}(G_n))$.

4.4 Linear Groups

Recall that a language L belongs to the complexity class RP (randomized polynomial time) if there exists a randomized polynomial time algorithm[2] A such that: (i) if $x \notin L$ then $\mathrm{Prob}[A \text{ accepts } x] = 0$ and (ii) if $x \in L$ then $\mathrm{Prob}[A \text{ accepts } x] \geq 1/2$. The choice of the failure probability $1/2$ in case $x \in L$ is arbitrary: By repeating the algorithm c times (where c is some constant), we can reduce the failure probability to $(1/2)^c$ and still have a randomized polynomial time algorithm. A language L belongs to the class coRP, if the complement of L belongs to RP. This means that there exists a randomized polynomial time algorithm A such that: (i) if $x \notin L$ then $\mathrm{Prob}[A \text{ accepts } x] \leq 1/2$ and (ii) if $x \in L$ then $\mathrm{Prob}[A \text{ accepts } x] = 1$.

Using results from [20, 30], the compressed word problem for a finitely generated linear group can be reduced to the problem whether a circuit over a polynomial ring $R[x_1, \ldots, x_n]$ (where R is either $\mathbb{Z}$ or the finite field $\mathbb{F}_p$) evaluates to the zero polynomial. This problem belongs to coRP by [16]. Hence, we obtain:

Theorem 5. *For a finitely generated linear group G,* $\mathrm{CWP}(G)$ *belongs to coRP.*

5 Applications

In this section, we present some immediate corollaries to the results from Section 3 and 4. We concentrate on automorphism groups. Since the automorphism group of a graph group is finitely generated [18, 29], Prop. 2 and Thm. 4 imply:

Corollary 1. *For a graph group G,* $\mathrm{WP}(\mathrm{Aut}(G))$ *belongs to P.*

Let S_g be the closed orientable surface of genus g. For example, S_0 is the two-sphere. Let $\pi_1(S_g)$ denote the fundamental group (see [31] for definitions). Crisp and Wiest [7] have shown that for every $g \geq 0$, $\pi_1(S_g)$ can be embedded in a graph group. Hence, by Prop. 1 and Thm. 4, the compressed word problems for these groups can be solved in polynomial time. (This gives a new proof of a result of [28].) Since S_g is a double cover of N_{g+1}, the non-orientable surface, [31, p. 87], it follows that $\pi_1(S_g)$ is an index-2 subgroup of $\pi_1(N_{g+1})$ [31, p. 162]. With Thm. 1 and Prop. 2 we obtain:

[2] A randomized algorithm A may flip coins. Hence, it accepts a given input only with some probability. If there exists a polynomial $p(n)$ such that for every input of length n and every possible outcome of the coin flips, A runs in time at most $p(n)$, then A is a randomized polynomial time algorithm.

Corollary 2. *Let G be the fundamental group of a closed (orientable or nonorientable) surface. Then* $\mathrm{CWP}(G)$ *and* $\mathrm{WP}(\mathrm{Aut}(G))$ *belong to P.*

Automorphism groups of fundamental groups of surfaces play an important role in algebraic topology; they are closely related to mapping class groups.

6 Open Problems

We finish this paper with some open problems concerning compressed word problems:

1. Is the compressed word problem for a hyperbolic group solvable in polynomial time? For torsion-free hyperbolic groups one might try to attack this question using the canonical representatives of Rips and Sela [27].
2. What about the compressed word problem for automatic groups? Is it possible to prove a non-trivial lower bound (e.g. NP-hardness or coNP-hardness) for the compressed word problem of some specific automatic group?
3. Is the uniform compressed word problem for graph groups solvable in polynomial time? In this problem, the independence alphabet (Σ, I), which defines the underlying graph group, is also part of the input. Note that in Thm. 3 the independence alphabet (Σ, I) is not part of the input.
4. Can Thm. 2 be generalized from free products to (suitably restricted) amalgamated free products and HNN-extensions?
5. Is it possible to relax the restriction to an automatic quotient group Q in Prop. 4?
6. The *compressed generalized word problem* (CGWP) for a finitely generated group G asks, whether for SLPs $\mathbb{A}, \mathbb{B}_1, \ldots, \mathbb{B}_n$ (over generators for G), the word $\mathrm{eval}(\mathbb{A})$ represents a group element from the subgroup $\langle \mathrm{eval}(\mathbb{B}_1), \ldots, \mathrm{eval}(\mathbb{B}_n) \rangle \leq G$. What is the complexity of $\mathrm{CGWP}(F(\{a, b\}))$? We only know an exponential time algorithm for this problem.

References

1. Baudisch, A.: Subgroups of semifree groups. Acta Math. Acad. Sci. Hungar. 38, 19–28 (1981)
2. Baumslag, G., Cannonito, F.B., Miller III, C.F.: Infinitely generated subgroups of finitely presented groups. Math. Z. 153(2), 117–134 (1977)
3. Beaudry, M., McKenzie, P., Péladeau, P., Thérien, D.: Finite monoids: From word to circuit evaluation. SIAM J. Comput. 26(1), 138–152 (1997)
4. Brady, N., Meier, J.: Connectivity at infinity for right angled Artin groups. Trans. Amer. Math. Soc. 353, 117–132 (2001)
5. Cannonito, F.B., Gatterdam, R.W.: The word problem and power problem in 1-relator groups are primitive recursive. Pacific J. Math. 61(2), 351–359 (1975)
6. Cockcroft, W.H.: The word problem in a group extension. Quart. J. Math. Oxford Ser. 2(2), 123–134 (1951)
7. Crisp, J., Wiest, B.: Embeddings of graph braid and surface groups in right-angled Artin groups and braid groups. Algebr. Geom. Topol. 4, 439–472 (2004)
8. Diekert, V.: Combinatorics on Traces. LNCS, vol. 454. Springer, Heidelberg (1990)

9. Diekert, V.: Word problems over traces which are solvable in linear time. Theoret. Comput. Sci. 74, 3–18 (1990)
10. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
11. Droms, C.: Graph groups, coherence and three-manifolds. J. Algebra 106(2), 484–489 (1985)
12. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: Word processing in groups. Jones and Bartlett, Boston (1992)
13. Gasieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding (extended abstract). In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 392–403. Springer, Heidelberg (1996)
14. Green, E.R.: Graph Products of Groups. PhD thesis, The University of Leeds (1990)
15. Hagenah, C.: Gleichungen mit regulären Randbedingungen über freien Gruppen. PhD thesis, University of Stuttgart, Institut für Informatik (2000)
16. Ibarra, O.H., Moran, S.: Probabilistic algorithms for deciding equivalence of straight-line programs. J. Assoc. Comput. Mach. 30(1), 217–228 (1983)
17. Kapovich, I., Myasnikov, A., Schupp, P., Shpilrain, V.: Generic-case complexity, decision problems in group theory, and random walks. J. Algebra 264(2), 665–694 (2003)
18. Laurence, M.R.: A generating set for the automorphism group of a graph group. J. London Math. Soc. (2), 52(2), 318–334 (1995)
19. Lifshits, Y.: Processing compressed texts: a tractability border. In: Proc. CPM 2007. Springer 2007 (to appear)
20. Lipton, R.J., Zalcstein, Y.: Word problems solvable in logspace. J. Assoc. Comput. Mach. 24(3), 522–526 (1977)
21. Lohrey, M.: Word problems and membership problems on compressed words. SIAM J. Comput. 35(5), 1210–1240 (2006)
22. Lyndon, R.C., Schupp, P.E.: Combinatorial Group Theory. Springer, Heidelberg (1977)
23. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: Hein, J., Apostolico, A. (eds.) Combinatorial Pattern Matching. LNCS, vol. 1264, pp. 1–11. Springer, Heidelberg (1997)
24. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, London, UK (1994)
25. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
26. Plandowski, W., Rytter, W.: Complexity of language recognition problems for compressed words. In: Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 262–272. Springer, Heidelberg (1999)
27. Rips, E., Sela, Z.: Canonical representatives and equations in hyperbolic groups. Invent. Math. 120, 489–512 (1995)
28. Schleimer, S.: Polynomial-time word problems. In: Commentarii Mathematici Helvetici (to appear)
29. Servatius, H.: Automorphisms of graph groups. J. Algebra 126(1), 34–60 (1989)
30. Simon, H.-U.: Word problems for groups and contextfree recognition. In: Proc. FCT'79, pp. 417–422. Akademie-Verlag (1979)
31. Stillwell, J.: Classical Topology and Combinatorial Group Theory. Springer, Heidelberg (1995)
32. Wrathall, C.: The word problem for free partially commutative groups. J. Symbolic Comput. 6(1), 99–104 (1988)

Constructing a Secret Binary Partition of a Digital Image Robust to a Loss of Synchronization

Alexei Lysenko

Ural State University, Yekaterinburg, Russia
lysenko@e1.ru

Abstract. Digital watermarking, named after paper watermarking, was proposed to protect copyrights of perceptual content owners. A *Watermark*, or an author's signature, is hidden in a signal by small modifications of the signal. This signature, particularly, can be used as the evidence of authorship/ownership in a court.

Performing *attacks*, a Malefactor is finding ways to destroy the embedded watermark so that it will not be found by a detection algorithm.

Frequently a watermark becomes undetectable after an attack due to the *loss of synchronization problem*. One of the ways to handle this problem is to build a reference system which would be robust to a certain range of attacks.

A novel algorithm of building such a system has been proposed by Delannay in [1].

Starting from ideas of Delannay we propose new approach to constructing a secret binary partition of an image.

Keywords: digital image watermarking, secret image partition, pseudorandom partition, loss of synchronization problem.

1 Introduction

Digital watermarking is a branch of modern digital steganography. *Digital steganography* itself is a study of information hiding techniques when additional information has to be hidden in a still image, audio or video content.

Digital watermarking, named after paper watermarking, was proposed to protect copyrights of perceptual content owners. A *Watermark*, or author's signature, is hidden in a signal by small modifications of the signal. This signature, particularly, can be used as the evidence of authorship/ownership in a court.

A Malefactor will be finding ways to destroy the embedded watermark so that it will not be found by a detection algorithm. Modifying the signal (performing *attacks*), the malefactor is also restricted in amount of modifications because the signal must not be badly degraded. In the case of digital image, typical attacks are loss-compression, cropping and geometrical transformations. More complicated operations, such as local geometrical distortions, special filtering, or noise addition, can be carried out to destroy the embedded watermark.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 259–268, 2007.

This paper concerns the robustness of the digital image watermarking. Watermark degradation may be caused both by intentional and non-intentional actions. Sometimes a watermark becomes undetectable after an attack due to the *loss of synchronization problem*; thus constructing a robust watermarking system, one must define the way to handle this problem.

Let us decribe the problem briefly.

A common way to embed information is to add a weak watermark signal to a content. For example we can represent a two-dimensional picture as a one-dimensional array of numbers and then add information modifying least significant bits of these numbers. Improving this algorithm (e.g. using error-correction codes) we can achieve robustness to a filtering, noise addition and loss compression. But if one crops the image, rotates it by a small angle or resizes, we are not able to restore information because we cannot find the original one-dimensional array in the modified picture. So the watermark is still present in the image, but the detector is not able to find it due to the *loss of synchronization*.

Different methods have been proposed to handle the loss of synchronization problem. First, for the embedding one can choose a domain that is invariant to the range of attacks under consideration. Fourier magnitude, log-polar mapping, log-log mapping and Fourier-Mellin domain can be used to achieve robustness with regard to affine transforms and a cropping [4,5].

The pilot signal can be embedded to determine and revert transformations [7].

Finally, we can exploit an image content to build a reference system which will be transformed along with the image. This approach is very promising because such reference systems are hard to modify with small modifications of an image. In [6] extraction of image corner points and triangulation are used to embed a robust watermark.

Different methods of synchronizing are reviewed and studied in [1] by Delannay.

In [1], Chapter 4.2, Delannay proposes construction of a reference system of an image. This system is robust to geometrical distortions. Delannay also suggests an image-dependent secret binary partition algorithm, based on this system.

Secret binary image partition is a mean to break an image into two parts in a way that seems random and can be reconstructed only with the knowlege of a secret key. For instance an image-independent secret partition can be obtained mapping a one-dimensional pseudorandom binary sequence on the two-dimensional array of image pixels.

Secret binary image partition can be used in the digital watermarking. For example, in a Patchwork algorithm [2] a watermark is embedded by increasing luminosity of one part of an image and decreasing lunimosity of another part by a small amount. More comlicated algorithms use secret partition to hide a watermark signal [3]. So the construction of a *robust* secret partition is a challenge adjacent to the construction of a robust watermark.

Briefly, Delannay's approach can be described as following. First for each pixel of an image a scale and rotation invariant characteristic (so called *local radius*)

is calculated. The calculation is based on the luminosity behavior of the image in the neighbourhood of the considered pixel. Then two circles with the center in the considered pixel are selected. Their radii are defined relative to the calculated *local radius* and unknown to attacker. Finally the membership of the considered pixel in the partition is determined by the angle between positions of maximum and minimum luminosities of the image along selected circles.

Using behavior of luminosity on circular objects centered in the considered pixel to determine pixel's membership is a significal benefit of Delannay's approach. It allows to achieve robustness towards image rotation and cropping. Using radii relative to scale-invariant local radius allows also handle the scaling attack.

We see several drawbacks of proposed algorithm. First drawback is insufficient secrecy: The key of partition consists of two radii, which are varying in a relative narrow interval. Second drawback, as it seems, consists in the procedure of defining membership of pixel. This procedure uses indiviual pixel luminosities (minimum and maximum luminosity along a circle), so it's result can be significantly affected by the noise.

The work was initially started as an attempt to eliminate mentioned drawbacks of Delannay's algorithm, but finally a new flexible approach with a good theoretical robustness was obtained.

We started from idea to repace the procedure of calculation of the angle between minimum and maximum of luminosity along the circle with the procedure of calculating difference of mean luminosities along two circles.

Our approach is described in following sections.

2 Elemental Convolution

To construct a robust secret binary partition of an image we shall try to "complicate" the image using operations robust to a certain range of attacks. Finally the partition can be obtained thresholding the complicated picture.

The "complicating" is performed as a sequence of elemental transformations of an image. Parameters of transformations on each step form a key of the secret partition.

Let us consider an elemental transformation of an image as following. The luminosity value of each pixel of the image is replaced with the difference between mean luminosities of two circles of different radii r_1 and r_2 with the center in the considered point. Radii of circles are the parameters of transformation.

Such a choice of an elemental transformation is caused by the robustness demands. Indeed, using an integral characteristic of an image (the mean luminosity) allows eliminating influence of attacks in the high frequency domain (e.g. loss compression or noise addition). Using a circular form gives an independence from the rotation of an image. Selecting radii small enough provides robustness to the local distortions of an image.

Considering the choice of r_1 and r_2 two approaches are available.

We can use fixed radii r_1 and r_2 for all pixels of an image. Let's denote it as the *global* radii selection. Alternatively, we can choose individual r_1 and r_2 for

each pixel basing on some robust geometrical local image characteristic near this pixel. We shell refer it as the *local* radii selection approach.

Local approach seems preferable because radii will be correctly recovered in the cropped or resized image. Nevertheless, two issues remain opened; first, we must construct a robust local geometrical characteristic of an image. For example we can try to use the *reference radius* characteristic described in [1] by Delannay. Also, performing computations of the mean luminosity of a circle for each pixel separately is very slow.

Global approach is attractive because of an availability of its quick realization. Indeed, if radii r_1 and r_2 are the same for each pixel in a step, required difference between the mean circle luminosities for each pixel can be obtained simultaneously after a single convolution of an image with the filter of the special kind. The filter consists of two uniform circles of radii r_1 and r_2 such that the integral luminosity of the first circle equals 1 and the integral luminosity of the second circle equals -1.

Although global approach potentially is less robust towards the cropping and resizing, several cases can be handled properly. Particular, a case when an image is only cropped (because cropping does not affect r_1 and r_2), or only resized (then we can select r_1 and r_2 relative to the image dimensions). Also we can think of constructing a geometrical characteristic of an "important" part of an image and defining radii relative to it. The radii can be reconstructed if an "important part" will be found in a cropped image.

We choose *global* approach to examine viability of the complicating scheme due to its computational efficiency.

As it was mentioned, an elemental transformation of an image in the global case can be performed as a convolution of an image and the kernel, defined by the radii r_1 and r_2. Let us refer such convolution as *elemental convolution* with parameters r_1, r_2.

After an elemental convolution an image with the luminosity from $[0..\ N]$ is transformed into an image with the luminosity from $[-N\ ..\ N]$.

In the resulting image pixels with a big absolute value of luminosity are distributed near the border pixels of the original image. Uniform areas of the original picture cause small absolute values after an elemental convolution.

Thresholding the result we can build the following binary image partition. Pixels are included in the first part if their luminosity value after a convolution is greater than zero; otherwise they are included in the second part.

Figure 1*a* shows the result of an elemental convolution of the Lena image with the filter, consisted of two circles of width 2 pixels and radii 8 and 12 pixels respectively. On the figure dark pixels correspond to negative values of the luminosity and bright pixels correspond to positive values. Obviously, binary image partition prescribed by this elemental convolution is far from the pseudorandom. Borders of the original image are clearly discovered in the "complicated" image. Nevertheless quantity of the border pixels after an elemental convolution is increased and we can complicate the image so forth applying next elemental convolution.

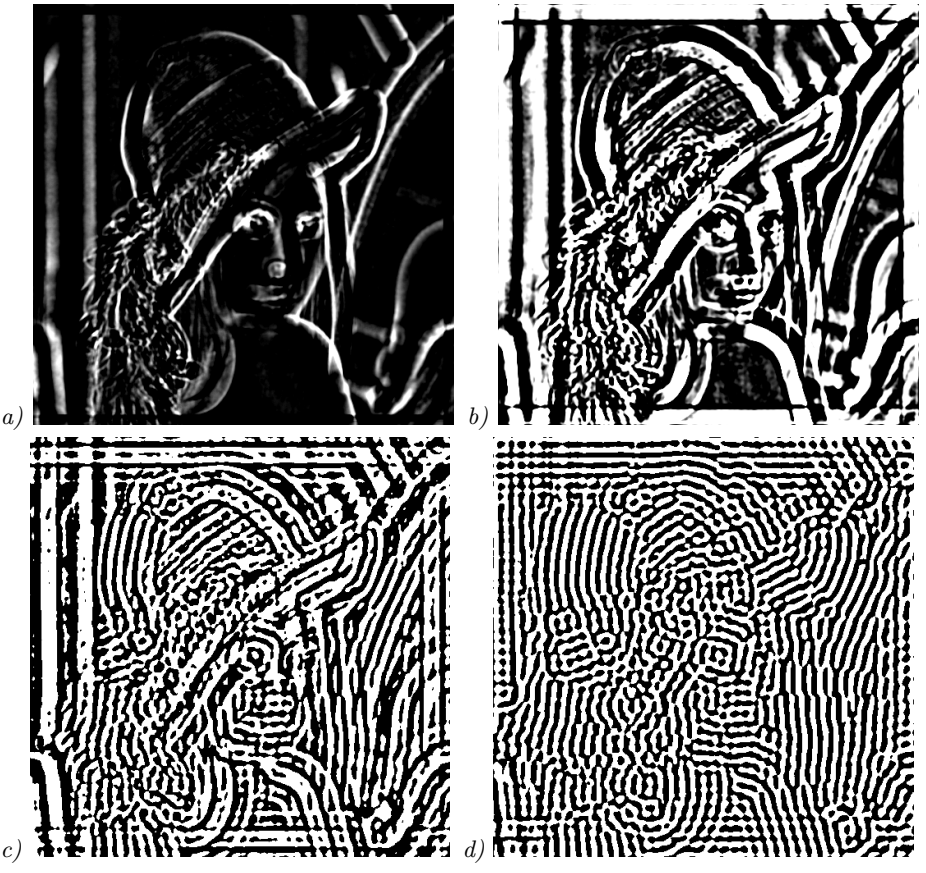


Fig. 1. Resulting image after 1(a), 2 (b), 5 (c) and 10 (d) elemental convolutions

Next elemental convolution has its own parameters; they are also included in the key of the partition. After the second convolution quantity of border pixels increases and the picture looks more random. We can continue this process with a third convolution and so forth. Finally a binary partition is constructed thresholding the resulting image with the zero threshold. Partition key is defined as the set of elemental convolution parameters on each convolution step.

Note that algorithm cam be optimized if replace a sequence of convolutions by one convolution with another filter; this filter itself is a convolution of elemental convolution filters.

Figure 1 shows resulting images after 1, 2, 5 and 10 elemental convolutions with the same parameters ($r_1 = 8, r_2 = 12$). A link between the original image and the image after ten elemental convolutions is not so obvious. Nevertheless, experiments showed that such partition do not meet demands of secrecy of a partition. An algorithm of secret image partition which overcomes this problem is described in the following section.

3 Secret Partition Algorithm

Preliminary tests of the algorithm described above showed a good robustness towards common attacks, but essential imperfection was also found. The weakness is related to secrecy of the partition. In some cases two partitions of the same image built with completely different keys were coincident on over than 80 percents of pixels. This effect is frequently observed on almost uniform image areas with a small number of details.

It is clear that good secret partition must be independent from other partitions, providing coincidence about 50 percents. This issue is especially important when robustness of the partition is required. Indeed, after attack partition will not be recovered perfectly, so the detection must base on then fact that coincidence essentially differs from 50 percents.

Let us figure out the reason of low secrecy of the described partition. After the first elemental convolution areas with big absolute values of the luminosity appear as lines around the border pixels of an image. On the next step bright borders appears around these borders and so on (see 1).

This process resembles a wave propagation and interference from the border pixels of an image. Parameters of a secret key affect the length of a wave and do not affect the wave front itself. This is the reason why we can see lines parallel to image borders even after a big number of elemental convolutions.

This metaphorical reasoning helps understand that we must supplement the algorithm with operations different from convolutions to obtain secrecy of the partition.

To decrease influence of image borders we try to combine intervening partitions after a certain number of elemental convolutions.

A scheme of the modified algorithm is shown on the figure 2.

Here the key of partition is a two-dimensional array K of $m \times n$ elemental convolution parameters.

The input is an image I_0. A sequence of elemental convolutions with parameters $K(1,1)$, $K(1,2)$, ... , $K(1,n)$ is applied to I_0. Result after n convolutions is stored as I_1. Next sequence of convolutions with parameters $K(2,1)$, $K(2,2)$, ... , $K(2,n)$ is applied to I_1, then result is combined with I_1 using operation $\times$ and stored as I_2 (the choice of operation $\times$ will be discussed below). I_3 is obtained as a combination of next n convolutions with I_2, and so on until m-th iteration.

Finally, binary partition is built thresholding I_m. The first part includes pixels (x, y) which has $I_m(x, y) > 0$, and the second part includes all remaining pixels.

Initially operation $\times$ was defined as comparison of two partitions defined by images I_k and I_l

$I_k \times I_l(x, y) = 1$, if $sign(I_k(x, y)) = sign(I_l(x, y))$, else $I_k \times I_l(x, y) = -1$.

That is equivalent to:

$I_k \times I_l(x, y) = 1$, if $I_k(x, y) \cdot I_l(x, y) \geq 0$, else $I_k \times I_l(x, y) = -1$.

Practically better robustness was achieved on the continuous operation of element-wise product of images I_k and I_l; this expands previous definition.

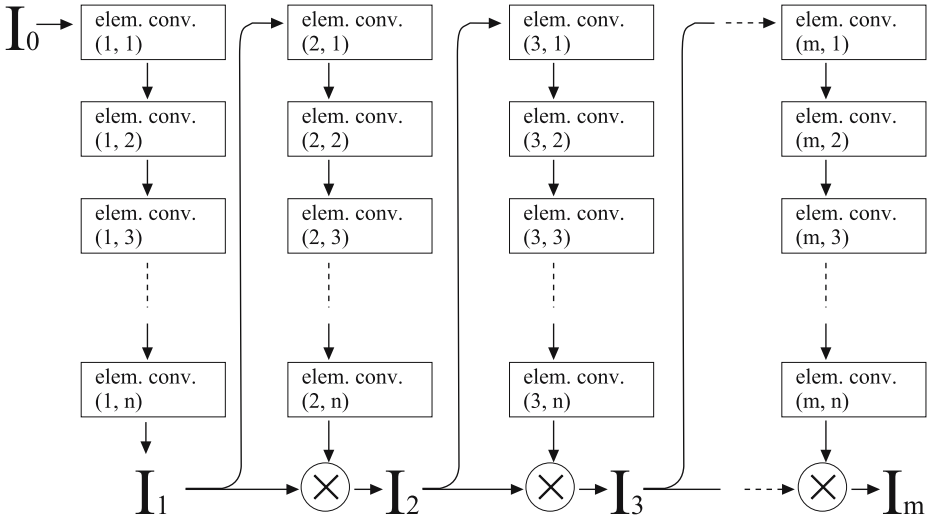


Fig. 2. Construction of a secret binary image partition

$$I_k \times I_l(x, y) = I_k(x, y) \cdot I_l(x, y).$$

An example of building a secret partition is shown on the figure 3. The resulting partition I_4 looks much more random than any of partitions on the figure 1. Experiments shown that varying parameters of an algorithm balance between secrecy and robustness can be shifted.

To achieve a better robustness towards the noise we used circles of width 2, 3, and 4 pixels in the convolution filter. We shell refer this parameter as *cirwidth*.

Experiments showed that an optimal combination of fuzziness and robustness of the partition is achieved when the distance between the circles in the filter is minimal, i.e. $r_2 = r_1 + cirwidth$. Therefore parameters of an elemental convolution on each step can be represented by the number r_1. We limit it by r_{min} below and vary it with a step *cirwidth*, i.e.

$$r_1 = r_{min} + c \cdot cirwidth, \text{where } c \in [0, V-1] \text{– is a part of the key.}$$

Thus the global parameters of an algorithm are

- n – the count of sequential convolutions between element-wise multiplications.
- m – the count of element-wise multiplications
- *cirwidth* – the width of circles. Typical values are 2, 3, 4 pixels.
- r_{min} – the minimal radius of circles. Typical values are 4, 6 pixels.
- V – the number of variants of choice r_1 on each elemental convolution step. Typical values are 6 and 8.

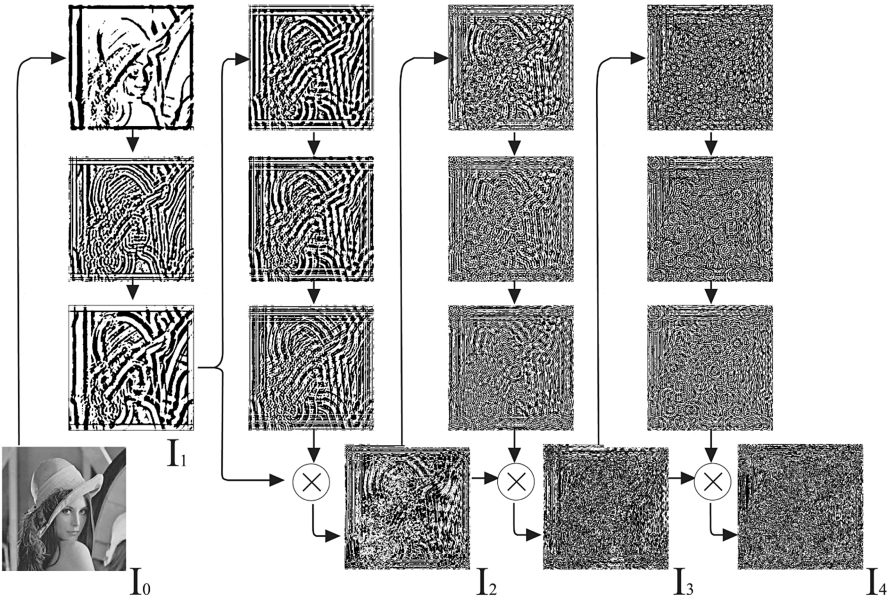


Fig. 3. Example of building a partition with random key of size 4×3

4 Experimental Results and Directions for Further Work

Evaluating properties of the proposed partition we used a Patchwork watermarking scheme. Image was divided into two parts according the given secret partition. Then luminosity of the first part was increased and luminosity of the second part was decreased by the same value δ (we choose $\delta = 1.25\%$ of the maximum image luminosity). The watermark can be detected breaking an image in two parts according the given secret partition and comparing mean luminosities of these parts. If difference is close to $2 \cdot \delta$ than watermark is detected. We evaluate the *strenth* of a watermark as the ratio of calculated difference to $2 \cdot \delta$.

Experiments were carried out in the MATLAB environment in the following way. For each selected combination of global parameters n, m, $cirwidth$, V, r_{min}, ten random keys where generated. For each key were calculated:

- strength of the watermark in an unmarked image (false positivity).
- strength of the watermark in a marked image
- strength of the watermark after the noise attack (Caussian noise with the zero mean and the variance of 0.005 was used)
- strength of the watermark after the rotation by 15 degrees attack

For each set of global parameters following values were calculated:

- mean and max strength of the watermark in an unmarked image
- mean and max coincidence of partitions built with different keys
- minimal, mean, and maximal strength of the watermark in a marked image

– minimal, mean, and maximal of the watermark after noise attack
– minimal, mean, and maximal of the watermark after rotation attack

Experiments demonstrated that varying length of the key via global parameters balance between robustness and secrecy can be shifted in a wide range.

A good level of the robustness and secrecy was achieved concerning non-geometrical attacks (noise and compression). Concerning a rotation attack most secret watermarks turn out to be insufficiently robust.

Exact demands to the robustness and secrecy of the partition are formed with a specific application. Some combinations of global parameters which showed a good results are listed in the table 1. However it is possible that another more complicated watermarking scheme exploiting described secret partition will demonstrate a better robustness. For instance we could add to the image a two-dimensional harmonic modulated by a secret partition; the detection then will be based on a presence of the frequency in the spectrum of the image multiplicated by the secret partition.

Table 1. Combinations of global parameters

Key length	n	m	V	r_{min}	False alarm	Strength	Strength (noise)	Strength (rotate)
45 bit	3	7	8	6	0.06	0.88	0.26	0.26
45 bit	3	7	8	4	0.03	0.67	0.16	0.25
39 bit	3	6	8	6	0.07	0.83	0.31	0.26
35 bit	2	7	8	6	0.1	0.765	0.29	0.30

The work is still in progress. Issues to investigate are:

– The choice of parameters. Although about 40 series of experiments were carried out to reveal an optimal set of global parameters, expanding image base, number of experiments in series, and range of attacks is desireable. *Stirmark* benchmarking is also advisable.
– The choice of the key. Now we use an array of size $m \times n$ of random generated integers from $[0, V-1]$. Nevertheless robustness of two random keys can differ drastically. Describing a way of optimal key generation can improve robustness of an algorithm.
– Revision of current algorithm. For example it well may be that optimal number of convolution steps between multiplications is not constant and depends on an iteration number. Smoothing of an image before building a partition can be useful.
– Defining a way to handle a scaling attack. Finding a robust geometrical characteristic of an image which allows to resize an image to an original scale before building a partition.
– Examining a viability of a *local* approach. Constructing of a geometrically robust local characteristic of an image and finding an algorithm for quick realization.

5 Conclusion

We have introduced an approach to the construction of the robust secret binary partition of an image. Proposed scheme is based on the sequential convolutions with secret parameters and allows an efficient trade of between robustness and secrecy. Good robustness towards non-geometrical attacks was achieved. Partition with a lower level of secrecy also demonstrated robustness towards a rotation attack. There might be ways to increase robustness of the proposed scheme regarding regarding the geometrical attacks. We believe the described approach can be useful in an image watermarking applications.

References

1. Delannay, D.: Digital Watermarking Algorithms Robust Against Loss of Synchronization. Dissertation for degree of Ph.D, Universite catholique de Louvain, Belgium (2004)
2. Bender, W., Gruhl, D., Morimoto, N., Lu, A.: Techniques for data hiding. IBM Syst. J. 35(34), 313–316 (1996)
3. Paiz, F.: Tartan Threads: A Method for the Real-time Digital Recognition of Secure Documents in Ink Jet Printers. MEng Thesis. MIT, Cambridge, MA (1999)
4. Ruanaidh, J.J.K.O., Pun, T.: Rotation, scale and translation invariant spread spectrum digital image watermarking. Signal Processing 66, 303–318 (1998)
5. Lin, C., Wu, M., Bloom, J.A., Cox, I., Miller, M., Lui, Y.: Rotation, scale, and translation resilient public watermarking for images. In: Proceedings of the SPIE, Security and Watermarking of Multimedia Contents, vol. 3971, pp. 90–98 (2000)
6. Bas, P., Chassery, J.-M., Macq, B.: Geometrically invariant watermarking using feature points. IEEE Transactions on Image Processing 11, 1014–1028 (2002)
7. Alvarez-Rodriguez, M., Perez-Gonzalez, F.: Analysis of pilotbased synchronization algorithms for watermarking of still images. Signal Processing - Image Communication, pp. 611–633 (2002)

On the Complexity of Matrix Rank and Rigidity

Meena Mahajan and Jayalal Sarma M.N.

The Institute of Mathematical Sciences, Chennai 600 113, India
{meena,jayalal}@imsc.res.in

Abstract. We revisit a well studied linear algebraic problem, computing the rank and determinant of matrices, in order to obtain completeness results for small complexity classes. In particular, we prove that computing the rank of a class of diagonally dominant matrices is complete for L. We show that computing the permanent and determinant of tridiagonal matrices over $\mathbb{Z}$ is in GapNC^1 and is hard for NC^1. We also initiate the study of computing the rigidity of a matrix: the number of entries that needs to be changed in order to bring the rank of a matrix below a given value. It is NP-hard over $\mathbb{F}_2$ and we prove that some restricted versions characterize small complexity classes. We also look at a variant of rigidity where there is a bound on the amount of change allowed. Using ideas from the linear interval equations literature, we show that this problem is NP-hard over $\mathbb{Q}$ and that a certain restricted version is NP-complete. Restricting the problem further, we obtain variations which can be computed in PL and are hard for $\mathsf{C}_=\mathsf{L}$.

1 Introduction

A series of seminal papers by a variety of people including Valiant, Mulmuley, Toda, Vinay, Grigoriev, Cook, and McKenzie, set the stage for studying the complexity of computing matrix properties (in particular, determinant and rank) in terms of logspace computation and poly-size polylog depth circuits. This area has been active for many years, and an NC upperbound is known for many related problems in linear algebra; see for instance [1]. In particular, the complexity of computing the rank of a given matrix over $\mathbb{Q}$ has been well studied. For general matrices, checking if the rank is at most r is $\mathsf{C}_=\mathsf{L}$-complete [3].

Complete problems for complexity classes are always promising, since they provide a set of possible techniques that are associated with the problem to attack various questions regarding the complexity class. Such results can be expected to flourish when the complete problem has well-developed tools associated with it. With this motivation, we look at special classes of the matrix rank problem and try to characterize small complexity classes. We consider restrictions which are combinations of non-negativity, 0-1 entries, symmetry, diagonal dominance, tridiagonal support, and we consider the complexities of three problems: computing the rank, computing the determinant and testing singularity. These, though intimately related, can have differing complexities, as Table 1 shows.

However, the corresponding optimization search problems can be considerably harder. Consider the following existential search question: Given a matrix M over

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 269–280, 2007.

Table 1. RANK BOUND, SINGULAR, and DETERMINANT for special matrices

Matrix type (over $\mathbb{Q}$)	RANK BOUND	SINGULAR	DETERMINANT
general (even 0-1)	$\mathsf{C_=L}$-complete [3]	$\mathsf{C_=L}$-complete [3]	GapL complete [9,22,19,21]
symmetric non-neg.	$\mathsf{C_=L}$-complete [3]	$\mathsf{C_=L}$-complete [3]	?
symmetric non-neg. diag. dominant (d.d.)	L complete (Theorem 1)	L complete (Theorem 1)	?
symmetric d.d.	L hard even when det $\in \{0,1\}$ (Theorem 2)		?
tridiagonal	?	$\mathsf{C_=NC^1}$ (Theorem 3)	$\mathsf{GapNC^1}$ (Thm. 3)
tridiagonal non-neg.	non-negative perm equivalent to planar #BWBP (Theorem 3)		

a field $\mathbb{K}$, a target rank r and a bound k, decide whether the rank of M can be brought down to below r by changing at most k entries of M. Intuitively, one would expect such a question to be in $\exists \cdot \mathsf{NC}$: guess k locations where M is to be changed, guess the new entries to be inserted there, and compute the rank in NC ([15]). However, this intuition, while correct for finite fields (this case was recently shown to be NP-complete [10]), does not directly translate to a proof for $\mathbb{Q}$ and $\mathbb{Z}$[1] since the required new entries may not have representations polynomially-bounded in the input size. In fact, the best upper bound we can see in the general case is recursive enumerability. In this paper, we explore the computational complexity of several variants of this problem.

The above question is a computational version of rigidity of a matrix, which is the smallest value of k for which the answer to the above question is yes. The notion of rigidity was introduced by Valiant [20] and independently proposed by Grigoriev [11]. The main motivation for studying rigidity is that good lower bounds on rigidity give important complexity-theoretic results in other computational models, like linear algebraic circuits and communication complexity. Though the question we address is in fact a computational version of rigidity, it has no direct implications for these lower bounds. However, it provides natural complete problems based on linear algebra for important complexity classes.

An important aspect of computing rigidity is its possible connection to the theory of natural proofs developed by Razborov and Rudich [17]. Valiant's reduction [20] identifies "high rigidity" as a a combinatorial property of the functions based on which he proves linear size lower bounds for log-depth circuits. However, the model of arithmetic circuits has not been studied in sufficient detail such that in the setting of natural proofs this can directly provide some evidence about the power of the proof technique. Nevertheless, this could be thought of as a motivation for the computational question of rigidity.

Our question bears close resemblance to the body of problems considered under *matrix completion*, see for instance [6,12]. Given a matrix with indeterminates in some locations, can we instantiate them in such a way that some desired

[1] Technically, rank over $\mathbb{Z}$ is not defined, since $\mathbb{Z}$ is not a field. In Section 2, we define a natural notion of rank over rings. Under this, since $\mathbb{Z}$ is an integral domain, the rank is the same as over the corresponding division ring $\mathbb{Q}$.

Table 2. Our bounds on RIGID when $k \in O(1)$ or $r = n$

$\mathbb{K}$, $S \subset \mathbb{K}$ (if –, then $S = \mathbb{K}$)	restriction	bound
$\mathbb{Z}$ or $\mathbb{Q}$, {0,1}		in NP
$\mathbb{Z}$ or $\mathbb{Q}$, {0,1}	$k \in O(1)$	$\mathsf{C}_=\mathsf{L}$-complete (Thm 4)
$\mathbb{Z}$ or $\mathbb{Q}$	$k \in O(1)$	$\mathsf{C}_=\mathsf{L}$-hard [3]
$\mathbb{Q}$	$r = n$	$\mathsf{C}_=\mathsf{L}$-complete [3] witness-search in $\mathsf{L}^{\mathsf{GapL}}$ (Thm 5)
$\mathbb{Z}$	$r = n$ and $k = 1$	in $\mathsf{L}^{\mathsf{GapL}}$ (Thm 6)
$\mathbb{Z}$ or $\mathbb{Q}$	bounded rigidity, $r = n$	NP-complete (Thm 7)
$\mathbb{Z}$ or $\mathbb{Q}$	bounded rigidity, $r = n, k = 1$	In PL, and $\mathsf{C}_=\mathsf{L}$-hard (Thm 8)

property (e.g. non-singularity) is achieved? In Section 4, we discuss how results from matrix completion can yield upper bounds for our question.

In this paper, we restrict our attention to $\mathbb{Z}$ and $\mathbb{Q}$ (some extensions to finite fields are discussed at the end). Since even an upper bound of NP is not obvious, we restrict the choice available in changing matrix entries. We consider two variants: (1) In the input, a finite subset $S \subseteq \mathbb{K}$ is given. M has entries over S, and the changed entries must also be from S; rank computation continues to be over $\mathbb{K}$. (For instance, we may consider Boolean matrices, so $S = \{0, 1\}$, while rank computation is over $\mathbb{Z}$.) It is easy to see that this variant is indeed in NP. (2) In the input, a bound θ is given. We require that the changes be bounded by θ; we may apply the bound to each change, or to the total change, or to the total change per row/column. (See for instance [13].) This version has close connections with another well-studied area called linear interval equations (see [18]) with applications in control systems theory. We obtain tighter lower and upper bounds for some of these questions. We show completeness for $\mathsf{C}_=\mathsf{L}$ when $k \in O(1)$ in the first variant, for NP when $r = n$ in the second variant, and for $\mathsf{C}_=\mathsf{L}$ when $r = n$ in the general case. Table 2 summarizes the results.

Due to shortage of space, many proofs are skipped; see [14] for details.

2 Preliminaries

Over any field $\mathbb{F}$, the rank of a $M \in \mathbb{F}^{n\times n}$ (we consider only square matrices in this paper) has the following equivalent definitions : (1) The maximum number of linearly independent rows or columns in M. (2) The maximum size of a non-singular square submatrix of M (3) The minimum r such that $M = AB$ for some $A \in \mathbb{F}^{n\times r}$ and $B \in \mathbb{F}^{r\times n}$. (4) The minimum r such that M is the sum of r rank-1 matrices, where a rank-1 matrix has every row as a multiple of the other rows. These definitions need not be equivalent when the underlying algebraic structure is not a field. Hence, the notion of rank is not well-defined over arbitrary rings. However, if the ring under consideration is an infinite integral domain (like $\mathbb{Z}$) (notice that a finite integral domain has to be a field), then the above definitions are indeed equivalent, and can be taken as a definition of rank. In fact, the rank

in that case can be easily seen to be same as the rank over the corresponding quotient field; thus rank over $\mathbb{Z}$ as defined above is the same as rank over $\mathbb{Q}$.

L and NL denote languages accepted by deterministic and nondeterministic logspace classes respectively, and FL is the class of logspace-computable functions. #L is the class of functions that count the number of accepting paths of an NL machine, and GapL is its closure under subtraction. Computing the determinant over $\mathbb{Z}$ or $\mathbb{Q}$ is complete for GapL . In contrast, computing the permanent is complete for # P , the class of functions counting accepting paths of an NP machine. NC^1 is the class of languages with polynomial size logarithmic depth Boolean circuits. $\#\mathsf{NC}^1$ is the class of functions computed by similar arithmetic circuits (gates compute + and $\times$), and GapNC^1 is its closure under subtraction. AC^0 (TC^0) is the class of languages with polynomial size constant depth unbounded fanin Boolean circuits, where gates compute AND, OR, NOT (and MAJORITY). For more details, see [23].

A language L is in the exact counting logspace class $\mathsf{C_{=}L}$ (or probabilistic logspace PL) iff it consists of exactly those strings where a certain GapL function is zero (positive, respectively). The languages

$$\text{SINGULAR}(\mathbb{K}) = \{M \mid \text{ Over } \mathbb{K},\ M \text{ is not full rank}\}$$

$$\text{RANK BOUND}(\mathbb{K}) = \{(M, r) \mid \text{ Over } \mathbb{K},\ \text{rank}(M) < r\}$$

for $\mathbb{K} = \mathbb{Z}$ or $\mathbb{Q}$ are complete for $\mathsf{C_{=}L}$ [3]. Note that for any type of matrices, and any complexity class $\mathcal{C}$, $\mathcal{C}$-hardness of SINGULAR implies $\mathcal{C}$-hardness of RANK BOUND. However the converse is not true:

Proposition 1. *Restricted to diagonal matrices,* SINGULAR($\mathbb{Z}$) *is in* AC^0 *while* RANK BOUND($\mathbb{Z}$) *and* DETERMINANT *are* TC^0*-complete.*

The rigidity function, and its decision version, are as defined below[2]. (Here $\text{support}(N) = \#\{(i,j) \mid n_{i,j} \neq 0\}$.)

$$R_M(r) \stackrel{def}{=} \inf_N \ \{\text{support}(N) : rank(M+N) < r\}$$

$$\text{RIGID}_{\mathbb{K}} = \{(M, r, k) \mid R_M(r) \leq k\}$$

Lemma 1. *(Valiant, folklore) Over any field* $\mathbb{F}$*,* $R_M(r+1) \leq (n-r)^2$*.*

3 Computing the Rank for Special Matrices

Computation of rank is intimately related to computation of the determinant. Mulmuley [15] showed that over arbitrary fields, rank can be computed in NC (with the the field operations as primitives). Over $\mathbb{Z}$ and $\mathbb{Q}$, RANK BOUND is $\mathsf{C_{=}L}$-complete ([3]), and we wish to characterize its subclasses by restricting the

[2] In much of the rigidity literature, $\text{rank}(M+N) \leq r$ is required. We use strict inequality to be consistent with the definition of RANK BOUND from [3].

types of matrices. A natural approach is to use characterizations of matrix rank in terms of associated combinatorial objects, like graphs. However, no known parameter of the graph of a matrix characterizes the matrix rank in general.

The following is easy to see:

Proposition 2. *The languages* RANK BOUND$(\mathbb{Z})$ *and* SINGULAR$(\mathbb{Z})$ *remain hard for* $\mathsf{C_=L}$ *even if the instances are restricted to be symmetric 0-1 matrices.*

However, we do not know similar hardness for DETERMINANT. While it remains GapL hard for 0-1 matrices, it is not clear that there are GapL -hard symmetric instances. (In a personal communication, Raghav Kulkarni has described to us why symmetric instances are GapL -hard under Turing reductions.)

We now consider an additional restriction. A matrix M is said to be *diagonally dominant* if for every i, $|m_{i,i}| \geq \sum_{j \neq i} |m_{i,j}|$. (If all the inequalities are strict, then M is said to be *strictly diagonally dominant.*) We show:

Theorem 1. SINGULAR$(\mathbb{Z})$ *restricted to non-negative diagonally dominant symmetric matrices is* L*-complete. The hardness is via uniform* TC^0*-computable many-one reductions.*

Proof Sketch: This result exploits a very nice combinatorial connection between such matrices and graphs. For a non-negative symmetric diagonally-dominant matrix M, its *support graph* $G_M = (V, E_M)$ has $V = \{v_1, \ldots v_n\}$, and $E_M = \{(v_i, v_j) \mid i \neq j \;\; m_{i,j} > 0\} \cup \{(v_i, v_i) \mid m_{i,i} > \sum_{i \neq j} m_{i,j}\}$. The following is shown in [8] for $\mathbb{R}$, and for matrices over $\mathbb{Q}$ this is same as the rank over $\mathbb{Q}$.

Lemma 2 ([8]). *Let M be a non-negative symmetric diagonally dominant matrix of order n over $\mathbb{Q}$ or $\mathbb{R}$. Then rank$(M) = n - c$, where c is the number of bipartite components in the support graph G_M.*

Note: the presence of a self-loop means a component is non-bipartite.

Membership in L now follows easily. To show L -hardness, we start with the L -hard problem of Undirected Forest Accessibility UFA and construct a graph which has no bipartite components on Yes instances but exactly one bipartite component on No instances. The graph construction is highly uniform, but going from the graph to the associated matrix requires TC^0 computations. □

Corollary 1. *The language* RANK BOUND$(\mathbb{Z})$*, restricted to instances that are symmetric non-negative diagonally dominant, is* L*-complete.*

Note that though rank for these matrices can be computed in L, we do not know how to compute the exact value of the determinant itself. (Note that by Theorem 1 this is hard for FL .) In a brief digression, we note the following: if a matrix is to have no trivial (all-zero) rows, and yet be diagonally dominant, then it cannot have any zeroes on the diagonal. How restrictive is this requirement? In general, it isn't too much so, as the following lemma shows. However, we do not know of a many-one reduction.

Lemma 3. *For every* GapL *function f and every input x, $f(x)$ can be expressed as* $\det(M) - 1$*, where M has no zeroes on the diagonal. M can be obtained from x via projections (each output bit is dependent on at most one bit of x).*

We also show via a somewhat different reduction that the L-hardness of SINGULAR in Theorem 1 holds even if we allow negative values, but disallow matrices with determinant other than 0 or 1.

Theorem 2. SINGULAR$(\mathbb{Z})$ *for symmetric diagonally dominant matrices is* L *-hard, even when restricted to instances with 0-or-1 determinant.*

Proof Sketch: As in Theorem 1, we begin with an instance (G, s, t) of UFA where G has two components, and add edge (s, t) to obtain graph H. The Laplacian matrix A of H is symmetric and diagonally dominant, and by the matrix-tree theorem, (see for e.g. Theorem II-12 in [4]), the determinant of its (1,1) minor counts spanning trees in H. But the number of spanning trees in H is 1 if $s \not\rightsquigarrow_G t$ (H itself is a tree) and is 0 if $s \rightsquigarrow_G t$ (H still has two components). □

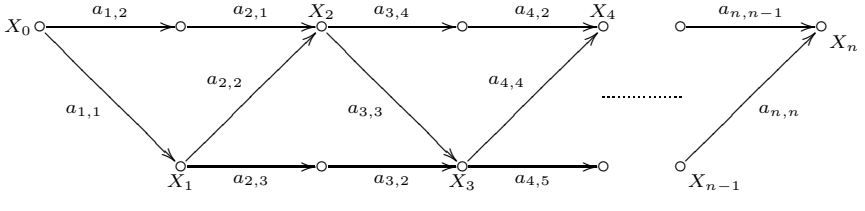
The next restriction we consider is tridiagonal matrices: $m_{i,j} \neq 0 \Longrightarrow |i-j| \leq 1$. We show that DETERMINANT and PERMANENT are in GapNC^1, by using bounded-width branching programs BWBP . In the the Boolean context, BWBP equals NC^1. However, in the arithmetic context, they are not that well understood. It is still open ([1,7]) whether the containment $\#\mathsf{BWBP} \subseteq \#\mathsf{NC}^1$ is in fact an equality. Layered planar BWBP are the G-graphs referred to in [2]. Counting paths in G-graphs may well be simpler than GapNC^1 due to planarity. However [2] (see also [1]) shows that even over width-2 G-graphs, it is hard for NC^1. We show that the permanent and determinant of tridiagonal matrices are essentially equivalent to counting in width-2 G-graphs.

Theorem 3. *Computing the permanent of a non-negative tridiagonal matrix over* $\mathbb{Z}$ *is equivalent to counting paths in a layered planar* BWBP *of width 2.*

Proof. Given a tridiagonal matrix A, let A_i be the top-left submatrix of A of order i, and let X_n and Y_n denote its permanent and determinant respectively. We have the following recurrences:

$$X_0 = Y_0 = 1 \qquad X_1 = Y_1 = a_{1,1}$$
$$X_i = a_{i,i}X_{i-1} + a_{i-1,i}a_{i,i-1}X_{i-2} \qquad Y_i = a_{i,i}Y_{i-1} - a_{i-1,i}a_{i,i-1}Y_{i-2}$$

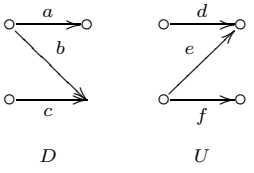
When all entries are 0-1, then it is easy to see that the branching program for X_n has width 2 and can be drawn in a layered planar fashion.(see below).



When the entries are larger, each edges can be replaced by a width-3 gadget with the appropriate number of paths in a standard way, giving width 5. If there are negative entries, or when we are computing the determinant, we need to either allow negative weights, or double the width and lose planarity.

To see the other direction, notice that any layer of a width-2 BWBP should look like one of the structures shown in figure.

Now any width-2 graph G corresponding to the BP can be encoded as a sequence of D and U components. First suppose that the encoded string has alternate DU. By just reading off the weights on the corresponding edges in the graph, we can produce two matrices M_1 and M_2 such that permanent of M_1 and the determinant of M_2 (by putting in appropriate negations) computes the number of weighted s-t paths in the graph.



Now it is sufficient to argue that the graph corresponding to any BWBP can be transformed to this form. If the string does not start with a D we will just put in a prefix D with $def = 101$. We need to handle the case when there are two UU and DD. Simply put in a D with $def = 101$ in between two U and a U with $abc = 101$ in between two Ds. Notice that the new width-2 graph when encoded will have only DU, and the weight of the paths are preserved in the transformation. The above reduction now gives the two matrices M_1 and M_2.

In addition, observe that if the BWBP has 0,1 weights then the matrix M_1 that we produce is also 0,1 and M_2 will have entries from $\{-1, 0, 1\}$. □

Corollary 2. *Computing the permanent and determinant of a tridiagonal matrix over $\mathbb{Z}$ is in* GapNC^1 *and is hard for* NC^1 *under* $\mathsf{AC}^0[5]$ *reductions.*

4 Complexity Results on Rigidity

In this section we study the problem of computing matrix rigidity, $\text{RIGID}_{\mathbb{K}}$, and also its restriction $\text{RIGID}_{\mathbb{K},S}$ defined below, where the matrices can have entries only from $S \subseteq \mathbb{K}$.

$$\text{RIGID}_{\mathbb{K},S} = \left\{ (M, r, k) \mid \begin{array}{c} M \text{ over } S, \ \ \exists M' \text{ over S} : \\ \text{rank}(M') < r \ \wedge \ \text{support}(M - M') \leq k \end{array} \right\}$$

We will mostly consider S to be either all of $\mathbb{K}$, or only $\mathbb{B} = \{0, 1\}$. We also consider the complexity of RIGID when k is fixed, via the following language:

$$\text{RIGID}_{\mathbb{K},S}(k) = \{(M, r) \mid (M, r, k) \in \text{RIGID}_{\mathbb{K},S}\}$$

As mentioned in the introduction, matrix rigidity and matrix completion are related. The MinRank problem takes as input a matrix with variables, and asks for the minimum rank achievable under all instantiations of the variables in the underlying field, see for instance [6]. 1-MinRank is its restriction where every variable occurs at most once, and is also called *minimum rank completion*. MaxRank and 1-MaxRank are similarly defined. The naive algorithm for rigidity, mentioned in the introduction, easily translates to an upperbound of NP(1-MinRank). While MinRank over $\mathbb{Z}$ is undecidable [6], this hardness proof does not carry over for 1-MinRank. Nonetheless, the best known upper bound for 1-MinRank is r.e.. Thus

the naive algorithm does not give any reasonable upper bound for RIGID. On the other hand, we do not know any hardness result for RIGID over $\mathbb{Q}$ or $\mathbb{Z}$ either.

We now consider restricted versions of the problem. The language $\text{RIGID}_{\mathbb{Z}}(0)$ is complete for $\mathsf{C}_=\mathsf{L}$, by [3]. When $k > 0$, we can still obtain some bounds provided S is finite. We have the following completeness result for one such case.

Theorem 4. *For each fixed k, $\text{RIGID}_{\mathbb{Z},\mathbb{B}}(k)$ is complete for $\mathsf{C}_=\mathsf{L}$.*

Proof Sketch: We show that for each k, $\text{RIGID}_{\mathbb{Z},\mathbb{B}}(k)$ is in $\mathsf{C}_=\mathsf{L}$. For a fixed k, there are only polynomially many changed matrices possible. Checking the rank of each of them can be done in $\mathsf{C}_=\mathsf{L}$. Since $\mathsf{C}_=\mathsf{L}$ is closed under logspace disjunctive truth-table reductions the membership follows.

To show corresponding hardness, note that the hardness for $\text{RIGID}_{\mathbb{Z},\mathbb{B}}(0)$ holds because SINGULAR remains $\mathsf{C}_=\mathsf{L}$-hard even when restricted to 0-1 matrices. Hardness for all the languages mentioned in the lemma also follows from this fact, and from the following claim:

$$M \in \text{SINGULAR}(\mathbb{Z}) \Longrightarrow (M \otimes I_{k+1}, n(k+1)-k) \in \text{RIGID}_{\mathbb{Z},\mathbb{B}}(0) \subseteq \text{RIGID}_{\mathbb{Z},\mathbb{B}}(k)$$
$$M \notin \text{SINGULAR}(\mathbb{Z}) \Longrightarrow (M \otimes I_{k+1}, n(k+1)-k) \notin \text{RIGID}_{\mathbb{Z}}(k)$$

Here $\otimes$ denotes tensor product and I_{k+1} denotes the $(k+1) \times (k+1)$ identity matrix. Note that $\text{rank}(M \otimes I_{k+1}) = (k+1)\text{rank}(M)$. The claim essentially follows from the fact that over any field $\mathbb{F}$, if two matrices M and N of the same order differ in exactly one entry, then their ranks can differ by at most 1. □

We also note that this result holds for any finite S, even if S is not fixed a priori but supplied explicitly as part of the input. The hardness of Theorem 4 essentially exploits the hardness of testing singularity. Therefore we now consider the complexity of RIGID at the singular-vs-non-singular threshold, i.e. when $r = n$. From Lemma 1 we know that over any field $\mathbb{F}$, (M, n, k) is in RIGID whenever $k \geq 1$. And $(M, n, 0)$ is in RIGID if and only if $M \in \text{SINGULAR}(\mathbb{F})$. So the complexity of deciding this predicate over $\mathbb{Q}$ is already well understood. We then address the question of how difficult it is to come up with a witnessing matrix.

Theorem 5. *Given a non-singular matrix M over $\mathbb{Q}$, a singular matrix N satisfying support$(M - N) = 1$ can be constructed in $\mathsf{L}^{\mathsf{GapL}}$.*

Another question that arises naturally is the complexity of RIGID at the singularity threshold over rings. Note that Lemma 1 does not necessarily hold for rings. For instance, changing one entry of a non-singular rational matrix M suffices to make it singular. But even if M is integral, the changed matrix may not be integral, and over $\mathbb{Z}$, $R_M(n)$ may well exceed 1. (It does, for the matrix $\begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$.) Thus, the question of deciding $R_M(n)$ over $\mathbb{Z}$ is non-trivial. We show:

Theorem 6. *Given $M \in \mathbb{Z}^{n \times n}$, deciding if (M, n, k) is in RIGID$(\mathbb{Z})$ is (1) trivial for $k \geq n$, (2) $\mathsf{C}_=\mathsf{L}$ complete for $k = 0$, and (3) in $\mathsf{L}^{\mathsf{GapL}}$ for $k = 1$.*

In particular, (3) above implies that if over $\mathbb{Z}$, $R_M(n) = 1$, then the non-zero entry of a witnessing matrix is polynomially bounded in the size of M.

However, if $R_M(n) > 1$ we do not know such a size bound. To demonstrate this difficulty, consider the case in which $k = 2$. Following the general idea in Theorem 5, for each choice of two entries in the matrix, replace them by variables x and y. This defines a family of $\binom{n}{2} = O(n^2)$ matrices and a family $\mathcal{P}$ of bilinear bivariate polynomials representing the corresponding determinants. The coefficients of each $p \in \mathcal{P}$ can be computed in GapL. Now, to test if $R_M(n) \leq 2$, it suffices to check if at least one of the Diophantine equations defined by $p \in \mathcal{P}$ (or equivalently, the single multilinear Diophantine equation $q(x_1, x_2 \ldots, y_1, y_2 \ldots) = \prod_{p \in \mathcal{P}} p(x_p, y_p) \ = 0$) has an integral solution.

5 Computing Bounded Rigidity

We now consider the bounded norm variant of rigidity described in Section 1: changed matrix entries can differ from the original entries by at most a prespecified amount θ. Formally, the functions of interest: the *norm rigidity* $\Delta_M(r)$ and *bounded rigidity* $R_M(r, \theta)$,as defined in [13], and the decision version, are

$$\Delta_M(r) \stackrel{def}{=} \inf_N \left\{ \sum_{i,j} |n_{i,j}|^2 : rank(M + N) < r \right\}$$

$$R_M(r, \theta) \stackrel{def}{=} \inf_N \ \{support(N) : rank(M + N) < r, \forall i, j : |n_{i,j}| \leq \theta\}$$

$$\text{B-RIGID}_{\mathbb{K}} = \{(M, r, k, \theta) \mid R_M(r, \theta) \leq k\}$$

Over $\mathbb{Z}$, the naive algorithm for B-RIGID$_{\mathbb{Z}}$ is now in NP . However over $\mathbb{Q}$, the bound θ still does not imply an a priori poly-size bound on the changed entries. Thus, unlike in Section 4, here computation over $\mathbb{Q}$ appears harder than over $\mathbb{Z}$.

The following lemma shows that the bounded rigidity functions can behave very differently from the standard rigidity function.

Lemma 4. *For any ϵ, and for any sufficiently large n such that $\frac{n}{\log n} > \epsilon + 1$, there is an $n \times n$ matrix M over $\mathbb{Q}$ such that $R_M(n) = 1$, $\Delta_M(n) = \Theta(4^n)$, and the bounded rigidity $R_M(n, n^\epsilon)$ is undefined.*

Since for a given a matrix M, a rank r and a bound θ, $R_M(r, \theta)$ can be undefined, we examine how difficult is it to check this. We show the following:

Theorem 7. *Given a matrix $M \in \mathbb{Q}^{n \times n}$, and a rational number $\theta > 0$, it is* NP*-complete to decide whether $R_M(n, \theta)$ is defined, and if Yes, is at most k.*

Proof. $R_M(r, \theta) \leq k$ iff there is a matrix N of rank r such that $\forall i, j : m_{i,j} - \theta \leq n_{i,j} \leq m_{i,j} + \theta$ and $m_{i,j} \neq n_{i,j}$ for at most k positions (i, j). A natural approach is to guess the $n_{i,j}$'s and compute the rank of N. However, for an NP upper bound, we need to show that if such an N exists, then in fact there exists such

an N' with entries having a bounded (poly sized) representation in terms of the input size. In [18] and [16], a poly-size bound is shown for the case when $r = n$.

For two matrices A and B, we say that $A \leq B$ if for each i,j, $A_{i,j} \leq B_{i,j}$. For $A \leq B$, the interval of matrices $[A, B]$ is the set of all matrices C such that $A \leq C \leq B$. An interval is said to be singular if it contains at least one singular matrix. Let J be the all 1-s matrix. Then the interval $[M-\theta J, M+\theta J]$ is singular if and only if $R_M(n, \theta)$ is defined. Now the hardness follows from a result from [16] showing that checking interval singularity is NP-hard. □

It is easy to see that, by clearing denominators, we have hard instances where M, θ take integral values. Thus, the hardness result holds for $\mathbb{Z}$ as well.

Unravelling the NP algorithm described in the membership part above, and its proof of correctness, is illuminating. Essentially, what is established in [18] and used in [16] is the following:

Lemma 5 ([18]). *If an interval $[A, B]$ is singular, i.e. the determinant vanishes for some matrix C within the bounds $A \leq C \leq B$, then the determinant vanishes for a matrix $D \in [A, B]$ which, at all but at most one position, takes an extreme value ($d_{i,j}$ is either $a_{i,j}$ or $b_{i,j}$).*

In particular, this implies that there is a matrix in the interval whose entries have representations polynomially long in that of A and B. This can be seen as follows: Let D be the matrix claimed to exist as above, and let k, l be the (only) position where $a_{kl} < d_{kl} < b_{kl}$. The other entries of D match those of A or B and hence are polynomially bounded anyway. Now put a variable x at k, l to get matrix D_x. Its determinant is a univariate linear polynomial $\alpha x + \beta$ which vanishes at $x = d_{kl}$. Now α and β can be computed from D_x in GapL, and hence are polynomially bounded. If $\alpha = 0$, then $\beta = 0$ and the polynomial is identically zero. Otherwise, the zero of the polynomial is $-\beta/\alpha$. Either way, there is a zero with a polynomially long representation.

In [18], the above lemma is established as part of a long chain of equivalences concerning determinant polynomials. However, it is in fact a general property of arbitrary multilinear polynomials, as we show below.

Lemma 6 (Zero-on-an-Edge Lemma). *Let $p(x_1 \ldots x_t)$ be a multilinear polynomial over $\mathbb{Q}$. If it has a zero in the hypercube H defined by $[\ell_1, u_1], \ldots [\ell_t, u_t]$, then it has a zero on an edge of H, i.e. a zero $(a_1, \ldots, a_t)$ such that for some k, $\forall (i \neq k)$, $a_i \in \{\ell_i, u_i\}$.*

Proof. The proof is by induction on the dimension of the hypercube. The case when $t = 1$ is vacuously true, since H is itself an edge. Consider the case $t = 2$. Let $p(x_1, x_2)$ be the multilinear polynomial which has a zero (z_1, z_2) in the hypercube H; $\ell_i \leq z_i \leq u_i$ for $i = 1, 2$. Assume, to the contrary, that p has no zero on any edge of H. Define the univariate polynomial $q(x_1) = p(x_1, z_2)$. Since $q(x_1)$ is linear and vanishes at z_1, $p(\ell_1, z_2)$ and $p(u_1, z_2)$ must be of opposite sign. But the univariate linear polynomials $p(\ell_1, x_2)$ and $p(u_1, x_2)$ do not change signs on the edges either, and so $p(\ell_1, u_2)$ and $p(u_1, u_2)$ also have opposite sign. By linearity of $p(x_1, u_2)$, there must be a zero on the edge $x_2 = u_2$, contradicting our assumption.

Let us assume the statement for hypercubes of dimension less than t. Consider the hypercube of dimension t and the polynomial $p(x_1, \ldots x_t)$. Let $(z_1 \ldots z_t)$ be the zero inside the hypercube. The multilinear polynomial r corresponding to $p(x_1, \ldots x_{n-1}, z_t)$ has a zero inside the $(t-1)$-dimensional hypecube H' defined by intervals $[\ell_1, u_1], \ldots [\ell_{t-1}, u_{t-1}]$. By induction, r has a zero on an edge of H'. Without loss of generality, assume that this zero is $(z'_1, \alpha_2 \ldots \alpha_{t-1})$ where $\alpha_i \in \{\ell_i, u_i\}$. Thus the polynomial $q(x_1, x_t) = p(x_1, \alpha_2 \ldots \alpha_{t-1}, x_t)$ has a zero in the hypercube defined by intervals $[\ell_1, u_1], [\ell_t, u_t]$. Hence the base case applies again, completing the induction. □

Analogous to Theorems 4, 5 and 6, we consider B-RIGID$_{\mathbb{K}}$ when $k \in O(1)$.

Theorem 8. B-RIGID$_{\mathbb{Q}}$ *and* B-RIGID$_{\mathbb{Z}}$ *are* $\mathsf{C_{=}L}$*-hard for each fixed choice of* k*, and remain hard when* $r = n$*. When* $k = 1$ *and* $r = n$*,* B-RIGID$_{\mathbb{Q}}$ *is in* PL*, while* B-RIGID$_{\mathbb{Z}}$ *is in* $\mathsf{L}^{\mathsf{GapL}}$*.*

Proof Sketch: For any k, $(M, n, k, 0) \in$ B-RIGID$_{\mathbb{K}} \iff M$ is singular; hence $\mathsf{C_{=}L}$-hardness.

To see the PL upper bound over $\mathbb{Q}$, let $\theta = \frac{p}{q}$. For each element (i, j), define the $(i, j)^{th}$ element as variable x and then write the determinant as $ax + b$. Thus, if $|x| = |\frac{b}{a}| \leq \frac{p}{q}$ for at least one such (i, j) pair, we are done. This is equivalent to checking if $(bq)^2 \leq (ap)^2$. a and b can be written as determinants, hence $(ap)^2$ and $(bq)^2$ are GapL functions, and comparison of two GapL functions can be done in PL. Since PL is closed under disjunction, the entire computation can be done in PL.

Over $\mathbb{Z}$, $q = 1$ and $\theta = p$, but we need an integral value for x as well. That is, we want an (i, j) pair where $|\frac{b}{a}| \leq \theta$ and a divides b. This can be checked in $\mathsf{L}^{\mathsf{GapL}}$. □

6 Discussion

While the matrix rigidity problem over finite fields is NP -complete ([10]), we can consider restricted versions there too. It is known [5] that SINGULAR$(\mathbb{F}_p)$ is complete for $\mathsf{Mod}_p\mathsf{L}$ (computing the exact value of the determinant over $\mathbb{F}_p$ is in $\mathsf{Mod}_p\mathsf{L}$.), and that (see e.g. [1]), for any prime p, RANK BOUND$(\mathbb{F}_p)$ is in $\mathsf{Mod}_p\mathsf{L}$. Using this, and closure properties of $\mathsf{Mod}_p\mathsf{L}$, we can obtain analogues of Theorem 4 and 5 over $\mathbb{F}_p$.

Acknowledgements

We thank V. Arvind, N.S. Narayanaswamy, R. Balasubramanian and Kapil Paranjape for many insightful discussions, and anonymous referees for useful comments.

References

1. Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajicek, J. (ed.) Complexity of Computations and Proofs, Quaderni di Matematica, Seconda Universita di Napoli, vol. 13, pp. 33–72 (2004)

2. Allender, E., Ambainis, A., Barrington, D.A.M., Datta, S., LeThanh, H.: Bounded-depth arithmetic circuits: counting and closure. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 149–158. Springer, Heidelberg (1999)
3. Allender, E., Beals, R., Ogihara, M.: The complexity of matrix rank and feasible systems of linear equations. Computational Complexity 8(2), 99–126 (1999)
4. Bollobas, B.: *Modern Graph Theory*. GTM, vol. 184. Springer, Heidelberg (1984)
5. Buntrock, G., Damm, C., Hertrampf, U., Meinel, C.: Structure and importance of logspace MOD-classes. Math. Systems Theory 25, 223–237 (1992)
6. Buss, J.F., Frandsen, G.S., Shallit, J.: The computational complexity of some problems of linear algebra. JCSS 58, 572–596 (1999)
7. Caussinus, H., McKenzie, P., Thérien, D., Vollmer, H.: Nondeterministic NC^1computation. JCSS 57, 200–212 (1998)
8. Dahl, G.: A note on nonnegative diagonally dominant matrices. Linear Algebra and Applications 317, 217–224 (1999)
9. Damm, C.: DET=$\mathrm{L}^{(\#\mathrm{L})}$. Technical Report Informatik-Preprint 8, Fachbereich Informatik der Humboldt–Universität zu Berlin (1991)
10. Deshpande, A.: Sampling-based dimension reduction algorithms. PhD thesis, MIT, May 2007 (expected)
11. Grigoriev, D.Y.: Using the notions of seperability and independence for proving the lower bounds on the circuit complexity. Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka, 1976 (in Russian)
12. Laurent, M.: Matrix completion problems. In: Floudas, C., Pardalos, P. (eds.) The Encyclopedia of Optimization, vol. 3, pp. 221–229. Kluwer, Dordrecht (2001)
13. Lokam, S.V.: Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity. JCSS 63(3), 449–473 (2001)
14. Mahajan, M., Sarma, J.M.N.: On the Complexity of Rank and Rigidity. Technical report (2006), http://eccc.hpi-web.de/eccc-reports/2006/TR06-100
15. Mulmuley, K.: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. Combinatorica 7, 101–104 (1987)
16. Poljak, S., Rohn, J.: Checking robust nonsingularity is NP-hard. Math. Control Signals Systems 6, 1–9 (1993)
17. Razborov, A.A., Rudich, S.: Natural proofs. JCSS 55(1), 24–35 (1997)
18. Rohn, J.: Systems of Linear Interval Equations. Linear Algebra and Its Applications 126, 39–78 (1989)
19. Toda, S.: Counting problems computationally equivalent to the determinant. manuscript (1991)
20. Valiant, L.G.: Graph theoretic arguments in low-level complexity. In: Gruska, J. (ed.) Mathematical Foundations of Computer Science 1977. LNCS, vol. 53, pp. 162–176. Springer, Heidelberg (1977)
21. Valiant, L.G.: Why is Boolean complexity theory difficult? In: Proceedings of the London Mathematical Society symposium on Boolean function complexity, pp. 84–94. Cambridge University Press, New York, NY, USA (1992)
22. Vinay, V.: Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In: Selman, A.L. (ed.) Structure in Complexity Theory. LNCS, vol. 223, pp. 270–284. Springer, Heidelberg (1986)
23. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Heidelberg (1999)

On the Usage of Clustering for Content Based Image Retrieval

Jorge R. Manjarrez Sanchez, Jose Martinez, and Patrick Valduriez

LINA, Université de Nantes
firstname.lastname@univ-nantes.fr, Patrick.Valduriez@inria.fr

Abstract. Retrieval of images based on the content is a process that requires the comparison of the multidimensional representation of the contents of a given example with all of those images in the database. To speed up this process, several indexing techniques have been proposed. All of them do efficiently the work up to 30 dimensions [8]. Above that, their performance is affected by the properties of the multidimensional space. Facing this problem, one alternative is to reduce the dimensions of the image representation which however conveys an additional loss of precision. Another approach that has been studied and seems to exhibit good performance is the clustering of the database. On this article we analyze this option from a computational complexity approach and devise a proposal for the number of clusters to obtain from the database, which can lead to sublinear algorithms.

1 Introduction

The retrieval of image takes place in a multidimensional space which is an abstract representation of the images in the database. It is the result of applying some analysis and processing techniques which processes the contents of the images and maps them to a multidimensional point, where each dimension can be, for example, a value of a bin of a color histogram [1]. The same process is applied to an image supplied as a query. Then this query point (or feature vector) is compared to all of those in the database. This process measures the similarity between them by computing some kind of distance, usually the Euclidean distance (L_2). The aim is to find the set of k most similar images in the database to the query; this is called k nearest neighbor (kNN) query processing which is the most usual kind of queries in content based image retrieval (CBIR) systems.

Processing kNN queries sequentially for a large database is inefficiently. Indexing methods, following a space or a data partitioning approach [2], allows to quickly find interesting regions and thus retrieving the set of k objects of interest. Cases of space partitioning are the X-tree [3] and the R-tree [4] and their derivatives, which are implemented using the vector space model, they take into consideration topological characteristics of the data space and to prune non interesting regions for higher dimensions it must visited almost all tree nodes, thus degrading their performance. Data partitioning methods can use a metric space representation [5][6] of multidimensional points. Thus the only consideration between images is their distance

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 281–289, 2007.

and takes advantage of the triangle inequality to find interesting ones. Notable approaches are M-tree [7] and iDistance [8]. The M-Tree is a balanced tree which optimizes disk IO's and distance evaluation costs. The iDistance order the multidimensional points based on their distance, and then uses a single dimensional B+Tree to process queries. This process still conveys a high cost in distance calculations which makes the CBIR process computationally expensive and the complexity increase exponentially with the number of dimensions.

Clustering, is a process that reduces the searching space by partitioning the images of the database into groups based on their similarity, the idea is to glue together images in a preliminary step, then to use a structure to find interesting cluster containing possibly the searched images [9][10][11], so a query can find in a cluster or relatively small set of clusters the images satisfying a kNN query. Though in clustering methods it must still be computed the distance to find the clusters of interest, they take advantage of a precomputed representative of each cluster, often called centroid. All the clusters centroids, act then as a general index of the database.

In this paper, our aim is to make a complexity analysis of the CBIR process using a clustering approach and to suggest a value to the number of clusters to partition the database so that the retrieval process can achieve a sublinear performance.

The rest of the article is structured as follows, in the next section we present our analysis for the estimated complexities of searching process in general, then in section 3 we state our conditions and derive the proposal based on the clustering of the database and section 4 reports some test on synthetic data, finally section 5 concludes.

2 Searching Without Clustering

The main problem CBIR face is the course of dimensionality. This means that it is not possible to index efficiently points in high dimensions. In general it was reported an average supported 15 dimensional space [12] for indexing methods, above that their performance is comparable with sequential search. Even when a recent proposal, the iDistance, reports performance up to 30 dimensions it has not been proved its limits.

The table 1 gives the principal complexities for the retrieval of k images in a database of size n, where k is a constant independent of and smaller than the size n.

The queries independent of the size of the database but tied to the result seems difficult to achieve. However, constant searches in $O(p)$ and those in $\mathrm{O}\left(\sqrt{n}\right)$ could be possible. In the notations, the factor $log_2 n$ is an optional sort.

Table 1. Some complexities for searching in a database of n images, eventually the k best with $k<<n$

constant	$\mathrm{O}\left(k\left\lceil \log_2 k \right\rceil\right)$
logarithmic	$\mathrm{O}\left(\log_2 n + k\left\lceil \log_2 k \right\rceil\right) = \mathrm{O}\left(\log_2 n\right)$ as $k\left\lceil \log_2 k \right\rceil \le \log_2 n$
Square root	$\mathrm{O}\left(\sqrt{n} + k\left\lceil \log_2 k \right\rceil\right) = \mathrm{O}\left(\sqrt{n}\right)$ as $k\left\lceil \log_2 k \right\rceil \le \sqrt{n}$
linear	$\mathrm{O}\left(n + k\left\lceil \log_2 k \right\rceil\right) = \mathrm{O}\left(n\right)$ as $k \log_2 k \le n$
sorted	$\mathrm{O}\left(n + n \log_2 n\right)$

In general, processing a query in logarithmic time is impossible because of the dimensionality of the space. However, when the number of dimensions is small enough it can be possible to use X-Trees, M-Tree or iDistance. But it implies that the image description is limited to small sizes, thus we can not represent with fidelity all the features' content of the images. Besides the above mentioned cases, which are not general, CBIR has linear performance. In the worst case the sequential search is followed by a general sort of the database, which gives the upper bound of an algorithm for CBIR, $O(n \log_2 n)$.

In practice, it must be taken into consideration not only the number of images n, but also the size of their abstract representation m. The relation between them is a constant $m = \lambda \cdot n$, more λ is bigger more are the repercussions on the performances. As an alternative to indexing we analyze the case of CBIR using a clustering of the database in the next section.

3 CBIR Using Clustering

To avoid the performance degradation exhibited by indexing methods, it is possible to proceed with a reduction of dimensions or a clustering process.

Dimensionality reduction is a process that transforms the data space into another less complex, but to be worth for CBIR, this transformation process must take into consideration as much as possible the characteristics of the initial space, to preserve them to that of arrival. For this reason, the weakness of dimensionality reduction approaches is an additional loss of precision. We are not going to discuss further about them.

Clustering suffers a priori of the same general problem of indexing methods. Indeed, multidimensional points are used as abstract representation of actual images and the searching process can not avoid distance calculations. But the intention is to compute as less as possible of them; and clustering methods can help by grouping together similar images in a compact entity, the cluster. Although clustering processes are computationally expensive, they can be executed off line in a preliminary step. Here, we do not suggest a specific clustering process but k-means is a process that can be used in conjunction with our proposal as it can accept, as an input, the desired number of clusters. Here we concentrate into taking advantage of it to provide efficient CBIR.

Assume the case of a search via some data clustering, then we can write the generic complexity as:

$$O(f(C)) + O(g(C')) \tag{1}$$

Where:

- $|C| \leq n$ is the number of clusters produced by a classification algorithm;
- $f(C)$ is the search complexity on the clusters;
- $|C'| \leq |C|$ is the number of clusters susceptible of contain enough *similar* images;
- $g(C')$ is the search complexity on the C' clusters of multidimensional points.

The standard complexities for *f(C)* and *g(C)* are reported in tables 2 and 3, respectively.

We can observe from table 2 that the logarithmic traversal of a clustered database, supposed hierarchical and well balanced, presents little importance. In fact, it sets a weak limit on the number of clusters susceptible of containing enough images of interest: $C' \leq log_2 C$. We will see that this hypothesis imposes a restriction (c.f. C'_{sup} in table 4).

Table 2. Some complexities *f(C)* for the sequential traversal of *C* clusters

Constant	$O(C')$
Logarithmic	$O(\log_2 C + C')$
linear	$O(C)$

Table 3 has a rewrite of the complexities of table 1, estimating that each cluster is in average of the same size, assume it is $O\left(\frac{n}{C}\right)$. The eventual complexities of results merging are inferior to those of each cluster; i.e, in the case *constant*, the selection for each cluster in $O(C'k[\log_2 k])$, can be followed by a step of merging that is in $O(C'k)$ even with a naïve algorithm.

Table 3. Some complexities *g(C)* for searching in a database of *n* images, with eventually the best *k*, with $k << n$

constant	$O(C'k[\log_2 k])$
logarithmic	$O\left(C'\log_2\frac{n}{C} + C'k[\log_2 k]\right)$
square root	$O\left(C'\sqrt{\frac{n}{k}} + C'k[\log_2 k]\right)$
linear	$O\left(C'\frac{n}{C} + C'k[\log_2 k]\right)$
sorted	$O\left(C'\frac{n}{C}\log_2\left(C'\frac{n}{C}\right)\right)$

The generic complexity (1), introduce an optimization problem. Which in order to achieve optimal processing it should satisfy the following constraints:

- to minimize *O(g(C'))* as a function of the number of candidate classes, i.e., $|C'| << |C|$, and the number of the selected classes;
- to ensure that $O(f(C)) \leq O(g(C'))$;
- to ensure that $|C| << n$.

Let us consider the worst case with:

- a linear selection of the clusters;
- a sequential scan within each selected cluster;
- a full sort based on the concatenation of the results issued from the selected clusters.

Lemma 1 (**upper bound for retrieval using clustering**). Under the abovementioned conditions, the general complexity in (1) becomes:

$$O(C) + O\left(C' \cdot \frac{n}{C} + C' \cdot \frac{n}{C} \cdot \log_2 \left(C' \cdot \frac{n}{C} \right) \right) \tag{2}$$

An thus, the searching algorithm based on data clustering is optimal in $O(\sqrt{n}\log_2 n)$, with clusters of size:

$|C| = \sqrt{n}\log_2 n$; $|C'| \leq \log_2 n.$ and clusters of similar cardinalities.

Proof. First simplify by setting C'=1. Then let propose $C = \sqrt{n}$ and substituting in equation (2) gives a complexity in $O\left(\sqrt{n} + \frac{n}{\sqrt{n}} \log_2 n \frac{n}{\sqrt{n}} \right) = O\left(\sqrt{n}\log_2 \sqrt{n} \right) = O\left(\sqrt{n}\log_2 n \right)$

with a multiplicative constant equal to $\frac{1}{2}$. Second, let also propose $C = \sqrt{n}\log_2 n$,

then equation (2) becomes:

$$O\left(\sqrt{n}\log_2 n + \frac{n}{\sqrt{n}\log_2 n} \log_2 n \frac{n}{\sqrt{n}\log_2 n} \right) = O\left(\sqrt{n}\log_2 n + \frac{\sqrt{n}}{n\log_2 n} \log_2 n \frac{\sqrt{n}}{n\log_2 n} \right)$$

$$= O\left(\sqrt{n}\log_2 n + \frac{\sqrt{n}}{\log_2 n} \left(\frac{1}{2}\log_2 n - \log_2 \log_2 n \right) \right) = O\left(\sqrt{n}\log_2 n \right)$$

The second proposition makes asymptotically equal the two terms, that is to say the optimal algorithm. Having said that, the complexity of the two approaches are the same. They define a range of validity for the number of clusters.

Substituting the proposed cardinalities, the complexity of (2) becomes

$$O\left(\sqrt{n}\log_2 n \right) + O\left(\log_2 n \cdot \frac{n}{\sqrt{n}\log_2 n} + \log_2 n \cdot \frac{n}{\sqrt{n}\log_2 n} \cdot \log_2 \left(\log_2 n \cdot \frac{n}{\sqrt{n}\log_2 n} \right) \right),$$

i.e.,

$O\left(\sqrt{n}\log_2 n \right) + O\left(\sqrt{n} + \sqrt{n}\log_2 \left(\frac{1}{2} \cdot \log_2 n \right) \right)$ which is certainly in $O\left(\sqrt{n}\log_2 n \right)$ ■

Notice that from the proof it can be derived some algorithmic variations, from optimal to suboptimal in $O(\sqrt{n}\log_2^2 n)$, under the conditions less restrictives, $C \in \lfloor \sqrt{n}, \sqrt{n}\log_2 n \rfloor$ and $C' \leq log_2 n$, the optimal case can be obtained with a near multiplicative factor λ, since C' is small and independent of n.

Table 4. Illustration of usability conditions for some sizes of images databases, assuming small sizes for personal collections then bigger for professional

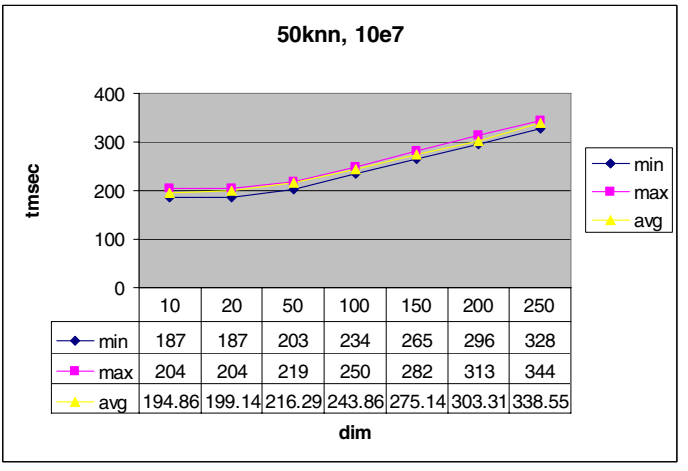
n	1, 024	8,192	32,768	1,048,576	33,554,432
$C_{inf}\sqrt{n}$	32	91	181	1,024	5,793
$C_{sup}\sqrt{n}\log_2 n$	320	1,177	2,715	20,480	144,815
$n'_{inf}\ \frac{n}{C_{\inf}} = \sqrt{n}$	32	91	181	1,024	5,793
$n'_{sup}\ \frac{n}{C_{\inf}} = \log_2 n$	3	7	12	51	232
Selectivity $[\lambda]\frac{1}{\sqrt{n}}$	3,13 %	1,10 %	0,55 %	0,10 %	0,02 %
$C'_{inf}\ \frac{\frac{n}{\sqrt{n}}}{n_{\inf}} = \log_2 n$	1	1	1	1	1
$C'_{sup}\ \frac{\frac{n}{\sqrt{n}}}{n_{\sup}} = \log_2 n$	11	13	15	20	25
Speedup $\frac{n + n\log_2 n}{\sqrt{n}\log_2 n} \approx \sqrt{n}$	35	97	193	1,075	6,024

This lemma is important since, thanks to the clustering hypothesis, it allows the design of a sub-linear search by-content algorithm, using basic algorithms which are not. The proposed algorithm is order of magnitudes under the sequential scan, though it is still much slower than the best theoretical achievement which would be in $log_2 n$.

Additionally, this lemma gives us the (asymptotic) optimal number of classes (i.e., it accepts small variations), which can be used as a parameter by the clustering algorithm. Table 4 shows computed values for different database sizes arranged in columns.

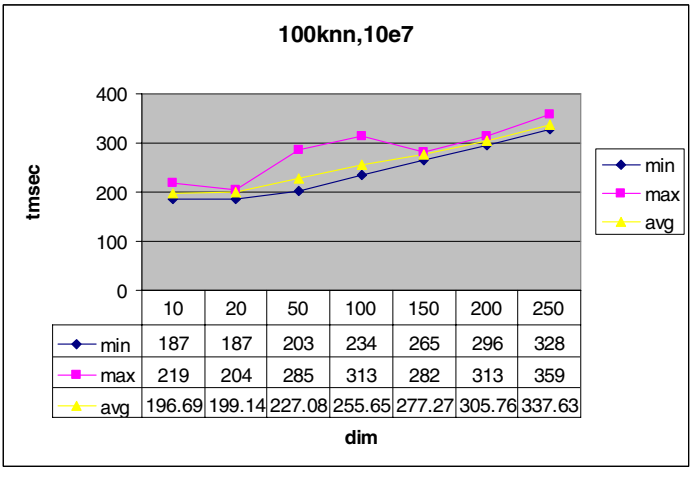
4 Experiments

To test the actual efficiency of our proposal, we have developed an experimental platform. (It is written in Java 1.5 running on a Pentium processor at 3 GHz with 1 GB of main memory). We generated synthetic clusters following the normal



	10	20	50	100	150	200	250
min	187	187	203	234	265	296	328
max	204	204	219	250	282	313	344
avg	194.86	199.14	216.29	243.86	275.14	303.31	338.55

Fig. 1. Processing time for 50 kNN queries using different dataset dimensionalities

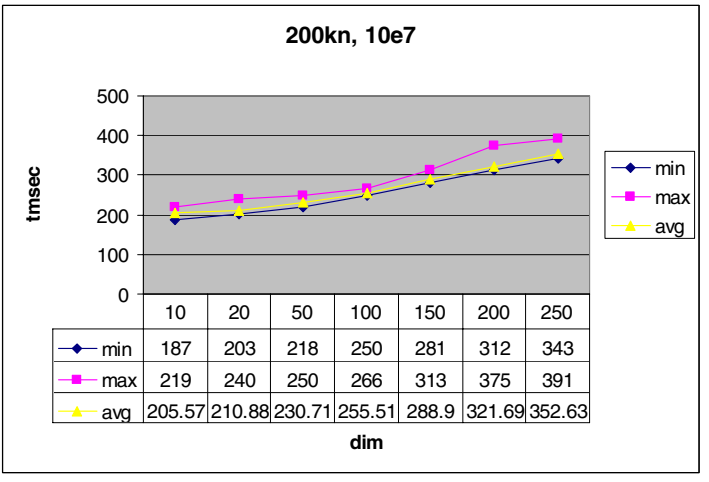


	10	20	50	100	150	200	250
min	187	187	203	234	265	296	328
max	219	204	285	313	282	313	359
avg	196.69	199.14	227.08	255.65	277.27	305.76	337.63

Fig. 2. Processing time for 100 kNN queries using different dataset dimensionalities

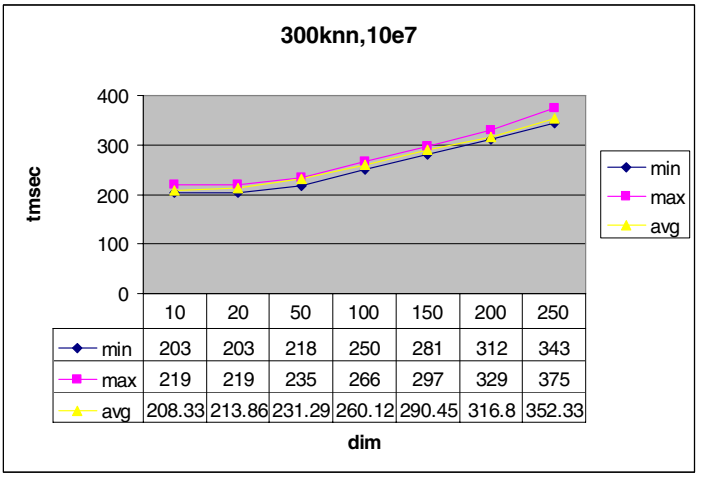
distribution for dimensions 10,20,50,100,150, 200 and 250 dimensions. The databases varied in size but here we show results for 10^7. The test consisted in generating 1000 random queries, then to process kNN queries for k=50,100, 200 and 300.

Searching process scans and prunes clusters centroids at the first time and the selection of the most interesting ones is based on their distance from the query point. No indexing structure is used and all the centroids and data points are stored in disk. Thus, measuring the behavior on the worst case as the centroids could fit into main memory and hence speeding up the pruning process.



	10	20	50	100	150	200	250
min	187	203	218	250	281	312	343
max	219	240	250	266	313	375	391
avg	205.57	210.88	230.71	255.51	288.9	321.69	352.63

Fig. 3. Processing time for 200 kNN queries using different dataset dimensionalities



	10	20	50	100	150	200	250
min	203	203	218	250	281	312	343
max	219	219	235	266	297	329	375
avg	208.33	213.86	231.29	260.12	290.45	316.8	352.33

Fig. 4. Processing time for 300 kNN queries using different dataset dimensionalities

We can observe that results are lightly influenced by the dimensionality of the dataset. This is due to the reduced searching space provided by the partitioning of the database into clusters of similar images thus reducing the time to seek for interesting regions. Also the size of the clusters make unnecessary to speed up the search process at the interior of each one of the selected ones. Just the merging process altogether with a pruning of the result to satisfy k is necessary.

5 Conclusions

Here we have presented a proposal for partitioning a database into a set of clusters. The proposal gives under worst case assumptions a range of acceptable values for the number of clusters. If we consider voluminous databases the proposal can then be used as a parameter estimation to the number clusters to place in the parallel machine. This is indeed a further work, altogether with experiments for non uniform cluster sizes, as derived analytically.

Acknowledgements

The first author thanks the support of the CONACYT and IPN from Mexico, as well as INRIA France.

References

1. Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P.: Query by Image and Video Content: The QBIC System. IEEE Computer 28(9), 23–32 (1995)
2. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. 33(3), 322–373 (2001)
3. Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-tree: An Index Structure for High-Dimensional Data. In: Proceedings of the 22th International Conference on Very Large Data Bases, pp. 28–39 (September 03-06, 1996)
4. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the, ACM SIGMOD international conference on Management of data, Boston, Massachusetts (June 18-21, 1984)
5. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
6. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces (Survey Article). ACM Trans. Database Syst. 28(4), 517–580 (2003)
7. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proceedings of the 23rd international Conference on Very Large Data Bases (August 25-29, 1997)
8. Jagadish, H.V., Ooi, B.C., Tan, K., Yu, C., Zhang, R.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. 30(2), 364–397 (2005)
9. Li, C., Chang, E., Garcia-Molina, H., Wiederhold, G.: Clustering for approximate similarity search in high-dimensional spaces. IEEE Transactions on Knowledge and Data Engineering 14(4), 792–808 (2002)
10. Yu, D., Zhang, A.: ClusterTree, ClusterTree: Integration of Cluster Representation and Nearest Neighbor Search for Large Datasets with High Dimensionality. IEEE Transactions on Knowledge and Data Engineering 15(5), 1316–1337 (2003)
11. Chen, Y., Wang, J.Z., Krovetz, R.: CLUE: Cluster-based Retrieval of Images by Unsupervised Learning. IEEE Transactions on Image Processing 14(8), 1187–1201 (2005)
12. Weber, R., Schek, H.-J., Blott, S.: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: VLDB 1998. Proc. 24th Int. Conf. Very Large Data Bases, pp. 194–205 (1998)

Performance Modeling of Wormhole Hypermeshes Under Hotspot Traffic

Reza Moraveji[1,2], Hamid Sarbazi-Azad[3,1], Abbas Nayebi[1,3], and Keyvan Navi[2]

[1] PM School of Computer Science, Tehran, Iran
[2] Shahid Beheshti University, Tehran, Iran
[3] Sharif University of Technology, Tehran, Iran
moraveji_r@ipm.ir, nayebi@ce.sharif.edu, azad@sharif.edu, navi@sbu.ac.ir

Abstract. Traffic pattern is a point of concern in today's modeling approach of network-based computing systems including NoC's and Clusters. Hypermesh is a promising network topology suitable for a range of networks. Although there are few models reported for hypermeshes with uniform traffic pattern, no analytical model has been reported yet that deal with hotspot traffic load. Since uniform traffic assumption is not always justifiable in practice as there are many parallel applications that exhibit non-uniform traffic patterns, which can produce hotspots in the network, in this study we propose a novel analytical model to analyze the mean message latency in wormhole-switched hypermesh in the presence of hot-spot traffic.

Keywords: Performance modeling, Interconnection network, Hypermesh, Hot-spot traffic.

1 Introduction

An *n*-dimension radix-*k* hypermesh is a regular topology that consists of *k* nodes in each dimension. Instead of having direct connection to the neighbors in each dimension, as in mesh and k-ary n-cube, each node in the hypermesh is connected to all the nodes in each dimension [1]. This topology has a very low diameter, and the average distance between nodes scales very well with network size. The hypermesh [6] has been well-evaluated using simulation experiments. Few works have been conducted on the performance modeling and analysis of hypermeshes [10, 11]. These models deal with uniform traffic pattern and adaptive routing.

Since uniform traffic assumption is not always justifiable in practice, in this study we propose a new analytical model to analyze the mean message latency in wormhole-switched hypermesh in the presence of hot-spot traffic. To our knowledge, no performance model has been proposed for hypermeshes under the non-uniform traffic load, such as hotspot traffic, with fully adaptive routing. Our analysis reveals that the proposed model exhibits good accuracy even under heavy traffic loads and near saturation region where other models usually fail to predict the behavior of the network.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 290–302, 2007.

2 The Hypermesh

The hypermesh topology $HM_{k,n}$, has $N(HM)=k^n$ nodes, arranged as n dimensional grid with k node in each dimension. Nodes in each dimension are connected together as a complete graph. A node in an n-dimension hypermesh with radix k consists of a processing element and a router. The processing element is comprised of a processor and a local memory. The router has $n(k\text{-}1)$ input physical channels as well as an injection channel and $n(k\text{-}1)$ output physical channels in addition to an ejection channel. The router contains flit buffers for each incoming channel. The flit buffers associated with each channel may be organized into several lanes (or virtual channels), and the buffers in each virtual channel can be allocated independently [7]. The input and output virtual channels are connected by a crossbar switch that can simultaneously connect multiple input channels to multiple output channels given that there is no contention over the output channels. Assuming that each physical channel in the hypermesh network is associated with V virtual channels, a $(n\cdot(k-1)+2)\cdot V$ - way crossbar switch inside the router directs messages from any input channel to any output channel.

The $HM_{k,n}$ is regular and symmetric, has a node degree of $N_d(HM)=n\cdot(k-1)$, a diameter of $d(HM)=n$, and $Ch(HM)=k^n\cdot(k-1)\cdot n$ channels.

Deadlock free fully adaptive routing in the hypermesh [3] requires V virtual channels per physical channel to ensure deadlock freedom, where V virtual channels are split into two classes A and B. Class A contains V_1 virtual channels, while Class B includes V_2 virtual channels. Virtual channels of class A are used by a fully adaptive routing function, and virtual channels of class B are used by a deadlock-free routing function. As the deadlock free routing used in this study is dimensioned-ordered routing, therefore in our simulation we set the value of V_1 to one and the value of V_2 to (V-*1*). The main reason behind this assumption is to use virtual channels efficiently. The minimum virtual channels required to ensure deadlock freedom is then two, one in class A and one in class B.

3 The Analytical Model

The model uses assumptions that are widely used in the literature [3-5, 8, 9]:

1) According to the traffic model used to generate hot-spot traffic load, each generated message at a source node is destined to the hot-spot node with the probability of α and to other nodes of the network with the probability of $(1-\alpha)$. Let us refer to these two types of messages as hot-spot and regular messages, respectively [12].
2) The proposed model assumes that there is one hot-spot node in the network.
3) Nodes generate traffic independently of each other, which follows a Poisson process with a mean rate of λ_g messages/cycle, consisting of regular and hot-spot portions of $(1-\alpha)\lambda_g$ and $\alpha\lambda_g$, respectively. The destination node of a regular message is randomly chosen among network nodes.

4) Messages are assumed to have fixed length of *msgl* flits. The transfer time of a flit between any two adjacent nodes is assumed to be one cycle. Also, intra-node delay is assumed to be equal to the delay of routing arbitration applied to the header flit and is assumed to be t_r unit cycles.
5) Messages at the destination node are transferred to the local PE one at time via the ejection channel.
6) To implement a fully adaptive routing *V* virtual channels are used per physical channel: V_1 virtual channels are used by the fully adaptive routing sub-function and V_2 virtual channels are used by the deterministic (deadlock free) routing sub-function. To simplify the model derivation, no distinction is made between the deterministic and adaptive virtual channels when computing the different virtual channels occupancy probabilities [4].

The mean message latency is composed of the mean network latency, $\overline{S}$, that is, the time to cross the network and the mean waiting time seen by a message in the source node, W_s. However, to capture the effects of virtual channels multiplexing, the mean message latency has to be scaled by a factor, $\overline{V}$, representing the average degree of virtual channels multiplexing that takes place at a given physical channel. Therefore, we can write the mean message latency as [4]:

$$Latency = \left(\overline{S} + W_s\right) \times \overline{V} \tag{1}$$

If $\overline{S}_r$ and $\overline{S}_h$ denote the mean network latency for regular and hot-spot messages, respectively, the mean network latency of the network, taking into account both types of messages, can be given by:

$$\overline{S} = (1-\alpha).\overline{S}_r + \alpha.\overline{S}_h \tag{2}$$

Therefore, the mean network latency for regular messages can be formulated as

$$\overline{S}_r = \frac{1}{\left(k^n - 1\right)\left(k^n - 1\right)} \sum_{(S,D) \in G \times G, D \neq H} S_{(S,D)} \tag{3}$$

where $S_{(S,D)}$ is the network latency seen by a regular message originated from a specific source node *S*, and destined to another specific destination node *D*. Therefore,

$$S_{(S,D)} = T_{Transmission} + T_{Nodal} + T_{\mathbf{B}locking,r}^{(S,D)} = Diff(S,D)(t_r + t_s) + (msgl) \cdot t_s + T_{Blocking,r}^{(S,D)} \tag{4}$$

where $T_{Blocking,r}^{(S,D)}$ is the mean blocking time seen by the regular message traveling between *S* and *D*. We can divide the nodes into *n* groups, each group consists of the nodes located *j* hops away from the hot-spot node where $1 \le j \le n$. Taking into account the number of nodes in each group, $\binom{n}{j}(k-1)^j$, the mean network latency for hot-spot messages can be calculated as

$$\overline{S}_h = \frac{\sum_{j=1}^{n} \left(\binom{n}{j} (k-1)^j \times S_{h,j} \right)}{k^n - 1} \tag{5}$$

where $S_{h,j}$ is the network latency seen by a hot-spot message originated from a node located j hops away from the hot-spot node. Therefore, $S_{h,j}$, considering both static and dynamic delay parts, can be calculated as

$$S_{h,j} = j \cdot (t_r + t_s) + (msgl) \cdot t_s + \sum_{i=1}^{j} T_{Blocking,i} \tag{6}$$

where $T_{Blocking,i}$ is the mean blocking time seen by the hot-spot message at its i^{th} hop along the path to the destination node.

In order to calculate the blocking time of a j-hop hot-spot message, the aggregation of all blocking times on the nodes over which the message may traverse should be calculated. Since all possible nodes which the message may visit on the i^{th} hop of its path are located $(j-i)$ hops away from the hot-spot node, the blocking times of the message on these nodes are equal. Therefore, it is better to classify the blocking times according to the message's steps along its path. Hence, we have the term $\sum_{i=1}^{j} T_{Blocking,i}$ for the total blocking time that a j-hop hot-spot message sees.

Blocking time of a hot-spot message in the i^{th} step of its path is approximated by the product of $P_i^{Block}(j-i)$ which is the blocking probability of the hot-spot message on the node located j-i hops away from the hot-spot node to the mean waiting time tolerated by the message on the given node to acquire a virtual channel. Thus,

$$T_{Blocking,i} = \overline{W}_{h,j-i} \times P_i^{Block}(j-i) \tag{7}$$

We have to find all possible nodes on which the message may position along its path and then adding up all blocking times tolerated by the given message on these nodes. Thus, we have

$$T_{Blocking,r}^{(S,D)} = \sum_{x \in G_{(S,D)}} T_{Blocking}(x) \tag{8}$$

where $G_{(S,D)}$ is the set of all nodes located in at least one of the minimal paths between (S,D) pair. The blocking time for a regular message on node x, $T_{Blocking}(x)$, is approximated by the product of the probability of blocking on the given node and the *minimum* waiting time tolerated by the message to acquire a virtual channel at the given node. Therefore, we have

$$T_{Blocking}(x) = P_{(S,D)}^{Block}(x) \times \overline{W}_{r,x} \tag{9}$$

Average message arrival rate on a specific channel consists of two parts, the arrival rate of hot-spot messages and the arrival rate of regular messages. Consider the set J of all channels located j hops away from the hot-spot node. For a specific channel that

is j hops away from the hot-spot node, the number of source nodes located at a distance of i hops from the given channel and $(i + j)$ hops from the hot-spot node that their hot-spot message traverses the given channel with the probability greater than zero is $\binom{n-j}{i} \cdot (k-1)^i$. By varying i from j to n-j, the whole number of source nodes can be found as $\sum_{i=j}^{n-j} \binom{n-j}{i} \times (k-1)^i$.

The probability of *visiting one node* which is located j hops away from the hot-spot node by a hot-spot message originated from a source node located i hops away from the given node and $(i + j)$ hops away from the hot-spot node in which the given node is located in the minimal path between the source node and the hot-spot node is computed using the following recursive formula as [12]

$$P_i^{Pass}(j) = P_{i-1}^{Pass}(j+1) \times \frac{i}{(j+1)} = \prod_{m=0}^{i-1} \frac{i-m}{j+1+m} \tag{10}$$

considering the termination condition (initial state) of $P_0^{Pass}(j+i) = P_{Source} = 1$.

Probability of *traversing a channel* which is located j hops away from the hot-spot node by a hot-spot message originated from a source node located i hops away from the given node and $(i + j)$ hops away from the hot-spot node in which the given channel is located in the minimal path between source node and hot-spot node, only depends on the location of the channel from the hot-spot node. The mentioned probability can be given by [12]

$$P_i^{Pass} < j > = P_{Pass}(j) \times \frac{1}{(j+1)} = \frac{1}{(j+1)} \times \prod_{m=0}^{i-1} \frac{i-m}{(j+1)+m} \tag{11}$$

The traffic rate or the mean arrival rate of hot-spot messages on a channel located j hops away from the hot-spot node is given by

$$\lambda_{h,j} = \alpha \times \lambda_g \times \sum_{i=j}^{n-j} \left(\binom{n-j}{i} (k-1)^i \times P_i^{Pass} < j > \right) \tag{12}$$

By considering the source of message, regular messages are also divided into two groups. The first group contains the regular messages originated from the hot-spot node and the second group contains the regular messages originated from all other nodes. Consider a regular message in the former group; there are $k^n - 1$ destination choices to which the message may be sent. Therefore, the probability that the given message reaches a specific destination node is $1/(k^n - 1)$. In the latter group, since the hot-spot node can not be the destination of a regular message, this probability gets riser by considering the fact that the number of choices is decreased by one. Therefore, the probability that the message is destined to a specific node becomes $1/(k^n - 2)$.

Let $P_{(S,D)}^{Pass} < j >$ indicate the probability of traversing channel $< j >$ (located j hops away from the hot-spot node) by a regular message with known source and destination nodes, the rate of the first group of regular messages with specific source-destination pair on the aforementioned channel can be given by

$(1-\alpha)\times\lambda_g\times\frac{1}{k^n-1}\times P^{Pass}_{(S,D)}<j>$ and the rate of the second group is given by $(1-\alpha)\times\lambda_g\times\frac{1}{k^n-2}\times P^{Pass}_{(S,D)}<j>\cdot$

Taking into account both message arrival rates, the mean arrival rate of regular messages traveling between *S* and *D* via channel $<j>$ is

$$\lambda_{r(S,D)}<j>=(1-\alpha)\times\lambda_g\times P^{Pass}_{(S,D)}<j>\times\left(\frac{1}{k^n-2}+\frac{1}{k^n-1}\right) \tag{13}$$

Generalizing the above formula by summing up all message arrival rates generated by all source-destination pairs, the mean arrival rate of regular *messages* on channel $<j>$ becomes

$$\lambda_{r,j}=\sum_{(S,D)\in G\times G, D\neq H}\lambda_{r(S,D)}<j> \tag{14}$$

Note that for the pairs in which channel $<j>$ does not locate in the minimal path, $P^{Pass}_{(S,D)}<j>$ will be equal to zero, regarding to the fact that the messages traveling between these pairs have no effect on the traffic rate of channel $<j>$. Taking into account both hot-spot and regular traffic rates, the mean message arrival rate on channel $<j>$ is can be formulated as

$$\lambda_j=\lambda_{h,j}+\lambda_{r,j} \tag{15}$$

The blocking probability of all permitted physical channels on node x is $P_{Block}(x)$ and in a similar manner the blocking probability of a physical channel emanating from the node x is $P_{Block}<x,y>$ (blocking probability of all adaptive and deterministic virtual channels of the physical channel $<x,y>$) [12]. Therefore, we have,

$$P_{Block}(x)=\sum_{y\in OUTch}P_{Block}<x,y> \tag{16}$$

where *OUTch* is the set of all endpoints of the output channels from node x specified by the routing algorithm to direct the given message. Note that set *OUTch* depends on the source and destination of the given message. We define the blocking probability of the message traveling between *S* and *D* via node x as the product of blocking probability of all permitted physical channels on node x to the probability of visiting (passing) node x as

$$P^{Block}_{(S,D)}(x)=P^{Pass}_{(S,D)}(x)\cdot P_{Block}(x) \tag{17}$$

For computing the probability of visiting node x, $P^{Pass}_{(S,D)}(x)$, by a message traveling between specific source and destination nodes, it should be noted that the number of

possible input channels through which the message can enter the node may vary from one node to the other, even if the distances of the nodes from the hot-spot node are equal. A value obtained by adding set of probabilities of passing input channels (located in a minimal path between source and destination nodes) to an arbitrary node like x , by the message, is the probability of visiting node x which can be given as

$$P_{(S,D)}^{Pass}(x) = \sum_{z \in INch} P_{(S,D)}^{Pass} < z, x > \tag{18}$$

where *INch* (endpoints of permitted input channels) is the set of all adjacent nodes to the node x with the property

$$INch = \{z | Diff(z, D) = Diff(x, D) + 1\} \tag{19}$$

that guaranties the placement of the nodes on the minimal path. In order to calculate $P_{(S,D)}^{Pass} < z, x >$, it is necessary to calculate the probability of traversing channel $< z, x >$ by the message emanating from node z given that the message is in node z , $P_{(S,D)}^{Pass}(< z, x > | z)$. We have the following dimensionless equation.

$$P_{(s,d)}^{Pass}(< x, y_i > | x) = \sum_{\substack{for\,all\,binary\,permutations \\ of\,A\,with\,(A_i = x) = 0}} \left(\frac{1}{\sum_{i=0}^{|A|-1} (1 - a_i)} \left(\prod_{i=0}^{|A|-1} (P_v < x, y_i >)^{a_i} \cdot (1 - P_v < x, y_i >)^{(1-a_i)} \right) + \left\{ \prod_{i=0}^{|A|-1} P_v < x, y_i > \right\}_{W(\min)=w} \right) \tag{20}$$

If two or more physical channels are free to use (the corresponding fields are zero), the probability of traversing each one by the message will be the same. This is the reason why the above equation contains term $1/\sum_{i=0}^{|A|-1}(1-a_i)$. $P_{Block} < z, i >$ is the blocking probability of an output channel emanating from node z and is calculated. The probability of traversing channel $< z, x >$ can be calculated as

$$P_{(S,D)}^{Pass}(< z, x >) = P_{(S,D)}^{Pass}(z) \times p_{(S,D)}^{Pass}(< z, x > | z) \tag{21}$$

Furthermore, the probability of visiting node $x, P_{(S,D)}^{Pass}(x)$, can be obtained iteratively as $P_{(S,D)}^{Pass}(x) = \sum_{z \in INch} P_{(S,D)}^{Pass} < z, x >$ and $P_{(S,D)}^{Pass} < z, x >= P_{(S,D)}^{Pass}(z) \times p_{(S,D)}^{Pass}(< z, x > | z)$ with terminating condition $P_{(S,D)}^{Pass}(x = S) = 1$.

Consider an $(i + j)$-hop hot-spot message visits a node, (j), located j hops away from the hot-spot node. The blocking probability of the given message in node (j) is calculated as

$$P_i^{Block}(j) = P_i^{Pass}(j) . P_{Block}(j) \tag{22}$$

where $P_{Block}(j)$ is the blocking probability of hot-spot message in the node (j) given that the message is in node (j). Therefore, the blocking probability of the hot-spot message given that it is in node (j) becomes [12]

$$P_{Block}(j) = P_{a,j}^{(j-1)} \times P_{a\&d,j} \tag{23}$$

and the probability of interest can be written as

$$P_i^{Block}(j) = \prod_{m=0}^{i-1} \frac{i-m}{(j+1)+m} \times \left(P_{a,j}^{(j-1)} \times P_{a\&d,j} \right) \tag{24}$$

Let $P_{v,j}$ be the probability that *V* virtual channels at a physical channel located *j* hops away from the hot-spot node are busy. The probability that all adaptive virtual channels at a physical channel that is *j* hops away from the hot-spot node are busy, $P_{a,j}$, can be written as [12]

$$P_{a,j} = \sum_{i=0}^{V_2} P_{v-i,j} \times \begin{pmatrix} V-V_1 \\ i \end{pmatrix} \Big/ \begin{pmatrix} V \\ V-i \end{pmatrix} \tag{25}$$

In the same manner, $P_{a\&d,j}$ which indicates the probability that all adaptive and deterministic virtual channels at a physical channel that is *j* hops away from the hot-spot node are busy can be formulated as [12]

$$P_{a\&d,j} = \sum_{i=0}^{V_2-1} P_{v-i,j} \times \begin{pmatrix} V_2-1 \\ V-i-V_1 \end{pmatrix} \Big/ \begin{pmatrix} V \\ V-i \end{pmatrix} \tag{26}$$

The waiting time seen by the message at a physical channel located *j* hops away from the hot-spot node to have at least one free virtual channel among the associated adaptive and deterministic virtual channels, is calculated by modeling the given physical channel as a single-server system with Poisson arrivals and arbitrary service-time distribution (M/G/1 queue). Thus, we have [8]

$$W_{<j>} = \frac{\rho_j \cdot S_j \cdot \left(1 + C_{S_j}^2\right)}{2\left(1-\rho_j\right)} \tag{27}$$

where ρ_j is the utilization factor of the channel located *j* hops away from the hot-spot node and in the case of single-server system its definition becomes

$$\rho_{<x,y>} = (Average\ Message\ Arrival\ Rate) \times (Average\ Service\ Time),\ \rho_j = \lambda_j \times S_j,\ C_{S_j}^2 = \sigma_{S_j}^2 \Big/ S_j^2$$

where $\sigma_{S_j}^2$ is the variance of service time distribution; following the suggestion made by Draper and Ghosh [8] according to the fact that the minimum service time at a

channel is equal to the message length, we can approximately write $\sigma_{S_j}^2 = \left(S_j - msgl\right)^2$. Note that S_j is the mean service time experienced by the message on a channel which is *j* hops away from the hot-spot node. Putting all together, the mean waiting time at a physical channel located *j* hops away from the hot-spot node can be written as

$$W_{<j>} = \frac{\lambda_j \cdot S_j^2 \cdot \left(1 + \frac{\left(S_j - msgl\right)^2}{S_j^2}\right)}{2\left(1 - \lambda_j \cdot S_j\right)} \tag{28}$$

The mean waiting time of the hot-spot message on the given node is equal to the waiting time of one of the related channels. Therefore, we have

$$\overline{W}_{h,j} = W_{<j>} \tag{29}$$

If a regular message is blocked on the node located *j* hops away from the hot-spot node, the emanating channels from the given node can be divided into channels located *j* or (*j* + 1) hops away from the hot-spot node, and therefore the waiting time of a regular message in the given node becomes

$$\overline{W}_{r,j} = Min(W_{<j>}, W_{<j+1>}) = W_{<j+1>} \tag{30}$$

The average service time of a channel considering its distance from the hot-spot node and also the effects of both hot-spot and regular messages, taking into accounts their appropriate weights, can be written as [4]

$$S_j = \frac{\lambda_r}{\lambda_j} \cdot \overline{S}_{r,j} + \frac{\lambda_h}{\lambda_j} \cdot \overline{S}_{h,j} \tag{31}$$

where $\overline{S}_{r,j}$ is the average service time experiences by regular messages to cross a channel located *j* hops away from the hot-spot node and $\overline{S}_{h,j}$ is the mean service time tolerated by hot-spot messages until crossing the given channel.

The mean waiting time in the source node is calculated in a similar way of a network channel. A message in the source node can enter the network through any of V virtual channels. Modeling the local queue in the source node that is *j* hops away from the hot-spot node as an M/G/1 queue with the mean arrival rate of λ_g / V, yields the mean waiting time as

$$\overline{W}_{s,j} = \frac{\left(\lambda_g / V\right) \cdot \overline{S}_j^{\,2} \cdot \left(1 + \left(\frac{\overline{S}_j - msgl}{\overline{S}_j}\right)^2\right)}{2\left(1 - \left(\lambda_g / V\right) \cdot \overline{S}_j\right)} \tag{32}$$

where $\overline{S}_j$ is the mean network latency for a message originated at a source node located *j* hops away from the hot-spot node. Therefore, considering both hot-spot and regular messages, and also taking into account their weights, we can write

$$\overline{S}_j = (1-\alpha)\cdot \overline{S}_{r,j} + \alpha \cdot \overline{S}_{h,j} \tag{33}$$

Taking into account the number of nodes located *j* hops away from the hot-spot node, $\binom{n}{j}(k-1)^j$, as the weight of the waiting time on the related source nodes, the mean waiting time at the source node becomes

$$W_s = \frac{\sum_{j=1}^{n}\left(\binom{n}{j}(k-1)^j \times \overline{W}_{s,j}\right)}{k^n} \tag{34}$$

The probability, $P_{v,j}$, that v virtual channels are busy at a physical channel located *j* hops away from the hot-spot node can be determined by a Markovian model as [7]

$$P_{v,j} = \begin{cases} \dfrac{1}{\sum_{i=0}^{V} Q_{i,j}} & \text{v}=0 \\ P_{0,j}\cdot Q_{v,j} & 1 \le v \le V-1 \end{cases}, \quad Q_{v,j} = \begin{cases} \left(\lambda_j \cdot S_j\right)^v & 0 \le v \le V-1 \\ \dfrac{\left(\lambda_j \cdot S_j\right)^v}{1-\lambda_j \cdot S_j} & v = V \end{cases} \tag{35}$$

The average degree of virtual channel multiplexing that takes place at a physical channel located *j* hops away from the hot-spot node can be estimated as [7]

$$\overline{V}_j = \frac{\sum_{v=1}^{V} v^2 \cdot P_{v,j}}{\sum_{v=1}^{V} v \cdot P_{v,j}} \tag{36}$$

Averaging over all multiplexing degrees for all groups of channels located in different distances from the hot-spot node, and taking into account the number of channels located *j* hops away from the hot-spot node, $\binom{n}{j}(k-1)^j$, the average degree of virtual channel multiplexing can be formulated as

$$\overline{V} = \frac{\sum_{j=1}^{n}\left(\left[\binom{n}{j}(k-1)^j\left((k-1)\cdot n - j\right) + (j+1)\cdot\binom{n}{j+1}(k-1)^{j+1}\right]\times \overline{V}_j\right)}{k^n \cdot n(k-1)} \tag{37}$$

4 Model Validation

The above model has been validated through a discrete-event simulator that performs a time-step simulation of the network operations at the flit level. Each simulation

experiment was run until the network reached its steady state. Extensive validation experiments have been performed for several combinations of network sizes, message lengths, and virtual channels and the general conclusions have been found to be consistent across all the cases considered. However, for the sake of specific illustration, figures 1 and 2 depict latency results predicted by the above model plotted against those provided by the simulator for the $HM_{4\times4\times4}$ only (more validation results can be seen in [12]). Message length *msgl*=32 and *msgl*=64 flits; Number of virtual channels per physical channel *V*= 4; Fractions of hot-spot traffic are $\alpha = 0$, 0.07, 0.21 and 0.35.

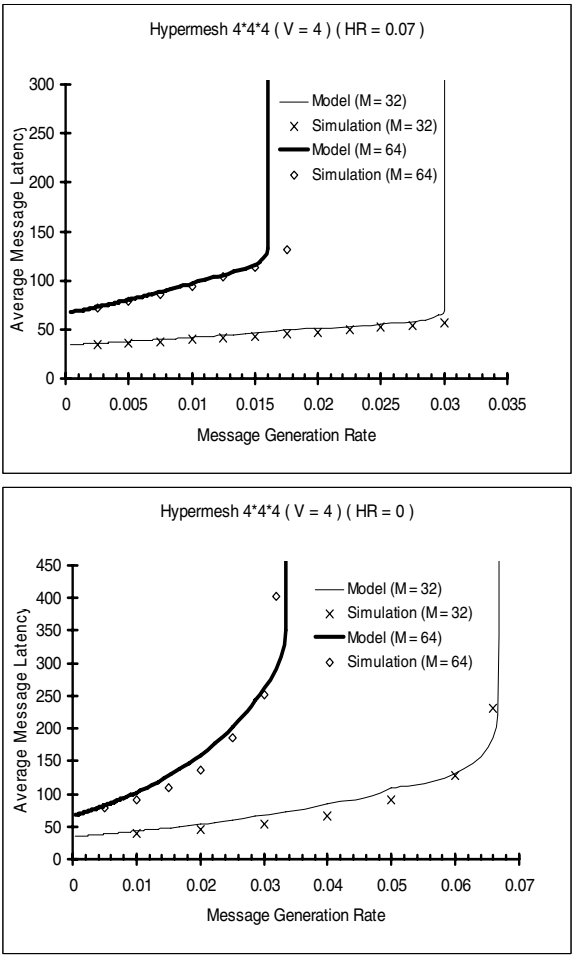


Fig. 1. Mean message latency predicted by the model against simulation in the 4*4*4 hypermesh: *V*=4 virtual channels, message lengths 32 and 64 flits, and different hot spot fractions α =0 (uniform traffic), 0.07

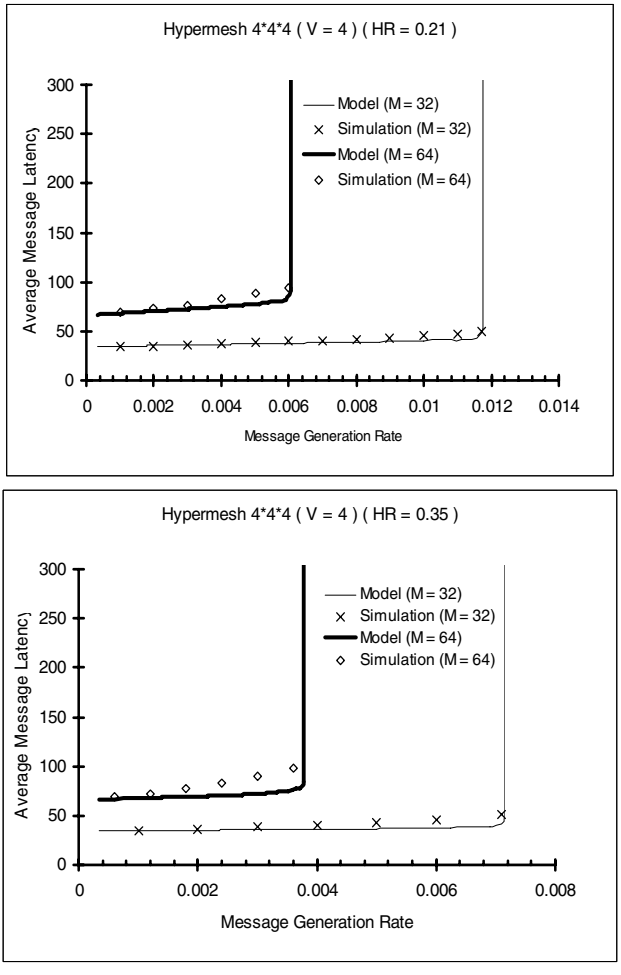


Fig. 2. Mean message latency predicted by the model against simulation in the 4*4*4 hypermesh: V=4 virtual channels, message lengths 32 and 64 flits, and different hot spot fractions α=0.21, 0.35

Figures 1 and 2 reveal that in all cases, the analytical model predicts the mean message latency with a good degree of accuracy in the steady-state regions. Moreover, the model predictions are still good even when the network operates in the heavy traffic region, and when it starts to approach the saturation region. However, some discrepancies around the saturation point are apparent. These can be accounted for by the approximations made to ease derivation of different equations of the model, e.g. the variance of the service time distribution at a channel. Such approximations greatly simplified the model as they allow us to avoid computing the exact distribution of the message service time at a given channel, which is not a

straightforward task due to inter-dependencies between service times at successive channels as wormhole routing relies on a blocking mechanism for flow control.

5 Conclusions

In this paper, we proposed an analytical model to calculate the average message latency in hypermesh topology in the presence of hot-spot traffic when fully adaptive wormhole routing is used. As reported by simulation results, the model exhibits a reasonable accuracy even when the rate of hot-spot messages is high. Future works in this line can be investigating a general performance model for irregular networks and for any traffic patterns.

References

1. Szymanski, T.: Hypermeshes: optical interconnection networks for parallel computing. J. Parallel and Distributed Computing 26, 1–23 (1995)
2. Duato, J., Yalamanchili, S., Ni, L.: Interconnection networks: An engineering approach. IEEE Computer Society Press, Los Alamitos (2003)
3. Loucif, S.: Performance evaluation of distributed crossbar switch hypermesh. Ph.D. Thesis, Computer Science Department, Glasgow University (1999)
4. Ould-Khaoua, M., Sarbazi-Azad, H.: An analytical model of adaptive wormhole routing in hypercubes in the presence of hot-spot traffic. IEEE Trans. Parallel and Distributed Systems 12, 283–292 (2001)
5. Najaf-abadi, H.H., Sarbazi-azad, H., Rajabzadeh, P.: Performance modeling of fully adaptive wormhole routing in 2-d mesh-connected multiprocessors. Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp. 528–534 (2004)
6. Loucif, S., Ould-Khaoua, M., Mackenzie, L.M.: The express channel concept in hypermeshes and k-ary n-cubes. Parallel and Distributed Processing, 566–569 (1996)
7. Dally, W.J.: Virtual channel flow control. IEEE Trans. Parallel and Distributed Systems 3, 194–205 (1992)
8. Draper, J.T., Ghosh, J.: A comprehensive analytical model for wormhole routing in multicomputer systems. J. Parallel and Distributed Computing 23, 202–214 (1994)
9. Sarbazi-azad, H., Ould-Khaoua, M., Mackenzie, L.M.: Analytical modeling of wormhole-routed k-ary n-cubes in the presence of hot-spot traffic. IEEE Trans. Parallel and distributed systems 50, 623–634 (2001)
10. Loucif, S., Ould-Khaoua, M.: On the merits of the hypermesh and k-ary n-cubes with adaptive routing. J. Systems Architecture 47, 795–806 (2002)
11. Loucif, S., Ould-Khaoua, M., Al-Ayyoub, A.: Hypermeshes: implementation and performance. J. Systems Architecture 48, 37–47 (2002)
12. Moraveji, R., Sarbazi-Azad, H., Nayebi, A., Navi, K.: Hypermesh modeling under hotspot traffic pattern, Technical Report, IPM School of Computer Science, Tehran, Iran (2006)

Application of Modified Coloured Petri Nets to Modeling and Verification of SDL Specified Communication Protocols

Valery A. Nepomniaschy, Gennady I. Alekseev, Victor S. Argirov, Dmitri M. Beloglazov, Alexander V. Bystrov, Eugene A. Chetvertakov, Tatiana G. Churina, Sergey P. Mylnikov, and Ruslan M. Novikov

A.P.Ershov Institute of Informatics Systems
Siberian Division of Russian Academy of Sciences
6, Lavrentiev ave., Novosibirsk 630090, Russia
vnep@iis.nsk.su

Abstract. In order to simplify simulation and verification of SDL specified communication protocols, we introduce modified coloured Petri nets called hierarchical timed typed nets (HTT-nets). A method for translation from SDL into HTT-nets is presented. A tool SPV (SDL protocol verifier) including a translator from SDL into HTT-nets, as well as means for editing, simulating, visualizing and verifying the net models, is described. For verification, the tool SPV uses a model-checking method. As case studies, we apply the tool SPV to RE-protocol [4], ATMR protocol [10] and *i*-protocol [5].

1 Introduction

The analysis, validation and verification of communication protocols are a challenge for computer science. In spite of considerable progress in theoretical research, obtained results find a limited use in modern practice. The formal description technique SDL accepted as an international standard [18] is widely used to represent communication protocols. Therefore, development of methods and tools for analysis and verification of SDL specified communication protocols is an important problem. It should be noted that high expressive power of SDL increases difficulties in verification of communication protocols.

A natural approach to overcome the problem is to use the models like finite state machines, Petri nets or their generalizations. Coloured Petri Nets (CPN) [11] should be distinguished among them because they have significant expressive power, a wide application, and simulation and analysis tools available [15, 17].

However, constructing the models of distributed systems manually is unreliable. Therefore, the problem of automatic construction of the net models arises for SDL specified distributed systems. It should be noted that the problem of automated translation of SDL specifications into CPN has been mentioned in [11]. Translation from SDL into so-called SDL time nets which extend conventional Petri nets by time intervals and guards for transitions has been described in [6]. Translation from SDL into high level Petri nets called M-nets has been described in [7]. A method for

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 303–314, 2007.

translation from an SDL dialect called TNSDL and from the standard SDL into high level Petri nets similar to Predicate / Transition nets has been described in [9] and [1], respectively.

Different tools have been implemented for the analysis, simulation and verification of these net models. Such tools as SITE [6], PEP [8], Emma [9], Maria [1], Design/CPN [15], CPN Tools [17] should be mentioned. A model-checking method is used by the tools PEP, Emma and Maria for verification of the net models.

In order to simplify the simulation and verification process in many cases, we introduce modified CPN called hierarchical timed typed nets (HTT-nets) and develop a method for translation from SDL into HTT-nets. We have implemented a tool SPV (SDL protocol verifier) which includes a translator from SDL into HTT-nets, as well as means for editing, simulating, visualizing and verifying the net models.

The purpose of the paper is to describe this method of translation, the tool SPV and its application to modeling and verification of communication protocols. The paper consists of 6 sections. HTT-nets are presented in Section 2. The translation method is described in Section 3. The tool SPV is presented in Section 4. Application of the tool SPV to RE-protocol [4], ATMR protocol [10] and *i*-protocol [5] is outlined in Section 5. Results and perspectives of our approach are discussed in Section 6.

This work is partly supported by Russian Foundation for Basic Research under grant 07-07-00173.

2 Net Model

Let us describe our net model resulting in a modification of CPN [11]. A nonhierarchical net consists of three parts: a net structure, declarations and a net marking. A hierarchical coloured net is a composition of a number of nonhierarchical nets called pages. Pages can contain a special kind of transitions called modules. Modules are connected with places on a page in the same way as transitions.

A module is a subnet placed on a separate page. The subnet can also contain modules. Pages presenting these modules are called subpages. A subpage contains a copy of each place ("copy-place"), which has been connected to the module. A copy-place can be an input/output place for some transition or module on the subpage if its prototype is an input/output place, respectively, for the module.

The behavior of a hierarchical net is equivalent to the behavior of a nonhierarchical net, where each module is replaced by the corresponding subnet. Connections within nonhierarchical nets are those obtained by substitution of transitions in hierarchical CPN, where each prototype is glued with all its copy-places.

An HTT-net has the basic set of colours corresponding to the standard data types: integer, real, string and boolean. A set of colours can be described by enumerating all possible values. Constructed types (arrays and records) are represented by tuples. There are also special places representing queues of tokens (queue-places).

Let us distinguish the type *multi-layer* which is the set of colours consisting of pairs *(n, t),* where *n* is the layer number and *t* is some colour. A place marked as *multi-layer* is called the *multi-layer place.* Any multi-layer place in which the type *t* is not a queue, can contain at most one token belonging to some layer. Otherwise, such a place can contain at most one queue belonging to some layer. The model allows only a finite number of layers.

Multiple firing of a transition is permitted in HTT-nets. At any time a transition can have a number of tokens in the input places sufficient for two or more simultaneous firings. Thus, one transition can fire several times simultaneously, but every time it fires only with tokens belonging to one layer. HTT-nets are *semi-safe* meaning that all places, except queue-places, can contain at most one token belonging to each layer.

The HTT-nets are provided with the Merlin's time mechanism [2] and priorities. A pair of non-negative numbers d_{min} and d_{max} is assigned to each transition. If t is the time moment when the transition is enabled, then it fires within the time range $[t+d_{min}, t+d_{max}]$.

The priority of a transition is defined by a non-negative integer number. The larger number defines the higher priority of the transition, as usual. The absence of the priority corresponds to the lowest priority.

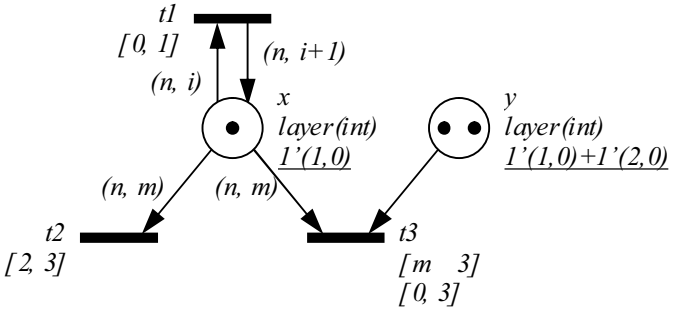


Fig. 1. Example of HTT-net

Let us consider an HTT-net given on Fig. 1. There is one integer token with the value 0 in the place *x* of the layer with number 1. There are two tokens in the place *y*, where one of them is in the layer with number 1, and another is in the layer with number 2. Each token has value 0. Each arc expression has the variable *n* in its first position. Thus, all transitions can fire with the tokens belonging to the same layer. The transitions *t1* and *t2* have no guards and their firings are defined only by timer constructions. The transition *t3* is enabled if the value of the token in the place *x* is less than or equal to 3. At first, the transitions *t1* and *t3* can fire. The transition *t2* can not fire because at least two time units have to pass before its firing can occur. At firing of the transition *t3*, the tokens of the places *x* and *y* (of the layer with number 1) are removed, and after that all transitions can not fire.

3 Translation of SDL Specifications into HTT-Nets

Let us describe generating an HTT-net model for a given SDL specification. Details of our translation method can be found in [3]. Pages which represent a system, blocks, processes, SDL-transitions and procedures are created by steps. At the first step, a page is created which represents a net corresponding to the general structure of the SDL specification. It contains one module per block. Nets related to the modules

which have been created at the first step, are constructed at the second step. These nets represent the structure of blocks and they are placed on separate pages connected with the modules. At the next step, the nets modeling the processes definitions are created and placed on separate pages. After that, transitions of the SDL system are translated into HTT-transitions and, finally, the pages containing the nets corresponding to the procedures defined in the specification are built.

Generation of top-level nets. One module is created in the net for each block definition. Channels and signal routes are represented by special places – queue-places. The signals which can be kept in every queue-place, are presented by a queue of tokens. Each token is a record that carries the information about the sender and receiver processes. Channels which connect blocks to one another or to the system boundary, are presented by queue-places connected with the modules by arcs.

A one-direction channel is presented by one place. The place modeling the channel, through which the signals only come into the block, is the input place to the module corresponding to the block. The place modeling the channel, through which the signals only come out of the block, is the output place of the module. A bi-direction channel is represented by two places, where one of them is the input place of the module and another is the output one.

All instances of a process are modeled by a single net constructed according to the process definition but each instance has its own tokens in the multi-layer places of the net. The first field of the tokens in all places, except otherwise specified, contain "PId" (Process Identifier) of the process instances. The *Pid* service place is created to generate unique PId values.

Thus, after the first step of the modeling, we have a net which consists of the modules corresponding to the SDL blocks, the places modeling the channels and the places modeling the exported variables and the place *Pid*.

Nets of the second level. The inner structure of each block is mapped onto the page related to this block. This page is built in the same way as the first page. The net from this page contains one transition for each new block or process definition, one or two places for each internal channel (uni- or bi-directional).

According to the rules of HTT-net generation, a copy of each place representing an external channel appears on the page related to the module. The places representing the internal channels or routes are merged with the places representing the external channels. Several internal channels can join to one external channel. Several places representing internal channels can be merged with one place representing an external channel.

Some peculiarities appear when we construct a net for a block consisting of processes. A signal coming through a channel to the block can be delivered through several signal routes connected to this channel. Our method supports the signal transition from an input signal route to all instances of the process connected to this signal route. To this end, an additional subnet is created. This subnet models the delivery of a signal to all instances of the process.

To provide a possibility for modeling a process creation, a special place *Pid* was created at the first step. At this step, several places cr_id (one for each process definition) are created on the block page. The *Pid* place is used to ensure uniqueness

of the PId value for each process instance. The *cr_id* place (where id is the offspring process name) is used to connect parent and offspring processes.

Nets of the third and fourth levels. Different instances of a process are modeled by a single net with multi-layer places. The tokens belonging to a process instance are of the same layer corresponding to this instance. The tokens are added into the places at the moment of the instance creation. They are removed when the process instance enters the STOP state. The service place *cr_id,* where *id* is the name of the offspring process, is the input and output place to the *activate* transition in the net modeling this offspring process. A new instance is created by firing the *activate* transition, i.e. tokens with the corresponding initial values are added to all output places of this transition.

A net fragment, which models the creation of the new PId value, is presented at Fig. 2. Assume that the process instance which creates the process *A* is modeled by the tokens belonging to the layer with the number *n.*

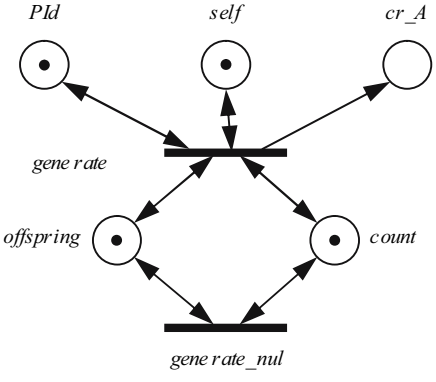


Fig. 2. Creation of the new PId value

At the firing of the transition *generate,* the following actions take place:

- the token with some value *p* is removed from the place *PId* and is added to it with the value *(p + 1)*;
- the token from the place *count* which is at the net page modeling the process *A* is removed and added to it with the same value;
- the token with the value *(n, n)* is removed from the place *self* and added to it;
- the token from the place *offspring* is removed and added to it with the value *(n, p)*;
- the token *(p, n)* is added to the queue at the place *cr_A, where p* is PId of the new instance of the process *A* and *n* is PId of the father instance.

At the firing of the transition *generate_nul*, the token with the value of the last created offspring is removed from the place *offspring* and added to it with the value *e*. This transition can fire when the number of instances of the process *A* is maximum in the system.

The inner structure of a process is mapped on a page connected with a module corresponding to the process definition. The net contains one transition for each SDL transition of the process definition. The net contains one multi-layer place for each SDL variable. The sort of the variable defines the colour set of the corresponding place.

Besides, this page contains several service multi-layer places. For example, the place *queue* presents the input ports of all instances of the same process and contains the queues of tokens corresponding to the queues of signals to all instances of the process. The multi-layer place *State* is associated with the states of the instances. This place is the input and output place of each transition modeling the SDL transition. Thus, the atomicity of the SDL transition execution is guaranteed.

A subnet for an SDL transition is created on the page connected with the module corresponding to the transition. Execution of some SDL transition can be modeled by one transition in the net. Such SDL transition contains a set of assignment statements and, perhaps, *OUTPUT* statements. The guard and the expressions on the surrounding arcs of the transition are defined at this step. The definition of coloured nets ensures that the execution of such SDL transition and firing of the corresponding net transition results in an equivalent change of the global state of the process instance and the net marking, respectively.

An SDL transition is called complex if it is not modeled by one net transition. Complex SDL transitions are divided into fragments. Each fragment is represented by a subnet. The subnets are connected consequently by means of connective places. The execution of a complex SDL transition is modeled by consequent firings of all net transitions.

Net modeling of the SET statement. Let us consider modeling of an SDL transition containing the statement *SET(N,t).* The modeling net is shown on Fig. 3. It has transitions *begin*, *end*, as well as the subnets *net1* and *net2* which represent statements standing before and after the statement *SET* in the SDL transition. The subnet *SAVE* is indicated by a dotted line. This net models saving all signals in a queue to a process instance except the signal *t*. To avoid overloading, some places and transitions, as well as arcs of the subnet *SAVE*, are not shown on the figure.

There is the place *now* which contains a token keeping the current time of the model. The transition *add* has a firing interval [1,1]. The place *now* is the input and output place of this transition. The transition *add* has no other input and output places. Firing of this transition removes the token with the value of the current time, waits for one time unit and returns the token with the value incremented by 1.

To model a timer *t,* the place *timer* is created. At the moment of creating a new process instance, a token with the value *(p,-1)* is put into the place *timer*, where *p* is a PId value of the instance. The token *(p,-1)* means that the timer is inactive. The token *(p,0)* means that the timer is active and its signal is in the queue of the process instance. A token of any other colour means that the timer has been set, but its value is greater than the current model time.

The transition *send* is enabled if the value *m* of the token *(p,m)* in the place *timer* is less than or equal to the value of the token at the place *now*. The firing interval of the transition *send* is *[0,0]*. This means that the transition has to fire as soon as it is enabled. At firing of the transition, the token *(p,m)* is replaced by the token *(p,0)* and the timer signal is put into the input queue of the process instance.

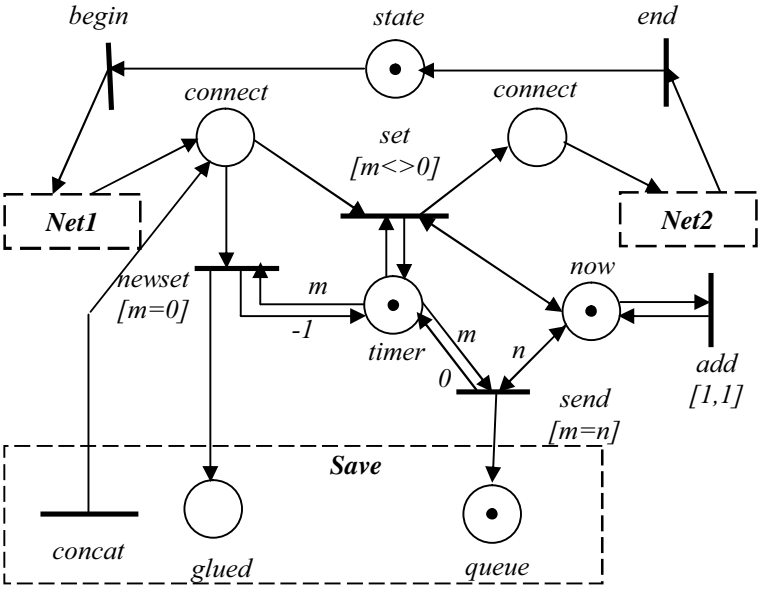


Fig. 3. Net presentation of a transition with the statement *SET(N,t)*

Consumption of a timer signal by a simple SDL transition of the process instance is modeled by firing of the corresponding net transition which removes the first element from the input queue and replaces the token *(p,0)* by the token *(p,-1)* in the place *timer*. Consumption of a timer signal by a complex SDL transition is similar but it is modeled by firing of the transition *begin* in the subnet corresponding to this SDL transition.

According to the SDL semantics, a new setting of the timer should cancel the previous setting. Moreover, if a timer signal corresponding to the previous setting is still in the input queue of the process instance, then it should be removed from the queue. In the modeling net, this is implemented in the following way. If the value of the second field of the token in the place *timer* is not equal to *0*, the transition *set* fires. As a result of firing, the token is replaced by a token corresponding to the new timer setting. Otherwise, the transition *newest* fires. It results in firing of the transitions of the subnet *SAVE* such that the timer signal is removed from the place *queue* and the token *(p,-1)* appears in the place *timer*. As a result, the transition *set* becomes enabled.

The RESET statement is modeled in a similar way.

Net size bound. Let us consider an upper bound of the resulting net. In order to take no account of modules and copy-places, we consider the size of the equivalent net which is not hierarchical. It is a result of module removing and gluing the copy-places of the same prototype. First, we consider the net modeling the process which contains descriptions of *var* variables including export and viewed variables, *par* parameters, *t* timers. Also the process has *m* input and output signal routes and *n* statements including *k* constructions *DECISION, SET*, *RESET*, *JOIN*, *SAVE* and *C* procedures

and macro calls. Modeling of the constructions *SAVE*, *SET* and *RESET* needs at most 7 places and 6 transitions for each of them.

Let *att* be *var+14+2*m+par+t*. The resulted net has at most *TN=(2*n+5+2*m+6*k)*(C+1)* transitions and *PN=(n+att+7*k)*(C+1)* places.

The net modeling the SDL specification consists of the nets representing the process definitions, channel-places, places *now* and *Pid,* additional transitions and places mapping the connection of the signal routes to the channel, at most *o*s,* where *o* is the number of different processes, *s* is the number of signals transmitted through the channel.

Thus, the net size bound is the sum of the net size bounds of all processes and *5*(o*(s+m)+1)+n* transitions and 6*(*m*p+1)* places, where *p* is the maximum number of different instances of one process.

4 The Tool SPV

The tool SPV is an integrated program environment for the design, verification, and simulation of net models of communication protocols. The tool consists of the following main components: translator, graphical editor, simulator, verifier, visualizer.

The translator performs automatic translation of SDL protocol specifications into net models.

The verifier allows one to examine the model properties using a model checking method.

In the multi-window graphic *editor,* the net model is represented as a tree of pages, on each of which the net is depicted in the form of a marked oriented graph. The editor provides means to control and modify the model.

The simulator is integrated with the editor and visualizes functioning of HTT-nets, as well as allows us to log a simulation session. In the course of simulation, the user has access to the following information:

- The current state of the HTT-net is graphically displayed as a change in place marking and in current temporal characteristics of transitions.
- The set of events that can occur at the next simulation step, and the user can control the choice of events.
- The sequence of events (trace) of the current simulation session which can be stored in the database for subsequent analysis or repetition of the saved simulation session from an arbitrary intermediate state.

The simulator can work in both step-by-step and automatic modes. In the step-by-step mode, full control of the simulation process is given to the user. In the automatic mode, the choice of a possible event at the next simulation step is performed in a random fashion. The tool provides means to facilitate model analysis and debugging, such as specifying the conditions for which the simulation process is suspended. An important feature of the tool is the capability of interaction of the editor/simulator with a text editor. When doing this, the user can easily compare an individual component or a model fragment obtained as a result of translation to the corresponding parts of the source specification.

The HTT-net is fed as the verifier input along with the formula of mu-calculus that expresses the network property to be checked. The verifier consists of a model constructor and an analyzer. Based on the net description, the model constructor generates a description of the net accessibility graph and places which, along with the mu-formula, are fed to the analyzer input that performs the procedure of model checking. Details of the verifier description can be found in [14].

The *visualizer* constructs a special model that operates with terms natural for the input protocol. In this model, instead of places and transitions, receivers/transmitters and connectors are used to perform the transmission of data packages. Besides, any other objects can be created that are related to the peculiarities of this or that protocol. This model is called visualization because its objects are represented in the form of graphic elements (ovals, rectangles, lines, pictures, etc.), and a change in a model state is reflected by the change of the elements and their graphic attributes (shape, filling, line thickness, etc.). The visualization has subordinate character, its entire behavior is determined by the behavior of the basic net model in the course of its simulation but the behavior of the protocol being modeled is reflected in the visualization in a graphic form.

The net model is represented in the standard language PNML [19] which provides the possibility for model exchange between the tool components and with third party tools.

The SPV tool is implemented using the object-oriented language Python which provides its portability and extensibility.

5 Case Studies

The SPV tool was used to perform checking of the most important properties of reliability for the following 3 communication protocols: RE-protocol [4], ATMR-protocol [10] and i-protocol [5].

The first case study was held for RE-protocol – a protocol which is used in ring networks. In this protocol each station sends a frame filled with data to its downstream neighbor. The frame has two special service bits (R and E) which are used to control correctness of network functioning. This control is performed by a special kind of a station in the ring called a monitor. According to the values of R and E bits, the monitor decides which action will perform: reinitialize the ring or do nothing.

The RE-protocol was studied for cases of reliable and unreliable medium with up to 3 stations and a monitor. It was checked to be satisfying the following properties:

1. *Presence of deadlocks.* This property can be identified at the stage of reachability graph construction or with mu-formula $\neg$<to>*true,* where *true* corresponds to all states in the model. RE-protocol in the cases studied has no deadlocks.
2. *Safety.* This property can be described with mu-formula $\mu X.$<to> (*received* $\vee X$), and it holds if it is possible to receive all sent messages. The predicate *received* corresponds to the states in the model where a message is received. This property holds for RE-protocol in all cases studied.

3. *Extended safety.* This property is described by formula *sent* → μX. (*received* ∨ [to] X). It means "all sent messages are received". The predicate *sent* corresponds to the states in the model where a message is sent. This property holds only for cases with reliable medium.
4. *Repeating messages.* We found that if the medium is unreliable, the message sent by one station to another may eventually come more than one time to its recipient. This property can be described by the following mu-formula: *received* ∧ <to>(μX.(*received* ∨ (¬ *sent* ∧ <to>X))). Our program verification confirmed that in case of unreliable medium the repetition of messages appears. This does not happen for models with the reliable medium. To avoid message repetition, we introduced some changes in the protocol specification. We also verified the corrected version of RE-protocol and found that message repetition does not appear.

The second protocol to verify was ATMR-protocol. It is also a ring-protocol and it is similar to RE-protocol in its basics. However, there is no special station to control the correctness of network functioning. ATMR-protocol is a high-speed protocol and it has no unreliable medium handler. It is supposed that high-level protocols should take care of re-sending messages. That is why we studied ATMR-protocol for cases with reliable medium only. We verified this protocol with 3 stations. We checked the same properties as for RE-protocol and found that ATMR-protocol has no deadlocks, satisfies safety and extended safety properties and message repetition does not appear. In contrast to the RE-protocol, the absence of duplicated messages confirms the effectiveness of the ATMR-protocol.

i-protocol is a part of the GNU UUPC package of Free Software Foundation. The protocol is used to transmit data via communication links. The *i*-protocol is a modification of the sliding window protocol. This protocol resends a minimum number of data packages, as compared to other versions of the sliding window protocol. A *live-lock* error, arising during the protocol functioning, is described in [5]. Transmitter resends the same data package. Receiver ignores them. Here, receiver does not send any data package because receiving of a new package causes timer reset. Thus, permanent activity of the protocol is registered, but the user does not send a new package and receiver does not receive any package. In the *i*-protocol environment, a special step sequence has been reproduced during the simulation using the tool SPV. This sequence causes the Kaivola error [12] in the Stenning protocol. The Kaivola error appears as a result of incorrect processing of receiver acknowledgments. As distinct from the Stenning protocol, the acknowledgments are correctly received by transmitter during the processing of this step sequence. However, the reproduction of this step sequence causes *live-lock* in the *i*-protocol. In addition, the experiments show that the same situation appears in the *i*-protocol as a result of any event sequence which leads to the state when the transmitter window is filled in and no acknowledgments are received.

6 Conclusion

This paper introduces a new modification of CPN called HTT-nets which has the following advantages:

- HTT-nets are semi-safe;
- Time concept in HTT-nets is simpler, as compared with CPN;
- Expressive power of HTT-nets allows us to model SDL specifications using dynamic constructs.

To the best of our knowledge, our translation method is the first such that net models of SDL specifications are semi-safe and the net size bounds are linear for communication protocols in many cases. Semi-safeness of HTT-nets simplifies simulation and analyses of communication protocols.

The tool SPV here described supports simulation, analysis and verification of these net models. Experiments with RE-protocol, ATMR-protocol and *i*-protocol have been successfully performed. The ineffectiveness of RE-protocol has been proven using the model-checking method. Also a modified effective version of the protocol has been verified. New cases of a live-lock for *i*-protocol, as compared with the cases in [5], have been found. Combination of different means for the analysis, simulation and verification of HTT-nets plays an important role, as demonstrated by experiments with RE-protocol, ATMR protocol and the *i*-protocol.

It is supposed to apply the tool SPV to verification of protocols with feature and service interaction in telecommunication systems [13].

References

1. Aalto, A., Husberg, N., Varpaaniemi, K.: Automatic formal model generation and analysis of SDL. In: Reed, R., Reed, J. (eds.) SDL 2003. LNCS, vol. 2708, pp. 285–299. Springer, Heidelberg (2003)
2. Berthomieu, B., Diaz, M.: Modelling and verification of time dependent systems using time Petri nets. IEEE Transactions on Software Eng. 17(3), 259–273 (1991)
3. Churina, T.G., Argirov, V.S.: Modeling SDL specifications using modified HTT-nets. Institute of Informatics Systems, Russian Academy of Sciences, Novosibirsk, Preprint vol. 124, pp. 1–62, 2005 (in Russian)
4. Cohen, R., Segall, A.: An efficient reliable ring protocol. IEEE Transactions on Communications 39(11), 1616–1624 (1991)
5. Dong, Y., Du, X., Ramakrishna, Y.S, Ramakrishnan, C.R., Ramakrishnan, I.V., Smolka, S.A., Sokolsky, O., Stark, E.W, Warren, D.S.: Fighting livelock in the i-protocol: a comparative study of verification tools. In: ETAPS 1999 and TACAS 1999. LNCS, vol. 1579, pp. 74–88. Springer, Heidelberg (1999)
6. Fisher, J., Dimitrov, E.: Verification of SDL'92 specifications using extended Petri nets. In: Proc. IFIP 15th Intern. Symp. on Protocol Specification, Testing and Verification, Warsaw, Poland, pp. 455–458 (1995)
7. Fleischhack, H., Grahlmann, B.: A compositional Petri net semantics for SDL. In: Desel, J., Silva, M. (eds.) ICATPN 1998. LNCS, vol. 1420, pp. 144–164. Springer, Heidelberg (1998)
8. Grahlmann, B.: Combining Finite Automata. In: Steffen, B. (ed.) ETAPS 1998 and TACAS 1998. LNCS, vol. 1384, pp. 102–117. Springer, Heidelberg (1998)
9. Husberg, N., Manner, T.: Emma: Developing an Industrial Reachability Analyser for SDL. In: Wing, J.M., Woodcock, J.C.P., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 642–661. Springer, Heidelberg (1999)

10. Imai, K., Ito, T., Kasahara, H., Morita, N.: ATMR: Asynchronous transfer mode ring protocol. Computer Networks and ISDN Systems 26, 785–798 (1994)
11. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 1,2,3. Springer, Heidelberg (1997)
12. Kaivola, R.: Using compositional preorders in the verification of sliding window protocol. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 48–59. Springer, Heidelberg (1997)
13. Keck, D.O., Kuehn, P.J.: The feature and service interaction problem in telecommunications systems: a survey. IEEE Trans. on Software Eng. 24(10), 779–796 (1998)
14. Kozura, V.E., Nepomniaschy, V.A., Novikov, R.M.: Verification of distributed systems modeled by high-level Petri nets. In: Proc. Intern. Conf. on Parallel Computing in Electrical Engineering, Warsaw, Poland, pp. 61–66. IEEE Computer Society Press, Los Alamitos (2002)
15. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured Petri nets. Internat. J. Software Tools for Technology Transfer 2(2), 98–132 (1998)
16. Peng, H., Tahar, S., Khendek, F.: SPIN vs. VIS: A case study on the formal verification of the ATMR protocol. In: Proc. 3rd Intern. Conf. on Formal Engineering Methods, pp. 79–87. IEEE Computer Society Press, Los Alamitos (2000)
17. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for editing, simulating and analysing coloured Petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003)
18. Specification and Description Language (SDL). Recommendation, Z.100, CCITT (1992)
19. Weber, M., Kindler, E.: The Petri net markup language, Petri Net Technology for Communication Based Systems. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 124–144. Springer, Heidelberg (2003)

Symmetry of Information and Nonuniform Lower Bounds

Sylvain Perifel

LIP*,
École Normale Supérieure de Lyon
Sylvain.Perifel@ens-lyon.fr

Abstract. In the first part we provide an elementary proof of the result of Homer and Mocas [3] that for all constant c, the class EXP is not included in P/n^c. The proof is based on a simple diagonalization, whereas it uses resource-bounded Kolmogorov complexity in [3].

In the second part, we investigate links between resource-bounded Kolmogorov complexity and nonuniform classes in computational complexity. Assuming a weak version of polynomial-time symmetry of information, we show that exponential-time problems do not have polynomial-size circuits (in symbols, $\mathsf{EXP} \not\subset \mathsf{P/poly}$).

Keywords: computational complexity, nonuniform lower bounds, resource-bounded Kolmogorov complexity, symmetry of information.

1 Introduction

Whereas some uniform lower bounds have been proved long ago thanks to hierarchy theorems, little progress have been made towards longstanding open problems concerning nonuniform lower bounds in complexity theory. This observation is explained by the lack of proof techniques for these nonrelativizing questions. In particular, a simple diagonalization is not suitable for the main question of whether exponential-time problems have polynomial-size circuits. Yet the advantage of diagonalization is its simplicity. In this paper, we show that a hypothesis of resource-bounded Kolmogorov complexity gives diagonalization enough power to settle these questions.

We are interested in the question of whether the class EXP of problems decided in exponential time has polynomial-size circuits (in symbols, whether $\mathsf{EXP} \subset \mathsf{P/poly}$). As mentioned above, the separation $\mathsf{EXP} \neq \mathsf{P}$ is well-known but this nonuniform counterpart is still open. On this problem, two approaches have yielded significant results.

The first approach was to find the smallest uniform class provably not contained in $\mathsf{P/poly}$. In this direction, Kannan [4] proved that $\mathsf{NEXP}^{\mathsf{NP}}$ does not have polynomial-size circuits, and afterwards Schöning [12] gave a simplified proof that $\mathsf{EXPSPACE}$ does not have polynomial-size circuits. Here, we see that

* UMR 5668 ENS Lyon, CNRS, UCBL, INRIA.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 315–327, 2007.

performing a diagonalization out of $\mathsf{P/poly}$ requires more than exponential time. Later, the second approach was to obtain the best nonuniform lower bound for EXP problems. Homer and Mocas [3] showed that EXP does not have circuits of size n^c for any fixed constant c.

As a first step toward our main theorem, another proof of this last result is provided in the first part of the present paper (Proposition 2). This is an elementary proof consisting in a mere diagonalization (whereas the original proof of [3] makes use of resource-bounded Kolmogorov complexity) which is included here because it familiarizes with the proof of the main result of the paper, and also because this easy proof has never been published to the author's knowledge (though it uses techniques very similar to [12]). As a corollary, included here as another illustration of this method, we obtain a nonuniform lower bound on PP problems (Proposition 4). Unfortunately this is much weaker than the result of Vinodchandran [13].

In the second and main part, we show that an assumption of resource-bounded Kolmogorov complexity enables to combine both approaches described above. Namely, if a weak version of polynomial-time symmetry of information holds true, then $\mathsf{EXP} \not\subset \mathsf{P/poly}$ (Theorem 2). This result therefore relates two major open questions. The proof once again consists in a simple diagonalization.

Symmetry of information is a beautiful theorem in Kolmogorov complexity and one of its versions can roughly be stated as follows: if x and y are two words, x contains the same quantity of information on y as y on x. This theorem is due to Levin [14] and Kolmogorov [6].

When requiring polynomial time bounds on the computations, however, the similar property, called polynomial-time symmetry of information, is a challenging open problem in resource-bounded Kolmogorov complexity. This problem has already been related to computational complexity by at least two results. First, Longpré and Watanabe [9] show that if $\mathsf{P} = \mathsf{NP}$ then polynomial-time symmetry of information holds. Second, more recently and closer to our present preoccupations, Lee and Romashchenko [7] show that if polynomial-time symmetry of information holds, then $\mathsf{EXP} \neq \mathsf{BPP}$. A longer discussion on symmetry of information, inspired by the introduction of [7], is provided in Section 4.

Here, assuming a weak version of polynomial-time symmetry of information (see Section 4) we prove that $\mathsf{EXP} \not\subset \mathsf{P/poly}$ (a stronger conclusion than in [7] since $\mathsf{BPP} \subset \mathsf{P/poly}$, see [1]). As we shall see, symmetry of information enables to divide advices into small blocks on which diagonalization can be performed in EXP.

All these results teach us that polynomial-time symmetry of information, even in its weakest forms, is a hard but central question to study. Indeed, if it holds, then EXP does not have polynomial-size circuits, else $\mathsf{P} \neq \mathsf{NP}$. In both cases, a fundamental question in complexity theory would find an answer.

Organization of the paper. Section 2 is devoted to definitions and notations in computational complexity and resource-bounded Kolmogorov complexity. Section 3 consists of another proof of the result of [3] that exponential-time problems do not have circuits of any fixed polynomial size n^c. A simple corollary is also shown there, namely a nonuniform lower bound on PP problems.

Section 4 precisely state the hypothesis of polynomial-time symmetry of information as well as some simple results about this. Finally, Section 5 proves the main result, namely that polynomial-time symmetry of information implies that exponential-time problems do not have polynomial-size circuits.

2 Preliminaries

For references on computational complexity we recommend the book [2]. For Kolmogorov complexity, we refer to [8]. The notions used in this paper are standard, though stated from the unifying point of view of universal Turing machines.

Universal machines. If M is a Turing machine, for simplicity we will assume that it is encoded in binary and denote by M this encoding. Therefore M is also seen as a program. We restrict ourselves to two-tape Turing machines; the following result on a universal Turing machine is then well-known.

Proposition 1. *There exists a universal Turing machine $\mathcal{U}$, with two tapes, which, on input (M, x), simulates the two-tape machine M on input x. There is a constant $c > 0$ depending only on the machine M such that the simulation of t steps of $M(x)$ takes ct steps of $\mathcal{U}$.*

Such a universal Turing machine $\mathcal{U}$ is fixed in the remainder of the paper. The machine M simulated by $\mathcal{U}$ will also be called the *program* of $\mathcal{U}$. For instance, we will say that the program M decides the language A if for all x, the computation $\mathcal{U}(M, x)$ halts, and it accepts iff $x \in A$.

Complexity classes. If $t : \mathbf{N} \to \mathbf{N}$ is a function, the class $\mathsf{DTIME}(t(n))$ is the set of languages A recognized in time $O(t(n))$. More precisely, $A \in \mathsf{DTIME}(t(n))$ if there exist a constant $c > 0$ and a fixed program $M \in \{0,1\}^*$ such that for all word x, the computation $\mathcal{U}(M, x)$ stops before $ct(|x|)$ steps and it accepts if and only if $x \in A$. We call EXP the class $\cup_{k \geq 0} \mathsf{DTIME}(2^{n^k})$.

Now, nonuniform computation is defined via advices as introduced by Karp and Lipton [5]. The advice class $\mathsf{DTIME}(t(n))/a(n)$ is the set of languages A such that there exist a program M, a constant $c > 0$ and a family (a_n) of advices (that is to say, words) satisfying:

1. $|a_n| \leq a(n)$;
2. $\mathcal{U}(M, x, a_{|x|})$ stops in less than $ct(|x| + a(|x|))$ steps;
3. $\mathcal{U}(M, x, a_{|x|})$ accepts iff $x \in A$.

The nonuniform class $\mathsf{P/poly}$ is defined as $\cup_{k \geq 0} \mathsf{DTIME}(n^k)/n^k$ (i.e. polynomial working time and polynomial-size advice) and is easily shown to be the set of languages recognized by a family of polynomial-size boolean circuits. Similarly, $\mathsf{EXP/poly}$ is the class $\cup_{k \geq 0} \mathsf{DTIME}(2^{n^k})/n^k$ (i.e. exponential working time and polynomial-size advice). By this definition, it is easy to see that

$$\mathsf{EXP} \subset \mathsf{P/poly} \iff \mathsf{EXP/poly} = \mathsf{P/poly}.$$

Another complexity measure is the space needed to decide a language. Space complexity counts the number of cells used by the machine. The class $\mathsf{DSPACE}(s(n))$ is the set of languages A recognized in space $O(s(n))$, and advice classes are defined accordingly. The class PSPACE is $\cup_{k\geq 0}\mathsf{DSPACE}(n^k)$.

In this paper, we shall also quickly meet the complexity class PP. This is the set of languages A such that there exist a language $B \in \mathsf{P}$ and a polynomial $p(n)$ satisfying $x \in A \iff \#\{y \in \{0,1\}^{p(|x|)} \ : \ (x,y) \in B\} \geq 2^{p(|x|)-1}$.

Resource-bounded Kolmogorov complexity. For two words x, y and an integer t, we denote by $C^t(x|y)$ the time t bounded Kolmogorov complexity of x conditional to y, that is, the size of a shortest program M which, when run on the universal Turing machine $\mathcal{U}$ on input y, outputs x in time $\leq t$. For a Turing machine M (and in particular for $\mathcal{U}$), we denote by $M^t(x)$ the word written on the output tape of the machine M after t steps of computation on input x. Thus in symbols we have

$$C^t(x|y) = \min\{k \ : \ \exists M \text{ of size } k \text{ such that } \mathcal{U}^t(M,y) = x\}.$$

We will use the notation $C^t(x)$ for $C^t(x|\epsilon)$, where ϵ is the empty word.

Advice and programs. For a fixed word length n, the words $x \in \{0,1\}^n$ of size n are lexicographically ordered and the i-th one is called $x^{(i)}$ (for $1 \leq i \leq 2^n$). Let A be a language. The *characteristic string* of $A^{=n}$ is the word $\chi \in \{0,1\}^{2^n}$ defined by $\chi_i = 1$ iff $x^{(i)} \in A$. We will often consider programs that output characteristic strings rather than programs that decide languages. We rely on the following obvious lemma.

Lemma 1. *If A is a language in $\mathsf{DTIME}(t(n))/a(n)$ (where $t(n) \geq n$), then there exist constants $\alpha, k > 0$ and a family (M_n) of programs satisfying:*

1. *$|M_n| \leq k + a(n)$;*
2. *for $1 \leq i \leq 2^n$ in binary, $\mathcal{U}(M_n, i)$ outputs the i first bits of the characteristic string χ of $A^{=n}$ in time $\alpha i t(n + a(n))$.*

Proof. Let M be a $\mathsf{DTIME}(t(n))$ machine deciding A with advice of size $a(n)$. The program M_n merely enumerates the i first words x of size n and simulates $M(x)$: M_n is therefore composed of the code of M, of an enumeration routine for the i first words of size n and of the advice for the length n. □

3 Diagonalizing Out of n^c Advice Length

We provide another proof of the following proposition of Homer and Mocas [3]. The initial proof of [3] makes use of resource-bounded Kolmogorov complexity. Here it consists in a usual diagonalization, similar to the proof of Schöning [12] that EXPSPACE does not have polynomial-size circuits (see [2, Th. 5.6]): at each step of the diagonalization process, we eliminate half of the possible programs. This easy proof can be considered as folklore; still we include it in the

present paper since it seems unpublished so far, and because it introduces some techniques used in the proof of the main result of this paper. Furthermore, this method yields a nonuniform lower bound on PP problems as a corollary. This lower bound is rather weak and a better one is already known; we include it here only as another application of the method.

Proposition 2. *For all constants $c_1, c_2 \geq 1$, there is a sparse language A in* $\mathsf{DTIME}(2^{O(n^{1+c_1c_2})})$ *but not in* $\mathsf{DTIME}(2^{O(n^{c_1})})/n^{c_2}$.

Proof. The idea of the proof is to diagonalize against all programs of size n^{c_2} thanks to a language that eliminates half of them at each step.

Let us define $A^{=n}$ for all n, therefore fix n. Recall that $x^{(1)} < x^{(2)} < \ldots < x^{(2^n)}$ are the words of $\{0,1\}^n$ sorted in lexicographic order. We will diagonalize over the programs M of size at most $n + n^{c_2}$ (of which there are $2^{n+n^{c_2}+1} - 1$), and the universal machine $\mathcal{U}$ will be simulated for $t(n) = 2^{n^{1+c_1c_2}}$ steps. The set $A^{=n}$ is defined word by word as follows:

$$x^{(1)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \text{ of size } \leq n + n^{c_2},\\ \text{the first bit of } \mathcal{U}^{t(n)}(M) \text{ is } 0,\end{array}$$

that is, at least half of the programs give the wrong answer for $x^{(1)}$. Let V_1 be the set of programs M giving the right answer for $x^{(1)}$, i.e. such that the first bit of $\mathcal{U}^{t(n)}(M)$ corresponds to "$x^{(1)} \in A$". Hence $|V_1| < 2^{n+n^{c_2}}$ (less than half of the programs of size $\leq n + n^{c_2}$ remain). We then go on with $x^{(2)}$:

$$x^{(2)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \in V_1,\\ \text{the second bit of } \mathcal{U}^{t(n)}(M) \text{ is } 0,\end{array}$$

that is, among the programs that were right for $x^{(1)}$, at least half make a mistake for $x^{(2)}$. Let V_2 be the set of programs $M \in V_1$ giving the right answer for $x^{(2)}$. We go on like this:

$$x^{(i)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \in V_{i-1},\\ \text{the } i\text{-th bit of } \mathcal{U}^{t(n)}(M) \text{ is } 0\end{array}$$

until V_i is empty. Call k the first i such that $V_i = \emptyset$. We decide arbitrarily that $x^{(j)} \notin A^{=n}$ for $j > k$. Note that $k \leq n + n^{c_2} + 1$ because $|V_i|$ is halved at each step, therefore A is sparse.

If $A \in \mathsf{DTIME}(2^{O(n^{c_1})})/n^{c_2}$, then by Lemma 1 there would be a constant k and a family (M_n) of programs of size $\leq k+n^{c_2}$ writing down the characteristic string of $A^{=n}$ in time $\alpha(n+n^{c_2}+1)2^{O(n^{c_1c_2})} \leq 2^{\beta n^{c_1c_2}}$ for some β. This is not possible as soon as $n \geq k$ and $t(n) > 2^{\beta n^{c_1c_2}}$ since all programs of size $n+n^{c_2}$ must make a mistake on some input of size n. Therefore $A \notin \mathsf{DTIME}(2^{O(n^{c_1})})/n^{c_2}$.

Now, in order to decide if $x^{(i)} \in A$ it is enough to decide if $x^{(j)} \in A$ for all $j \leq i$. This is done in the order $j = 1, \ldots, i$ because we need the answer of j for $j+1$. For $x^{(j)}$ we proceed as follows: we enumerate all the programs M of size $\leq n + n^{c_2}$, compute $\mathcal{U}^{t(n)}(M)$ by simulating $\mathcal{U}$ for $t(n)$ steps, we test

whether $M \in V_{j-1}$ (this is done by comparing for each $k < j$ the k-th bit of $\mathcal{U}^{t(n)}(M)$ with the already computed value of "$x^{(k)} \in A$"), and count how many $M \in V_{j-1}$ produce an output whose j-th bit is 0. If there are more than half such M, then $x^{(j)} \in A$, otherwise $x^{(j)} \notin A$. The overall running time of this algorithm is $(n + n^{c_2})2^{O(n^{c_2})}t(n)$, thus $A \in \mathsf{DTIME}(2^{O(n^{1+c_1c_2})})$. □

The same proof also works for space complexity.

Proposition 3. *For all constants $c_1, c_2 \geq 1$, there is a sparse language A in $\mathsf{DSPACE}(n^{1+c_1c_2})$ but not in $\mathsf{DSPACE}(n^{c_1})/n^{c_2}$.*

The following corollary is now immediate.

Corollary 1. *For every constant $c > 0$, $\mathsf{EXP} \not\subset (\mathsf{P}/n^c)$ and $\mathsf{PSPACE} \not\subset (\cup_k \mathsf{DSPACE}(\log^k n)/n^c)$.*

Remark 1. The original result of Homer and Mocas [3] has since been improved by Ronneburger [11, Th. 5.21] in the following way: there is a language $R \in \mathsf{EXP}$ such that for all k, there exists a language $L \in \mathsf{EXP}$ which is not truth-table reducible to R in time 2^{n^k} with n^k bits of advice.

The construction of the language A of Proposition 2 enables us to prove the following nonuniform lower bound for PP problems. As already mentioned, this is a much weaker result than Vinodchandran [13] showing that PP does not have circuits of size n^k for any fixed k.

Proposition 4. *For any fixed $k > 0$, $\mathsf{PP} \not\subset \mathsf{DTIME}(n^k)/(n - \log n)$.*

Proof. The idea relies on the following remark: in the proof of Proposition 2, if the simulation time of the machine $\mathcal{U}$ is polynomial, then deciding whether "for at least half of the programs $M \in V_{i-1}$, the i-th bit of $\mathcal{U}^{t(n)}(M)$ is 0" is a PP problem.

Let us now fill the details. Take $t(n) = n^{3+k}$ for the simulation time and diagonalize over programs of size $\leq n - (\log n)/2$ (of which there are less than $2^n/\sqrt{n}$). For conveniency, if $k \leq n+1-(\log n)/2$ and $b_1, \ldots, b_k$ are k bits, define

$$B(b_1, \ldots, b_k) = \{M |\ M \text{ is a program of size } \leq n - (\log n)/2 \text{ such that } \forall i \leq k, \text{ the } i\text{-th bit of } \mathcal{U}^{t(n)}(M) \text{ is } b_i\}.$$

Let us now define the following language C:

$$C = \{(b_1 \ldots b_{k+1}, 0^{m_k}) |\ \text{for at least half of the programs } M \in B(b_1, \ldots, b_k), \text{ the } (k+1)\text{-th bit of } \mathcal{U}^{t(n)}(M) \text{ is } b_{k+1}\}.$$

The second term 0^{m_k} of the couple in C is only a padding term, so that the length of the queries to C will be always the same. Since $k \leq n+1-(\log n)/2$, one can assume by choosing an appropriate encoding that the length of $(b_1 \ldots b_{k+1}, 0^{m_k})$ is always n.

Note that deciding whether $M \in B(b_1, \ldots, b_k)$ can be done in polynomial time (there are $k \leq n + 1 - (\log n)/2$ simulations to perform, each of which requires time $O(t(n))$). Therefore $C \in \mathsf{PP}$.

We can now define our main language A very similarly as in Proposition 2:

$$x^{(1)} \in A^{=n} \iff (0, 0^{m_1}) \in C,$$

that is, $x^{(1)} \in A$ if and only if at least half of the programs of size $\leq n - (\log n)/2$ reject $x^{(1)}$ (i.e., they make a mistake on $x^{(1)}$). Call $b_1 \in \{0, 1\}$ the right answer for $x^{(1)}$, that is, $b_1 = 1$ iff $x^{(1)} \in A^{=n}$. Then we go on with $x^{(2)}$:

$$x^{(2)} \in A^{=n} \iff (b_1 0, 0^{m_2}) \in C.$$

Once again, among the programs that were right for $x^{(1)}$, at least one half give the wrong answer for $x^{(2)}$. Going on like this for $n + 1 - (\log n)/2$ steps, all the programs are wrong on at least one word because every program has been diagonalized against. We decide arbitrarily that $x^{(i)} \notin A^{=n}$ for $i > n + 1 - (\log n)/2$. We thus have $A \notin \mathsf{DTIME}(n^{k+2})/(n - \log n)$.

The language A can then be recognized in time $O(n^2)$ with oracle access to C: it is enough to decide successively whether $x^{(1)} \in A$, $x^{(2)} \in A$, etc. These $n + 1 - (\log n)/2$ steps can be done thanks $n + 1 - (\log n)/2$ queries of size n to C (the time complexity of the algorithm is $O(n^2)$ because it has to ask $O(n)$ questions of size $O(n)$).

Suppose for a contradiction that $\mathsf{PP} \subset \mathsf{DTIME}(n^k)/(n - \log n)$. Then $C \in \mathsf{DTIME}(n^k)/(n - \log n)$ and the algorithm above only queries words of size n. Hence $A \in \mathsf{DTIME}(n^{k+2})/(n - \log n)$ which is a contradiction. □

4 Symmetry of Information

In this section we state the hypothesis of resource-bounded symmetry of information we will use. For the sake of completeness, we first state a version of symmetry of information for exponential time bounds. For a proof one can easily adapt the unbounded case, see for instance [8, Th. 2.8.2 p. 182].

Theorem 1. *There exist constants α, β such that for all words x, y and all time bound t, the following equality holds:*

$$C^t(x, y) \geq C^{t2^{\alpha(|x|+|y|)}}(x) + C^{t2^{\alpha(|x|+|y|)}}(y|x) - \beta \log(|x| + |y|).$$

Notice that the other inequality also holds up to a logarithmic factor and is much easier to show. Here we are only interested in the "hard part" of symmetry of information. This is an open question whether this inequality holds for polynomial time bounds, i.e. whether there exists a polynomial $q(n)$ such that $C^t(x, y) \geq C^{tq(|x|+|y|)}(x) + C^{tq(|x|+|y|)}(y|x) - \beta \log(|x| + |y|)$. However, if one-way functions exist (as is often believed), then polynomial-time symmetry of information does not hold, see [9]. This suggests that this version of polynomial-time symmetry is too strong.

One way to relax this hypothesis is to allow a larger error, replacing the $O(\log(|x|+|y|))$ error term by $\delta(|x|+|y|)$. This would imply that polynomial-time computable functions can be inverted in time $2^{O(\delta(n))}$. This does not seem completely impossible if $\delta(n)$ is large enough, for instance $\delta(n) = \epsilon n$. Note also that the hypothesis is true for $\delta(n) = n$, since trivially there exists a polynomial q such that $2C^t(x,y) \geq C^{tq(|x|+|y|)}(x) + C^{tq(|x|+|y|)}(y|x)$. All these remarks lead us to the following version of the hypothesis of polynomial-time symmetry of information. It is interesting to note that Lee and Romashchenko [7], in the introduction of their paper, already ask a very similar question: can we show that $(2-\epsilon)C(x,y) \geq C(x) + C(y|x)$ for some contant $\epsilon > 0$ when polynomial-time bounds are required?

(SI) There exist a constant $\alpha > 1/2$ and a polynomial q such that for all time bound t and all words x, y, z of size $|x|+|y|+|z| = n$,

$$C^t(x,y|z) \geq \alpha\Big(C^{tq(n)}(x|z) + C^{tq(n)}(y|x,z)\Big).$$

Note that we need time bounds $tq(n)$ in the right-hand side, instead of $q(t)$ in the usual settings of polynomial-time symmetry of information, in order to limit the growth of the time bound when iteratively applying (SI). It would be nice to rule this problem out and use the usual $q(t)$ time bound instead. Note finally that the hypothesis can be weakened, as mentioned in the following remark.

Remark 2. For our purpose, the following restrictions can further be applied on the hypothesis (SI):

1. we can require $|x| = |y|$ and $C^{tq(n)}(x|z) = C^{tq(n)}(y|x,z)$;
2. the time bound t can be taken $< 2^{n^2}$ and in this framework, the hypothesis can hold only for all but a finite number of words x and y;
3. the constant α can be replaced by the nonconstant term $1/2 + 1/\sqrt{\log(|x|+|y|)}$, which is closer to $1/2$.
4. the multiplicative time bound $q(n)$ can in fact be much larger than a polynomial: we could take $q(n) = 2^{2^{\sqrt{\log n}}}$, which is greater than $2^{\log^k n}$ for all k. All we need is a function q such that $q(n^{\log n})^{\log^2 n} < 2^{n^k}$ for some k.

Putting these points together, here is the weaker (but more complicated) hypothesis we obtain:

Let $f(n) = 2^{2^{\sqrt{\log n}}}$. For all n, all $t < f(n)$, all word z and all but finitely many words x, y of same length, if $|x|+|y|+|z| = n$ and $C^{tq(n)}(x|z) = C^{tq(n)}(y|x,z)$, then

$$C^t(x,y|z) \geq \left(\frac{1}{2} + \frac{1}{\sqrt{\log(|x|+|y|)}}\right)\Big(C^{tf(n)}(x|z) + C^{tf(n)}(y|x,z)\Big).$$

Note that relative to the oracle $O = \{(M, x, b) : M(x) = b \text{ in } \leq 2^{|x|} \text{ steps}\}$, (SI) is true. Furthermore, since our proofs below relativize and the conclusion does not, we obtain the following proposition (certainly also provable directly).

Proposition 5. *There exists an oracle A relative to which* (SI) *is true and an oracle B relative to which* (SI) *is false.*

We wish to iteratively apply (SI). In order to do that, we need to precise how we encode tuples. The main property we will need is that $(x_1, \ldots, x_{2n})$ should have the same encoding as $((x_1, \ldots, x_n), (x_{n+1}, \ldots, x_{2n}))$. This is achieved for instance by representing three symbols "zero", "one" and a delimiter # by 00, 11 and 01. Hence the size of the encoding of an n-tuple $(x_1, \ldots, x_n)$ will be $2(n-1) + 2(|x_1| + \ldots + |x_n|)$. Of course, much shorter encodings could also be chosen, but it would not help in this paper.

Lemma 2. *Suppose* (SI) *holds and take a corresponding polynomial q. Let t be a time bound, $u_1, \ldots, u_n$ be words of size s and z be another word of arbitrary size. We define $m = ns + |z|$ the size of all these words. Suppose there exists a constant k such that for all $j \leq n$, we have $C^{q(m)^{\log n} t}(u_j | u_1, \ldots, u_{j-1}, z) \geq k$.*

Then $C^t(u_1, \ldots, u_n | z) \geq (2\alpha)^{\lfloor \log n \rfloor} k$.

Proof. Fix a sequence of words $(u_i)_{i \geq 1}$ of size s. Let us first show the result when n is a power of 2. We show by induction on n (only for powers of 2) the following hypothesis: for every time bound t, every word z and all $m \geq ns + |z|$, if for all $j \leq n$, $C^{q(m)^{\log n} t}(u_j | u_1, \ldots, u_{j-1}, z) \geq k$ then $C^t(u_1, \ldots, u_n | z) \geq (2\alpha)^{\log n} k$.

This is clear for $n = 1$. For $n > 1$, take t, z and $m \geq ns + |z|$. By (SI),

$$C^t(u_1, \ldots, u_n | z) \geq \alpha \Big(C^{tq(m)}(u_1, \ldots, u_{n/2} | z) + C^{tq(m)}(u_{n/2+1}, \ldots, u_n | u_1, \ldots, u_{n/2}, z) \Big).$$

By induction hypothesis at rank $n/2$, for the time bound $tq(m)$ and where for the last term we take as new z the word $u_1, \ldots, u_{n/2}, z$, the right-hand side is at least $\alpha((2\alpha)^{\log(n/2)} k + (2\alpha)^{\log(n/2)} k) = (2\alpha)^{\log n} k$.

Now, if n is not a power of 2, let p be the largest power of 2 less than n. Then $C^t(u_1, \ldots, u_n | z) \geq C^t(u_1, \ldots, u_p | z) \geq (2\alpha)^{\log p} k = (2\alpha)^{\lfloor \log n \rfloor} k$. □

We now establish links between the Kolmogorov complexity of a characteristic string and the length of the advice.

Lemma 3. *Let A be a language and $\chi^{(n)}$ the characteristic string of $A^{=n}$ (i.e. $\chi^{(n)}_i = 1$ iff $x^{(i)} \in A^{=n}$). We denote by $\chi^{(n)}[1..i]$ the string consisting of the i first bits of $\chi^{(n)}$. Let $r(n)$ be a function and suppose that there exists an unbounded function $s(n) \geq 0$ such that for all constant $\alpha > 0$, there exist infinitely many n and $1 \leq i \leq 2^n$ satisfying $C^{\alpha i r(n + a(n))}(\chi^{(n)}[1..i]) > s(n) + a(n)$.*

Then $A \notin \mathsf{DTIME}(r(n))/a(n)$.

Proof. Suppose that $A \in \mathsf{DTIME}(r(n))/a(n)$. Then there exist a fixed program M together with an advice function $c(n)$ of size $\leq a(n)$, such that for all $x \in \{0,1\}^n$, $\mathcal{U}(M, x, c(n))$ works in time $O(r(n+a(n)))$ and accepts iff $x \in A^{=n}$. By enumerating the first i words of size n in lexicographic order and simulating the program M on each of them, there is another program N with advice $c(n)$ that enumerates $\chi^{(n)}[1..i]$ in time $O(ir(n+a(n)))$. Hence there exists a constant $\alpha > 0$ such that for all n and for all $i \leq 2^n$, $C^{\alpha ir(n+a(n))}(\chi^{(n)}[1..i]) \leq |N| + a(n)$. □

5 Diagonalizing Out of Polynomial Advice Length

We are now interested in diagonalizing over all polynomial advices, not just of size n^c for some fixed c. We will use the hypothesis of symmetry of information above. Here the main difficulty is to diagonalize over all advices without enumerating them, otherwise we would go outside of EXP. The idea is simple:

- Two different "useful" and "independent" parts of size k_1 and k_2 of an advice "must" carry roughly $k_1 + k_2$ bits of information.
- We therefore decompose the advice in small blocks (of size $O(n)$) and diagonalize over the blocks instead of the whole advice, while making sure that these blocks are "independent". The hypothesis (SI) then enables us to "glue" these blocks together.

This can also be seen as efficiently finding a string of high Kolmogorov complexity and Lemma 2 helps us do that.

Theorem 2. *If* (SI) *holds true, then* $\mathsf{EXP} \not\subset \mathsf{P/poly}$.

Proof. Suppose (SI) holds true: this gives a corresponding polynomial q. We diagonalize over programs ("blocks") M of length $\leq n-1$ and simulate the universal machine $\mathcal{U}$ for $t(n) = q(n^{1+\log n})^{\log^2 n} n^{2+\log n+\log^3 n}$ steps. Define the language A as follows, by length as in the proof of Proposition 2. The n first steps of the definition are the same as for Proposition 2; the difference occurs only after, when we reuse the initial segment of $A^{=n}$ in our simulation. So we define for $i \leq n$:

$$x^{(i)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \in V_{i-1}^{(0)},\\ \text{the } i\text{-th bit of } \mathcal{U}^{t(n)}(M) \text{ is } 0,\end{array}$$

where $V_{i-1}^{(0)}$ is the set of the remaining programs of size $\leq n-1$ which give the right answer for $x^{(i-1)}$. Remark that $V_n^{(0)} = \emptyset$ since all of the $2^n - 1$ programs M have been eliminated.

Call $u^{(1)}$ the characteristic string of the initial segment of size n of $A^{=n}$ defined above: thus $|u^{(1)}| = n$ and for $i \leq n$, $u_i^{(1)} = 1$ if and only if $x^{(i)} \in A^{=n}$. We now define the next segment of size n of $A^{=n}$.

$$x^{(n+1)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \text{ of size } \leq n-1,\\ \text{the first bit of } \mathcal{U}^{t(n)}(M, u^{(1)}) \text{ is } 0,\end{array}$$

that is, at least half of the programs M of length $\leq n-1$ give the wrong answer for $x^{(n+1)}$ even with the string $u^{(1)}$ as advice. Let $V_1^{(1)}$ be the set of programs M giving the right answer for $x^{(n+1)}$, i.e. such that the first bit of $\mathcal{U}^{t(n)}(M, u^{(1)})$ corresponds to "$x^{(n+1)} \in A$". Hence $|V_1^{(1)}| < 2^{n-1}$. We then go on like this:

$$x^{(n+i)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \in V_{i-1}^{(1)},\\ \text{the } i\text{-th bit of } \mathcal{U}^{t(n)}(M, u^{(1)}) \text{ is } 0.\end{array}$$

Call $u^{(2)}$ the characteristic string of the second segment of size n of $A^{=n}$ defined above: thus $|u^{(2)}| = n$ and $u_i^{(2)} = 1$ if and only if $x^{(n+i)} \in A^{=n}$. We define the third segment of size n of $A^{=n}$ analogously:

$$x^{(2n+1)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \text{ of size } \leq n-1,\\ \text{the first bit of } \mathcal{U}^{t(n)}(M, u^{(1)}, u^{(2)}) \text{ is } 0,\end{array}$$

etc. Going on like this we have (for the $(j+1)$-th segment):

$$x^{(jn+i)} \in A^{=n} \iff \begin{array}{l}\text{for at least half of the programs } M \in V_{i-1}^{(j)},\\ \text{the } i\text{-th bit of } \mathcal{U}^{t(n)}(M, u^{(1)}, u^{(2)}, \ldots, u^{(j)}) \text{ is } 0.\end{array}$$

We stop when $j = n^{\log n}$ and decide arbitrarily that $x^{(k)} \notin A$ for $k > n \times n^{\log n}$.

Let us first show that $A \notin \mathsf{P/poly}$. Note that at each step of the definition of A, we have $C^{t(n)}(u^{(j)}|u^{(1)} \cdots u^{(j-1)}) \geq n$ because no program of length $\leq n-1$ writes $u^{(j)}$ in time $\leq t(n)$ on input $u^{(1)} \cdots u^{(j-1)}$. Since (SI) holds and by definition of $t(n)$, Lemma 2 asserts that for $i = n \times n^{\log n}$, $C^{i.n^{1+\log^3 n}}(\chi^{(n)}[1..i]) \geq (2\alpha)^{\log^2 n} n = n^{1+\log(2\alpha)\log n}$ for large enough n.

Hence by Lemma 3, if we let $a(n) = n^{\log(2\alpha)\log n} - n$ and $r(n) = n^{\log n}$, we have $A \notin \mathsf{DTIME}(r(n))/a(n)$. In particular, $A \notin \mathsf{P/poly}$.

It is straightforward to see that $A \in \mathsf{EXP}$, and the theorem follows. □

Remark 3. The same proof also works for space complexity if we assume a corresponding version of symmetry of information for polylogarithmic space bounded Kolmogorov complexity. That is, under such an assumption we can prove that $\mathsf{PSPACE} \not\subset (\cup_k \mathsf{DSPACE}(\log^k n)/\mathsf{poly})$.

Let us now give a consequence of symmetry of information on randomized algorithms. The following theorem of Nisan and Wigderson [10] will be useful for our purpose. By approximating a problem we mean an algorithm that is right on all but a fraction $1/f(n)$ of the inputs, where $f(n)$ is superpolynomial (see [10]).

Theorem 3. *If there exists $\epsilon > 0$ such that EXP cannot be approximated by circuits of size 2^{n^ϵ} then there exists $c > 0$ such that $\mathsf{BPP} \subseteq \mathsf{DTIME}(2^{\log^c n})$.*

We can use this theorem as follows. In the proof of Theorem 2, if we build segments of size 2^{n^ϵ} (instead of $n^{1+\log n}$) for $\epsilon < 1$ and repeat the process for each segment until we fill $\{0,1\}^n$, then it is easy to see that every program of size 2^{n^ϵ} must make a mistake not only on one, but on a fraction $\geq 1/(2n^\epsilon)$ of the

inputs (otherwise the segments could be compressed by encoding separately the "good" program and the positions where it makes a mistake). That is, assuming (SI), EXP cannot be approximated by circuits of size 2^{n^ϵ}, in the sense of [10]. Theorem 3 therefore yields the following corollary.

Corollary 2. *If* (SI) *holds then there is* $c > 0$ *such that* $\mathsf{BPP} \subseteq \mathsf{DTIME}(2^{\log^c n})$.

6 Further Research and Acknowledgement

It would be interesting to overcome the problem with $q(t)$ time bounds in the hypothesis (SI) (instead of $tq(n)$), in order to be able to use the usual statement of polynomial-time symmetry of information. Then one could try to obtain unconditional results by using variants of resource-bounded Kolmogorov complexity such as CBP or CAMD (see [7]).

The author is really indebted to Andrei Romashchenko for the useful and numerous discussions on this paper and on Kolmogorov complexity (in particular on symmetry of information). He also wants to thank Pascal Koiran for pointing out the open problem "$\mathsf{EXP} \subset \mathsf{P/poly}$?", and the anonymous referees for useful comments.

References

1. Adleman, L.M.: Two theorems on random polynomial time. In: Proceedings of the 19th IEEE Symposium on Foundations of Computer Science, pp. 75–83. IEEE Computer Society Press, Los Alamitos (1978)
2. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity I. EATCS monographs on theoretical computer science, vol. 11. Springer, Heidelberg (1988)
3. Homer, S., Mocas, S.: Nonuniform lower bounds for exponential time classes. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 159–168. Springer, Heidelberg (1995)
4. Kannan, R.: Circuit-size lower bounds and non-reducibility to sparse sets. Information and Control 55, 40–56 (1982)
5. Karp, R., Lipton, R.: Turing machines that take advice. L'Enseignement Mathématique 28, 191–209 (1982)
6. Kolmogorov, A.: Combinatorial foundations of information theory and the calculus of probabilities. Russian Mathematical Surveys 38(4), 29–40 (1983)
7. Lee, T., Romashchenko, A.: Resource bounded symmetry of information revisited. Theoretical Computer Science, 345(2–3), 386–405 (2005). Earlier version in: 29th Symposium on the Mathematical Foundations of Computer Science (2004)
8. Li, M., Vitányi, P.: An introduction to Kolmogorov complexity and its applications, 2nd edn. Graduate texts in computer science. Springer, Heidelberg (1997)
9. Longpré, L., Watanabe, O.: On symmetry of information and polynomial time invertibility. Information and Computation 121(1), 14–22 (1995)
10. Nisan, N., Wigderson, A.: Hardness vs randomness. Journal of Computer and System Sciences 49(2), 149–167 (1994)
11. Ronneburger, D.: Kolmogorov Complexity and Derandomization. PhD thesis, Rutgers University (2004)

12. Schöning, U.: Complexity and Structure. LNCS, vol. 211. Springer, Heidelberg (1986)
13. Vinodchandran, N.V.: A note on the circuit complexity of PP. In: Electronic Colloquium on Computational Complexity, Report No. 56 (July 2004)
14. Zvonkin, A., Levin, L.: The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. Russian Mathematical Surveys 25(6), 83–124 (1970)

Perceptrons of Large Weight

Vladimir V. Podolskii*

Moscow State University
podolskii@lpcs.math.msu.su

Abstract. A threshold gate is a sum of input variables with integer coefficients (weights). It outputs 1 if the sum is positive. The maximal absolute value of coefficients of a threshold gate is called its weight. A perceptron of order d is a circuit of depth 2 having a threshold gate on the top level and any Boolean gates of fan-in at most d on the remaining level.

For every constant $d \geqslant 2$ independent of the number of inputs n we exhibit a perceptron of order d that requires weights at least $n^{\Omega(n^d)}$, that is, the weight of any perceptron of order d computing the same Boolean function is at least $n^{\Omega(n^d)}$. This bound is tight: every perceptron of order d is equivalent to a perceptron of order d and weight $n^{O(n^d)}$. In the case of threshold gates (i.e. $d = 1$) the result was established by Håstad in [1]; we use Håstad's techniques.

1 Introduction

A threshold gate with input Boolean variables $x_1, \ldots, x_n$ is a Boolean function of the form $sgn(\sum_{i=1}^{n} w_i x_i - t)$, where $w_1, w_2, \ldots, w_n, t$ are integers, called the weights and the threshold, respectively, and sgn stands for the sign function: $sgn(x) = 1$ if x is positive, $sgn(x) = 0$ if $x = 0$ and $sgn(x) = -1$ otherwise. For technical reasons we assume that the variables $x_1, \ldots, x_n$ range over $\{-1, 1\}$ (and not $\{0, 1\}$, as usual).

The maximal absolute value of $t, w_1, \ldots, w_n$ is an important parameter of the threshold gate and is called its weight. If a Boolean function is computed by a threshold gate, it can be computed by a threshold gate of weight $n^{O(n)}$ (see [2] or [1]). In the paper by Håstad [1] it was shown that this bound is tight: there exists a function of n variables, that is computable by a threshold gate, but all such threshold gates have weights at least $n^{\Omega(n)}$.

Perceptrons [3] are natural generalizations of threshold gates. A perceptron of order d is a circuit of depth 2 having a threshold gate on the top level and any gates of fan-in at most d on the remaining level. The weight of a perceptron is the weight of its threshold gate.

Any Boolean function $f : \{-1, 1\}^d \to \{-1, 1\}$ can be expressed by a degree-d polynomial in its input variables $x_1, \ldots, x_d$, whose coefficients are rational

* Work is partially supported by grant 06-01-00122 from Russian Federation Basic Research Fund.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 328–336, 2007.

numbers of the form $i/2^d$ with $|i| \leq 2^d$. Since there are at most $2^{2^d}(n+1)^d$ Boolean functions of fan-in at most d of n variables, we can transform any perceptron of order d and weight w into an equivalent one of order d and weight $2^{2^d+d}(n+1)^d w$ with only XOR-gates at the bottom level (a XOR-gate outputs the product of its inputs).

Therefore, in the sequel we assume that perceptrons have only XOR-gates on the bottom level. In other words, in the sequel a perceptron of order d is a polynomial of degree d with integer coefficients, and its weight is the maximal absolute value of its coefficients.

There are at most $O(n^d)$ functions on the bottom level of a perceptron of order d. Consider them as independent variables, the upper bound for weight of threshold gates translates to the upper bound $n^{O(n^d)}$ for perceptrons. That is, every perceptron of order d with n input variables is equivalent to a perceptron of order d and weight $n^{O(n^d)}$ (the constant hidden in O-notation depends on d). The same result was proved in the paper [5] by generalization of the proof of the upper bound for threshold gates and there were conjectured that this bound is tight. In the present paper we prove this conjecture: we show that for every $d > 1$ and every n of the form $d2^m$ there is a perceptron of order d such that every perceptron of order d that computes the same function has weight at least $n^{\Omega(n^d)}$ (the constant hidden in Ω-notation depends on d).

In the case $d = 1$ the similar statement was proved by Håstad in [1]. Our function is a generalization of Håstad's and we use some of the intermediate results of his paper.

Beigel in [4] exhibited a threshold gate with n inputs such that for every d any perceptron of order d computing the same function has weight at least $2^{\Omega(n/d^2)}$. Beigel's bound is weaker for $d = 1$ than Håstad's one, but on the other hand it claims a lower bound for every d.

It is unknown whether there is a statement that generalizes both Beigel's and Håstad's results. More precisely, we do not know whether for any d and n there is perceptron of order d such that for every d' any perceptron of order d' computing the same function has weight at least $n^{f(d')n}$, where f is a fixed positive-valued function.

Some exponential lower bounds for the weight of perceptrons were also shown by Minsky and Papert [3]. However they used a very special set of functions allowed on the bottom level.

2 The Result

Theorem 1. *For some positive ε for any large enough N of the form $d2^m$ there is a function $G : \{-1,1\}^N \to \{-1,1\}$ that is computable by a perceptron of order d and such that any perceptron of order d computing that function has weight at least $N^{\varepsilon N^d}$.*

Proof. Let us denote $n = N/d$. An input to G is a sequence of $d2^m$ bits $-1, 1$ and will be viewed as d functions $f_1, \ldots, f_d$ from $\{-1,1\}^m$ to $\{-1,1\}$. First n

bits are the values of function f_1 on all inputs, the following n bits are the values of f_2 and so on.

For an integer m let $[m]$ denote the set $\{1,\dots,m\}$. For all $\alpha \subseteq [m]$ we consider the monomial φ_α in variables $x_1,\dots,x_m$, that is equal to the product of all variables in α. If α is empty then φ_α is the constant 1. The functions φ_α form an orthogonal basis in the linear space of all functions of the type $\{-1,1\}^m \to \mathbb{R}$. where the inner product is defined as $(f,g) = \sum_{x\in\{-1,1\}^m} f(x)g(x)$.

We use the following lemma from [1]:

Lemma 1. *There is an ordering $\alpha_1,\alpha_2,\dots,\alpha_n$ of all subsets of $[m]$ such that*

1. $|\alpha_i| \leqslant |\alpha_j|$ *for* $i<j$,
2. $|\alpha_i \Delta \alpha_{i+1}| \leqslant 2$ *for all* i.

We will denote this ordering by $<_0$. We also need another ordering denoted by $<_1$. With respect to this ordering the subsets of $[m]$ are arranged as follows: $\overline{\alpha}_1,\overline{\alpha}_2,\dots,\overline{\alpha}_n$. Note that the $<_0$-minimal subset coincides with $<_1$-maximal one, which is the empty set, and vice verse. However the orderings $<_0$ and $<_1$ are not dual: it might happen that both inequalities $\alpha <_0 \beta$ and $\alpha <_1 \beta$ hold simultaneously.

Now we will define an ordering $<$ of all the d-tuples of subsets of $[m]$. This order will be essentially the lexicographic one. More specifically, the first components α^1,β^1 of two tuples $t=\langle\alpha^1,\alpha^2,\dots,\alpha^d\rangle$ and $s=\langle\beta^1,\beta^2,\dots,\beta^d\rangle$ are compared with respect to the order $<_0$, and we declare $t<s$ if $\alpha^1 <_0 \beta^1$ and $t>s$ if $\alpha^1 >_0 \beta^1$. If the first components coincide, the second components are compared w.r.t. the order $<_i$, where i is the parity of the ordinal number of the first component $\alpha_1=\beta_1$ w.r.t. the ordering $<_0$. If both the first and the second components coincide, the third components are compared w.r.t. the order $<_j$, where j is the parity of the ordinal number of the second component w.r.t. the order $<_i$, and so on.

In other words, we define recursively, for each k-tuple t, an ordering $<^t$ of subsets of $[m]$ as follows:

$$<^{\text{empty tuple}} = <_0 \qquad \text{and} \quad <^{\langle t,\alpha\rangle} = <_i$$

where i is the parity of the ordinal number of α with respect to the order $<^t$. The ordering of k-tuples is defined recursively: $\langle t,\alpha\rangle < \langle s,\beta\rangle$ if either $t<s$ (with respect to the order on $(k-1)$-tuples), or $t=s$ and $\alpha <^t \beta$.

Now we need to recall the function F that requires large weights from [1], namely

$$F(f) = sgn(f,\varphi_\alpha),$$

where α stands for the maximal subset with respect to the order $<_0$ such that $(f,\varphi_\alpha)\neq 0$. We define the following generalization of F:

$$G(f_1,\dots,f_d) = sgn((f_1,\varphi_{\alpha^1})\cdot\ldots\cdot(f_d,\varphi_{\alpha^d})),$$

where $\langle\alpha^1,\dots,\alpha^d\rangle$ denotes the maximal d-tuple (with respected to the introduced order) such that $(f_1,\varphi_{\alpha^1})\cdot\ldots\cdot(f_d,\varphi_{\alpha^d})\neq 0$.

Lemma 2. *The function G is computable by a perceptron of order d.*

This lemma is a generalization of a similar lemma of Håstad for $d = 1$.

Proof. We claim that

$$G(f_1, f_2, \ldots, f_d) = sgn\left(\sum_{k=1}^{n^d}(n^d+1)^{k-1}(f_1, \varphi_{\alpha_k^1})\ldots(f_d, \varphi_{\alpha_k^d})\right),$$

where $\langle \alpha_k^1, \ldots, \alpha_k^d\rangle$ denotes k-th d-tuple w.r.t. our ordering (note that (f, φ_α) is a linear function in the variables $f(x)$ and hence this is a representation in the required form).

Let l be the number of the last nonzero term in the displayed sum. It is enough to prove that

$$|(n^d+1)^{l-1}(f_1, \varphi_{\alpha_l^1})\ldots(f_d, \varphi_{\alpha_l^d})| > |\sum_{k=1}^{l-1}(n^d+1)^{k-1}(f_1, \varphi_{\alpha_k^1})\ldots(f_d, \varphi_{\alpha_k^d})|.$$

Since for all k we have $|(f_1, \varphi_{\alpha_k^1})\ldots(f_d, \varphi_{\alpha_k^d})| \leqslant n^d$ and for l we have $|(f_1, \varphi_{\alpha_l^1})\ldots(f_d, \varphi_{\alpha_l^d})| \geqslant 1$, we only need to show

$$(n^d+1)^{l-1} > \sum_{k=1}^{l-1}(n^d+1)^{k-1}n^d,$$

and it is not hard to prove this by induction on l.

Assume that G is represented as a degree-d polynomial in the variables $f_i(x)$, $i = 1, \ldots, d$, $x \in \{-1, 1\}^m$:

$$G(f_1, f_2, \ldots, f_d) = sgn\sum_k u_k g_k \tag{1}$$

where every g_k is a monomial of degree at most d in the variables $f_i(x)$ and all u_k are integers. We need to show that $\max|u_k|$ is large.

Lemma 3. *Setting to 0 all the coefficients u_k such that in the monomial g_k not all functions $f_1, \ldots, f_d$ are presented preserves Equation (1).*

Proof. Consider an arbitrary j from $[d]$. Observe that

$$G(f_1, f_2, \ldots, -f_j, \ldots, f_d) = -G(f_1, f_2, \ldots, f_j, \ldots, f_d)$$

As G is represented by Equation (1) we have

$$G(f_1, f_2, \ldots, -f_j, \ldots, f_d) = sgn(\sum_{i\in A} u_i g_i - \sum_{k\in B} u_k g_k) \tag{2}$$

where A is the set of all indices i such that g_i contains an even number of terms of the form $f_j(x)$ and B is the set of all remaining indices. Since the sign of G

is changing, as we change the sign of f_j, the absolute value of the second sum is greater than the absolute value of the first one and the sign of G depends only on the sign of the second sum. Since this holds for all inputs $f_1, \dots, f_d$, we can set all weights in the first sum to zero. Performing this procedure consequently for all j we get rid of all terms containing even number of occurrences of $f_j(x)$ for some j. As zero is an even number, all the remaining monomials contain some $f_j(x)$ for all j.

Thus we may assume that each monomial in Equation (1) has degree exactly d and each function appears in the monomial exactly once. It will be more convenient to switch to a representation of G as a linear combination of the degree-d monomials in variables (f_i, φ_α), where $\alpha \subseteq [m]$. The following construction is an obvious generalization of Håstad's construction for threshold functions.

The family of functions $\{\varphi_\alpha \mid \alpha \subseteq [m]\}$ form an orthogonal basis in the space $\mathbb{R}^{2^m}$. Using the Fourier expansion, every function f can be expressed as a linear combination of φ_α,

$$f(x) = \frac{1}{2^m} \sum_\alpha (f, \varphi_\alpha)\varphi_\alpha(x). \tag{3}$$

(By the way, as $\varphi_\alpha(x)$ is a product of $x_1, \dots, x_m$, this proves the claim from the Introduction that every Boolean function $f : \{-1, 1\}^d \to \{-1, 1\}$ can be expressed by a degree-d polynomial in its input variables $x_1, \dots, x_d$, whose coefficients are rational numbers of the form $i/2^d$ with $|i| \le 2^d$.)

Replacing in Equation (1) each $f_j(x)$ by the right hand side of Equation (3), we obtain a representation of the form

$$G(f_1, f_2, \dots, f_d) = sgn \sum_{\alpha^1, \dots, \alpha^d} w_{\alpha^1, \dots, \alpha^d} (f_1, \varphi_{\alpha^1})(f_2, \varphi_{\alpha^2}) \dots (f_d, \varphi_{\alpha^d}) \tag{4}$$

where the sum is over all d-tuples. Each $w_{\alpha^1, \dots, \alpha^d}$ is a linear combination of coefficients from Equation (1) u_k with rational coefficients of the form $\pm 1/n^d$. Multiplying the representation (4) by n^d we turn all $w_{\alpha^1, \dots, \alpha^d}$ into integers not exceeding $n^d \max |u_k|$ in absolute value.

Thus it suffices to show that for any representation of G in the form (4), where all $w_{\alpha^1, \dots, \alpha^d}$ are integer we have $\max |w_{\alpha^1, \dots, \alpha^d}| = n^{\Omega(n^d)}$.

Let us arrange all d-tuples t according to the ordering defined above and consider the corresponding sequence $w_{t_1}, \dots, w_{t_{n^d}}$. Now we are going to identify a subsequence in which each term is greater than the preceding one, and most terms are much greater than the preceding ones.

First note that all w_t are positive. Indeed the value of G on any input of the form $f_1 = \varphi_{\alpha^1}, \dots, f_d = \varphi_{\alpha^d}$ is 1. On the other hand, if we substitute $f_1 = \varphi_{\alpha^1}, \dots, f_d = \varphi_{\alpha^d}$ in the right hand side of Equation (4), we obtain $sgn\, w_{\alpha^1, \dots, \alpha^d}$. Thus $sgn\, w_{\alpha^1, \dots, \alpha^d} = 1$.

Fix an arbitrary $(d-1)$-tuple $r = \langle \alpha^1, \alpha^2, \dots, \alpha^{d-1} \rangle$ and consider all the terms in the sequence $w_{t_1}, \dots, w_{t_{n^d}}$ corresponding to d-tuples t_i of the form $\langle r, \alpha^d \rangle$ (that is, whose projection on the first $d-1$ coordinates is equal to r).

Note that

$$G(\varphi_{\alpha^1}, \ldots, \varphi_{\alpha^{d-1}}, f_d) = F(f_d)$$

where F is the Håstad's function, or

$$G(\varphi_{\alpha^1}, \ldots, \varphi_{\alpha^{d-1}}, f_d) = \widehat{F}(f_d),$$

where $\widehat{F}$ is the "dual" of F, that is $\widehat{F}(f) = sgn(f, \varphi_\alpha)$, where the set $\alpha \subseteq [m]$ is maximal w.r.t. the order $<_1$ such that $(f, \varphi_\alpha) \neq 0$. Indeed, the value $(\varphi_{\alpha^1}, \varphi_{\beta^1}) \cdot \ldots \cdot (\varphi_{\alpha^{d-1}}, \varphi_{\beta^{d-1}}) \cdot (f_d, \varphi_{\beta^d})$ is nonzero only if $\beta^1 = \alpha^1, \ldots, \beta^{d-1} = \alpha^{d-1}$ and the ordering on the last component is either $<_0$, or $<_1$.

On the other hand,

$$G(\varphi_{\alpha^1}, \ldots, \varphi_{\alpha^{d-1}}, f_d) = sgn \sum_{\alpha^d} w_{\alpha^1, \ldots, \alpha^{d-1}, \alpha^d}(f_d, \varphi_{\alpha^d}).$$

Thus the coefficients $w_\alpha = w_{\alpha^1, \ldots, \alpha^{d-1}, \alpha}$ represent F or $\widehat{F}$, and we are able to use the following bound from [1]:

Lemma 4. *Let the function F be represented in the form $F(f) = sgn \sum_\alpha w_\alpha(f, \varphi_\alpha)$. For all $i \in [n]$ let α_i be i-th set w.r.t. the ordering $<_0$. For all i such that $|\alpha_i| \geqslant 2$ we have*

$$w_{\alpha_i} > (2^{|\alpha_i|-1} - 1) w_{\alpha_{i-1}} \geqslant 2^{|\alpha_i|-2} w_{\alpha_{i-1}}.$$

Consider the minimal set α with respect to $<_0$ of cardinality 2, call it A. Assume that $F(f) = sgn \sum_\alpha w_\alpha(f, \varphi_\alpha)$ is a representation of F. Let α increase from A to its complement $\bar{A}$. By Lemma 4 w_α increases at least

$$\prod_{2 \leqslant |\alpha| < m-2} 2^{|\alpha|-2}$$

times, as α increases from A to $\bar{A}$. The sum of cardinalities of all subsets of $[m]$ is at least $\Omega(m2^m)$ (indeed more than half of them have cardinality at least $m/2$). Therefore this product is at least

$$2^{\sum_{2 \leqslant |\alpha| < m-2} |\alpha| - 2^{m+1}} = 2^{\Omega(m2^m) - 2(1+m+m(m-1)/2) - 2^{m+1}} = 2^{\Omega(m2^m)} = n^{\Omega(n)}.$$

Thus for every fixed $(d-1)$-tuple r such that $<^r = <_0$ we have $w_{\langle r, \bar{A} \rangle} \geqslant n^{\Omega(n)} w_{\langle r, A \rangle}$.

We need a similar bound for those $(d-1)$-tuples r such that $<^r = <_1$. We prove it by translating the properties of F into properties of $\widehat{F}$.

Lemma 5. *For all weights w_α, the equality $\widehat{F}(f) = sgn \sum_\alpha w_\alpha(f, \varphi_\alpha)$ holds for all f if and only if the equality $\widehat{F}(f) = sgn \sum_\alpha w_\alpha(f, \varphi_{\overline{\alpha}})$ holds for all f.*

Proof. Assume that $F(f) = sgn \sum_\alpha w_\alpha(f, \varphi_\alpha)$ for all f. Consider the function $F(\varphi_{[m]} \cdot f)$ where $(f \cdot g)(x) = f(x) \cdot g(x)$.

Observe that $(\varphi_{[m]} \cdot f, \varphi_\alpha) = (f, \varphi_\alpha \cdot \varphi_{[m]}) = (f, \varphi_{\overline{\alpha}})$. Thus $F(\varphi_{[m]} \cdot f) = sgn(\varphi_{[m]} \cdot f, \varphi_\alpha) = sgn(f, \varphi_{\bar{\alpha}})$, where α is maximal w.r.t. the order $<_0$ such that inner product is nonzero. Hence by definition of the order $<_1$ the set $\overline{\alpha}$ is maximal in this order such that inner product is nonzero. Therefore we have $\widehat{F}(f) = F(\varphi_{[m]} \cdot f) = sgn \sum_\alpha w_\alpha (\varphi_{[m]} \cdot f, \varphi_\alpha) = sgn \sum_\alpha w_\alpha (f, \varphi_{\overline{\alpha}})$.

The converse implication is proved in a similar way.

This lemma implies that for every $(d-1)$-tuple r such that $<^r = <_1$ we have $w_{\langle r,A\rangle} \geqslant n^{\Omega(n)} w_{\langle r,\bar{A}\rangle}$.

Now we introduce some notation allowing to express the proved bounds in a uniform way. Let $\min(<_0) = \max(<_1) = A$ and $\min(<_1) = \max(<_0) = \bar{A}$. Let $\text{next}_{<_i}(\alpha)$ (for $\alpha \neq \max(<_i)$) denote the set following α with respect to the order $<_i$. Using this notation we can summarize the proved bounds as follows. Let t be a d-tuple and let α denote ith component of t so that $t = \langle p, \alpha, r\rangle$ and the length of p is $i-1$. Call the ith component of t *good* if $\alpha \in [\min; \max]$ w.r.t. $<^p$. Otherwise call the ith component of t *bad*.

- If d-tuples s and t have the same common prefix r of length $d-1$ and their last components are equal to $\max(<^r)$ and $\min(<^r)$, respectively, then $w_s > n^{\Omega(n)} w_t$.
- If d-tuples s and t coincide in all components except the ith component and the ith component of t is good and follows the ith component of s w.r.t. the ordering $<^r$ (where r is their length-$(i-1)$ common prefix) then we have $w_s > w_t$. Indeed, if we fix all components except the ith component, we will obtain either function F, or function $\widehat{F}$ and by Lemma 4 and Lemma 5 we will get the desired inequality. (In fact, these lemmas provide even a stronger inequality, which we do not need.)

We distinguish 2 cases.

Case 1. The parity of the ordinal number of A with respect to the ordering $<_0$ is different from that with respect to the ordering $<_1$. (This implies that the same is true for $\bar{A}$.)

Let us delete from the sequence $w_{t_1}, \ldots, w_{t_{n^d}}$ all w_t for d-tuples t that contain a bad component. The remaining terms split naturally into $(n')^{d-1}$ continuous subsequences, where n' denotes the number of subsets of $[m]$ between A and $\bar{A}$ w.r.t. $<_0$ (which is equal to the number of subsets of $[m]$ between $\bar{A}$ and A w.r.t. $<_1$). We have shown that the last term of each such continuous subsequence is more than $n^{\Omega(n)}$ times bigger than the first one. We claim that the first term w_t of each such continuous subsequence is greater than the last term w_s of the preceding continuous subsequence.

To prove the claim we show that s and t differ only in one component. Indeed, s and t have the form

$$s = \langle r, \alpha, \max_{i+1}, \max_{i+2}, \ldots, \max_d\rangle, t = \langle r, \text{next}(\alpha), \min_{i+1}, \min_{i+2}, \ldots, \min_d\rangle$$

where "next" is with respect to $<^r$ and all minima and maxima are taken with respect to the orders associated with respective prefixes.

The orders corresponding to length-i prefixes of s and t are different and hence the sets $\max_{i+1}$ and $\min_{i+1}$ coincide. Moreover, their ordinal numbers in respective orderings have different parities. Therefore the sets $\max_{i+2}$ and $\min_{i+2}$ coincide as well. Reasoning by induction we see that $\max_{i+k}$ and $\min_{i+k}$ coincide for all $k \leq d-i$.

Thus we have $w_t > w_s$. This implies that $\max |w_t| > (n^{\Omega(n)})^{(n')^{d-1}}$. Obviously, $n' > n/2$ and we are done.

Case 2. The parities of the ordinal numbers of A with respect to the orderings $<_0, <_1$ coincide. What changes in the above argument? In this case s and t differ in all components $i+1, i+2, \ldots, d$ and we cannot claim that $w_t > w_s$.

We modify the argument as follows. Consider the tuples

$$\begin{aligned} s &= \langle r, \alpha, \max_{i+1}, \max_{i+2}, \ldots, \max_d \rangle, \\ s_1 &= \langle r, \mathrm{next}(\alpha), \min_{i+1}, \max_{i+2}, \ldots, \max_d \rangle, \\ s_2 &= \langle r, \mathrm{next}(\alpha), \mathrm{next}(\min_{i+1}), \min_{i+2}, \ldots, \max_d \rangle, \\ &\ldots \\ s_{d-i} &= \langle r, \mathrm{next}(\alpha), \mathrm{next}(\min_{i+1}), \mathrm{next}(\min_{i+2}), \ldots, \mathrm{next}(\min_{d-1}), \min_d \rangle. \end{aligned}$$

In each of these tuples next, min, max are understood with respect to the orderings specified by the respective prefixes of the tuple.

Each of these tuples differs from the preceding one only in one component, thus $w_{s_{d-i}} > \cdots > w_{s_1} > w_s$. Note that s_{d-i} is the first term of a continuous subsequence. Although that subsequence is not the subsequence that follows s, the total number of d-tuples between s and s_{d-i} is rather small. Indeed, each such d-tuple has a component that is equal either to A, or to $\bar{A}$. So all d-tuples all components of which lie strictly between A and $\bar{A}$ will lie in our subsequence. This implies that $\max |w_t| > (n^{\Omega(n)})^{(n'-2)^{d-1}}$. Obviously, $n'-2 > n/2$ and we are done.

Remark 1. The proof of the theorem could be modified as follows. First, we could try to find an ordering in Lemma 1 so that to get rid of Case 2. However we don't know if there is such ordering. Second, we could reason as follows: Remove in advance all the subsets of $[m]$ that do not belong to the union of segment $[A, \bar{A}]$ (w.r.t. $<_0$) and $[\bar{A}, A]$ (w.r.t. $<_0$). (Note that these segments do not coincide, as the orders $<_0$ and $<_1$ are not dual.) Then define recursively a large enough ordered set M of d-tuples of remaining sets so that every two consecutive tuples in M differ only in one component. Then define G, as above, but this time using only d-tuples from M. This option requires to re-prove Lemma 4, as we cannot use its statement directly. We believe that the presented proof is simpler than that obtained along these lines.

Acknowledgements. The author is grateful to Professor Nikolay Vereshchagin for stating the problem and for constant attention to this work.

References

1. Håstad, J.: On the size of weights for threshold gates. SIAM Journal on Discrete Mathematics 7(3), 484–492 (1994)
2. Muroga, S.: Threshold logic and its applications. Wiley-Interscience, Chichester (1971)
3. Minsky, M.L., Papert, S.A.: Perceptrons. MIT Press, Cambridge, MA (1968)
4. Beigel, R.: Perceptrons, PP and the polynomial hierarchy. Computational Complexity 4, 339–349 (1994)
5. Buhrman, H., Vereshchagin, N., de Wolf, R.: On computation and communication with small bias. In: Accepted for IEEE Conf. on Computational Compl. (2007)

A Padding Technique on Cellular Automata to Transfer Inclusions of Complexity Classes

Victor Poupet

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 France

Abstract. We will show how padding techniques can be applied on one-dimensional cellular automata by proving a transfer theorem on complexity classes (how one inclusion of classes implies others). Then we will discuss the consequences of this result, in particular when considering that all languages recognized in linear space can be recognized in linear time (whether or not this is true is still an open question), and see the implications on one-tape Turing machines.

1 Introduction

Cellular automata (CA) are a simple yet powerful and complex massively parallel computing model. It is known to be Turing-complete, but the parallelism of the computation makes it quite different from usual sequential models (Turing machines, RAM machines, etc.) in terms of algorithmic complexity. Many examples have shown how it can lead to very efficient computations [4,5].

For these reasons it is interesting to study cellular automata as language recognizers. Because of the parallelism it is very hard to prove non-trivial time lower bounds for the recognition of languages: there is currently no known language that can be recognized in linear space that cannot be recognized in linear time.

Also, some techniques that seem simple on sequential models such as Turing machines can be more complicated on cellular automata. In this article we will see how "padding techniques" can be used on cellular automata to transfer an inclusion between two complexity classes to more general classes. This technique has been widely known and used in the case of Turing machines, but it does not work immediately on cellular automata (since all cells work at each step, the cells in the "padded" region also work from the beginning).

The article is structured as follows: in section 2 we give the necessary definitions and recall some useful properties of cellular automata. Section 3 states the transfer theorem and proves it by explaining how to make padding work on one-dimensional cellular automata. Finally, in section 4 we study some consequences of this result, mainly concerning the long time open question of deciding whether or not there exist languages that can be recognized in linear space but not in linear time, and how it is linked to a tight equivalence between one-tape Turing machines and one-dimensional cellular automata.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 337–348, 2007.

2 Definitions and Useful Properties

2.1 Cellular Automata

In this article, we will only consider one-dimensional cellular automata working on the classical neighborhood.

Definition 1. *A* cellular automaton *(CA) is a pair* $\mathcal{A} = (\mathcal{Q}, \delta)$ *where* $\mathcal{Q}$ *is a finite set called* set of states *containing a special state* B*, and* $\delta : \mathcal{Q}^3 \to \mathcal{Q}$ *is the* transition function *such that* $\delta(B, B, B) = B$ *(*B *is a* quiescent *state).*

For a given automaton $\mathcal{A}$, we call *configuration* of $\mathcal{A}$ any function $\mathcal{C} : \mathbb{Z} \to \mathcal{Q}$. From the local function δ we can define a global function

$$\Delta : \left\{ \begin{array}{l} \mathcal{Q}^{\mathbb{Z}} \to \mathcal{Q}^{\mathbb{Z}} \\ \mathcal{C} \mapsto \mathcal{C}' \end{array} \right. \quad | \quad \forall x \in \mathbb{Z}, \mathcal{C}'(x) = \delta(\mathcal{C}(x-1), \mathcal{C}(x), \mathcal{C}(x+1))$$

Elements of $\mathbb{Z}$ are called *cells*. Given a configuration $\mathcal{C}$, we will say that a cell c is in state q if $\mathcal{C}(c) = q$.

If at time $t \in \mathbb{N}$ the CA is in a configuration $\mathcal{C}$, we will say that at time $(t+1)$ it is in the configuration $\Delta(\mathcal{C})$. This defines the *evolution* of a CA from a configuration. This evolution is completely determined by the initial configuration $\mathcal{C}$ and the automaton.

Definition 2 (Finite Configuration). *A configuration* $\mathcal{C} : \mathbb{Z} \to \mathcal{Q}$ *is said to be* finite *if there exists* $x, y \in \mathbb{Z}$ $(x \leq y)$ *such that for all* $n \notin [\![x, y]\!]$, $\mathcal{C}(n) = B$.

The size *of the configuration is the minimal value of* $(y - x + 1)$ *for all* (x, y) *statisfying the definition.*

It is obvious (because the state B is quiescent) that for every finite configuration $\mathcal{C}$, $\Delta(\mathcal{C})$ is also finite. In this article we will only consider finite initial configurations so at all times the configuration of the automaton will be finite.

Definition 3 (Signals). *A* signal *in a CA is a special set of states that "moves" in a direction at a given speed. For example, a signal* s *moving at speed 1 to the right is a pair of states* $\{s, \overline{s}\}$ *such that* $\delta(\overline{s}, s, \overline{s}) = \overline{s}$ *and* $\delta(s, \overline{s}, \overline{s}) = s$ *(if a cell sees that its left neighbor is in state* s*, it becomes of state* s *at the next time, so in some sense the* s *state has moved right). In this example,* $\overline{s}$ *is a neutral state.*

To have a signal move at speed $1/k$ $(k \in \mathbb{Z}^+)$*, we will use* $k+1$ *states* $\{s_1, \ldots, s_k, \overline{s}\}$*, and the rules*

$$\begin{array}{ll} \delta(\overline{s}, s_i, \overline{s}) = s_{i+1} & i \in [\![1, k-1]\!] \\ \delta(\overline{s}, s_k, \overline{s}) = \overline{s} & \\ \delta(s_k, \overline{s}, \overline{s}) = s_1 & \end{array}$$

so that the state s_i *stays on a cell during* k *steps before moving to the right (again, all cells that do not hold the signal are in state* $\overline{s}$*).*

Signals are a very useful tool for computations on cellular automata. Given an automaton $\mathcal{A} = (\mathcal{Q}, \delta)$, adding a signal $S = \{s_1, \ldots, s_k, \overline{s}\}$ corresponds to making a new cellular automaton whose states are $\mathcal{Q} \times S$. Any finite number of signals can be added to an automaton (while still having a finite number of states). Signals can then evolve according to their move rule (as explained in the definition) but can also interact the ones with the others when they meet on a cell (disappear, generate new signals etc.), or even change the main evolution of the automaton on a given cell when it receives a given signals (and that is what they are useful for).

Definition 4 (Space-Time Diagram). *Given a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and a configuration $\mathcal{C}$, we can represent by a two-dimensional diagram the complete evolution of the automaton from $\mathcal{C}$. The initial configuration is drawn horizontally on the bottom line and all the following configurations are drawn successively (time goes from bottom to top). Such a diagram is called* space-time diagram *of $\mathcal{A}$ from configuration $\mathcal{C}$.*

For obvious reasons, we only represent a finite part of a space-time diagram (finite in both space and time) however since we will only consider finite configurations and computations over a finite time we will be able to represent all the necessary information.

The space-time diagram of a CA is a discrete diagram. However, we will sometimes represent it as a continuous figure. In this case the signals (some specific states that move through the configuration) will be represented as line segments, and they will partition the diagram in polygonal parts.

2.2 Language Recognition

Definition 5 (Word Recognition). *We consider a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and an* accepting state $q_f \in \mathcal{Q}$ *such that for all $q_1, q_2 \in \mathcal{Q}$ we have $\delta(q_1, q_f, q_2) = q_f$ (if a cell is in state q_f at one point, it stays in this state forever, such a state is called* persistent*). Let $w = w_0 w_1 \ldots w_{l-1}$ be a word on a finite alphabet $\Sigma \subseteq \mathcal{Q}$. We define the configuration $\mathcal{C}_w$ as follows.*

$$\mathcal{C}_w : \quad \mathbb{Z} \to \mathcal{Q}$$
$$\begin{cases} x \mapsto w_x & \text{if } 0 \leq x < l \\ x \mapsto B & \text{otherwise} \end{cases}$$

We will say that the CA $\mathcal{A}$ recognizes *the word w with accepting state q_f in time $t_w \in \mathbb{N}$ and space $s_w \in \mathbb{N}$ if, starting from the configuration $\mathcal{C}_w$ at time 0, the cell 0 is in state q_f at time t_w and no cell other than the ones in $[\![0, s_w - 1]\!]$ was ever in a state other than B.*

Definition 6 (Language Recognition). *Let $\mathcal{A} = (\mathcal{Q}, \delta)$ be a CA, $L \subseteq \Sigma^*$ a language on the alphabet $\Sigma \subseteq \mathcal{Q}$ and $q_f \in \mathcal{Q}$ a persistent state. Given two functions $T : \mathbb{N} \to \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$, we will say that the language L is* recognized *by $\mathcal{A}$ in time T and space S with accepting state q_f if for every word $w \in \Sigma^*$ of length l, the CA $\mathcal{A}$ recognizes w with accepting state q_f in time $T(l)$ and space $S(l)$ if and only if $w \in L$.*

We will denote as CATIME(T) the class of languages recognizable in time T and as CASPACE(S) the class of languages recognizable in space S.

2.3 Functions

Definition 7 (Time-Constructible Function). *Given a function $f : \mathbb{N} \to \mathbb{N}$, we will say that f is* time-constructible *if there exists a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and two states $q_1, q_f \in \mathcal{Q}$ such that for every $n \in \mathbb{N}$, starting from the initial configuration $\mathcal{C}_{q_1^n}$ (the unary encoding of n into a finite configuration of $\mathcal{A}$) at time 0, the cell 0 is in state q_f for the first time at time $f(n)$.*

The above definition simply means that, given an integer n in unary form, the automaton can "count" $f(n)$ steps and mark the origin when it is done.

Definition 8 (Space-Constructible Function). *Given a function $f : \mathbb{N} \to \mathbb{N}$, we will say that f is* space-constructible *if there exists a CA $\mathcal{A} = (\mathcal{Q}, \delta)$ and two states $q_1, q_f \in \mathcal{Q}$ such that for every $n \in \mathbb{N}$, starting from the initial configuration $\mathcal{C}_{q_1^n}$, after some time all cells ranging from 0 to $f(n)-1$ switch to the state q_f, no cell was in this state before and during the whole computation none of the cells c such that $c < 0$ or $c \geq f(n)$ was ever in a state other than B.*

This definition means that, on input n (encoded in unary), the automaton will compute $f(n)$ in unary without using more cells in the process than the ones needed to write the output (the ones between 0 and $f(n)-1$). The synchronization is not a problem because we can apply the firing squad technique at the end of the computation (see Proposition 1).

Remark. It is obvious, according to this definition, that any function f that is space-constructible is such that $f(x) \geq x$ for all x.

Remark. Any time-constructible function f such that $\forall x, f(x) \geq x$ is space-constructible : we can modify the automaton so that it first compresses its input to a word of half its initial length, and then performs the computation of f using half the space (all states correspond to two states of the initial automaton). While this computation occurs, a signal moves at speed 1/2 from the origin to the right. When the time $f(n)$ is computed on the origin, a new signal appears that moves to the right at speed 1. These two signals meet at the cell $f(n)$, and no more space has been used during the computation.

2.4 Basic Properties

Proposition 1 (Firing Squad). *It is possible to synchronize a segment of k adjacent cells (have them enter a specific state for the first time at the same time). It can be done in time $ak + b$ for any $a \geq 2$ and $b \geq 0$, starting from the time when the leftmost cell emits the "synchronization" signal. The rightmost cell need only be created at time $(a-1)k$.*

The firing squad problem has been well studied, and many ingenious solutions have been found. In this article we will need a solution that can be delayed so

that the synchronization takes exactly ak steps for some integer $a \geq 2$ and such that the rightmost cell can be constructed as late as possible (at time $(a-1)k$). Such a solution is a particular case of the general "delayed" solutions explained in [7].

Proposition 2. *Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. If for all $x \in \mathbb{N}$, we have $f(x) \geq 3x$ then the function $x \mapsto f(x) - x$ is also time-constructible.*

Proof. The first step is to mark the cell $n/3$. This can be done at time $2n/3$ by sending a signal at speed $1/2$ to the right from the origin and a signal at speed 1 to the left from the rightmost letter of the word. Then, with a firing squad technique between the origin and the cell $n/3$ (see proposition 1) and a compression of the information it is possible to start the computation of $f(n)$ at scale $1/3$ from time n. This will mark the time $n + f(n)/3$ on the origin, and from this time, a signal s_1 appears and moves to the right at speed 1

Meanwhile, we mark the time $2n$ on the origin, and from there a signal s_2 starts going right at speed $1/2$. This signal meets the signal s_1 at time $2f(n)/3$ on cell $f(n)/3 - n$ (the signals meet only if s_2 was generated before s_1, which means that $2n \leq n + f(n)/3$, i.e. $3n \leq f(n)$), and from here a new signal starts moving towards the origin at speed 1, this last signal will arrive at the origin at time $2f(n)/3 + f(n)/3 - n = f(n) - n$.

Proposition 3. *Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function and $k \geq 2$ an integer. If for all $x \in \mathbb{N}$ we have $f(x) \geq 2kx$ then the function $x \mapsto f(x)/k$ is also time-constructible.*

Proof. This construction is simpler than the previous one, all we have to do is mark the cell $f(n)/k$ at time $(k-1)f(n)/k$ with the use of a signal moving to the right at speed $1/k$ and one going to the left at speed 1, and then with a firing squad technique start the computation of $f(n)$ at scale $1/k$ to mark the origin at time $f(n)/k + n$. We then use Proposition 2 to construct $x \mapsto f(x)/k$.

The combination of both constructions requires $f(x) \geq 2kx$.

Proposition 4 (Turing-Cellular Comparison). *Let L be a language in Σ^*. If L is recognized in time $T : \mathbb{N} \to \mathbb{N}$ and space $S : \mathbb{N} \to \mathbb{N}$ by a Turing machine using one tape and one head, then it is recognizable in time T and space S by a cellular automaton.*

Moreover, if L is recognized by a cellular automaton in time T and space S then it is recognized by a one-tape Turing Machine in time $T \times S$ and space S.

Proof. This property follows from straightforward simulations of one-tape Turing machines by cellular automata and *vice versa*. Details on these simulations can be found in [9].

Proposition 5. *For every language L and every integer k, if L can be recognized in space $x \mapsto kx$ then it can be recognized in space $x \mapsto x$. In other words* $\bigcup_{k \in \mathbb{N}} \mathrm{CASPACE}(k.\mathrm{Id}) = \mathrm{CASPACE}(\mathrm{Id})$.

The proof is identical to that of the same result in the Turing case.

Proposition 6. *For every language L recognizable in time T and space S on a cellular automaton, there is a CA that recognizes L in time $T' \leq T$ and space $S' \leq S$ and such that all cells $c < 0$ remain in the blank state B during all the computation (for every word w).*

The proof of this proposition is identical to the proof that semi-infinite tape Turing machines are equivalent to bi-infinite tape Turing machines.

3 Transfer Theorem

In this section, we will state and prove the transfer theorem. Some consequences of this theorem will be discussed in the next section.

Theorem 1. *Given two space-constructible functions $f, g : \mathbb{N} \to \mathbb{N}$ if* CASPACE$(f) \subseteq$ CATIME(g) *then for any time-constructible function $h : \mathbb{N} \to \mathbb{N}$ such that $\forall x, h(x) \geq 6x$ we have*

$$\text{CASPACE}(f \circ h) \subseteq \text{CATIME}(g \circ h)$$

To prove this, we will consider three functions f, g and h that satisfy the hypothesis of Theorem 1 and we will assume that

$$\text{CASPACE}(f) \subseteq \text{CATIME}(g)$$

Moreover we consider a language L over the alphabet Σ that can be recognized in space $f \circ h$. We will show that L can be recognized in time $g \circ h$.

3.1 The Language $\widetilde{L}$

Lemma 1. *The language $\widetilde{L} = \{w\#^{h(|w|)-|w|} \mid w \in L\}$, where $\#$ is a new symbol not in Σ, is recognizable in time g.*

Proof. We first show that $\widetilde{L}$ is in CASPACE(f). To do so we have to check that w is in L and that there are exactly the right amount of $\#$ symbols. Because L is in CASPACE$(f \circ h)$ and h is space-constructible both verifications can be done (if more space is needed it means that there were not enough $\#$). The conclusion follows from the hypothesis that CASPACE$(f) \subseteq$ CATIME(g).

From now on, we will assume that we have a CA $\mathcal{A}$ that recognizes $\widetilde{L}$ in time g. We will now construct a CA $\mathcal{A}'$ that recognizes L in time $g \circ h$.

3.2 Compression of the Space-Time Diagram

The aim of this section is to explain how it is possible to construct the cellular automaton $\mathcal{A}'$ that will, on input w, simulate the behavior of $\mathcal{A}$ on input $w\#^{h(|w|)-|w|}$ without any loss of time. The initial configuration of $\mathcal{A}$ is very simple, and the only information that $\mathcal{A}'$ is missing is the exact location of the cell $(h(|w|) - 1)$.

Let us have a look at a typical space-time diagram of $\mathcal{A}$ on input $\omega = w\#^{h(|w|)-|w|}$ (see figure 1). According to proposition 6 we can consider that no computation takes place on the negative cells. We will now consider separately the area of the space-time diagram that is on top of the initial word w (the cells on which the letters were initially written) and the rest of the diagram, starting from the cell $|w|$ (these two areas are separated by a dashed line on the figure). If we assume that $\mathcal{A}'$ is able to compute correctly all the states on the column that is immediately to the right of this dashed line (all the states of the cell $|w|$), it will mean that it is also capable of computing all the states in the area corresponding to the cells from 0 to $(|w|-1)$ because the initial states are known (the letters of w) and that at each time the states of these cells only depend on the previous states of these cells and the cell $|w|$.

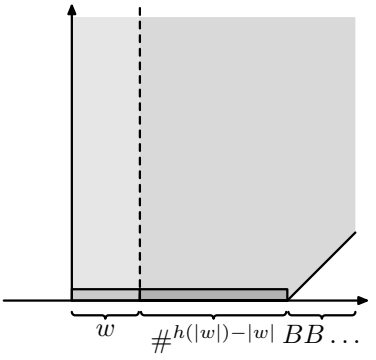


Fig. 1. Space-time diagram of $\mathcal{A}$

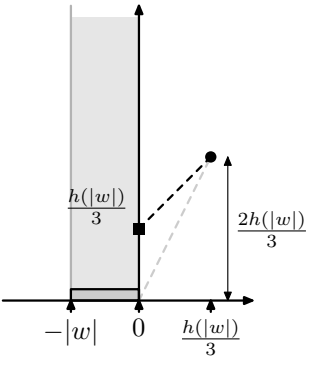


Fig. 2. Construction of the point $(h(|w|)/3, 2h(|w|)/3)$ at time $2h(|w|)/3$ (with shifted coordinates)

To compute the states on the cell $|w|$, we will operate a geometric transformation of the rightmost part of the space-time diagram (the part that is represented on the right of the dashed line) that will preserve this column and let the automaton $\mathcal{A}'$ construct the missing portion of the initial configuration of $\mathcal{A}$: the segment of # symbols of length $h(|w|) - |w|$.

Definition of the compression. Let us shift the system of coordinates on the diagram so that the new origin is now the cell $|w|$ (the cell that was previously referred to as c is now the cell $(c - |w|)$). Let us consider the transformation

$$\sigma : \begin{cases} \mathbb{N}^2 \to (\frac{1}{3}\mathbb{N})^2 \\ (x, y) \mapsto (\frac{x}{3}, y + \frac{2x}{3}) \end{cases}$$

The vertical axis is invariant by σ and the image of the horizontal axis (the initial configuration) is now a line of slope 2. Figure 3 illustrates the effects of σ on the right part of the space-time diagram of $\mathcal{A}$. We see that each cell

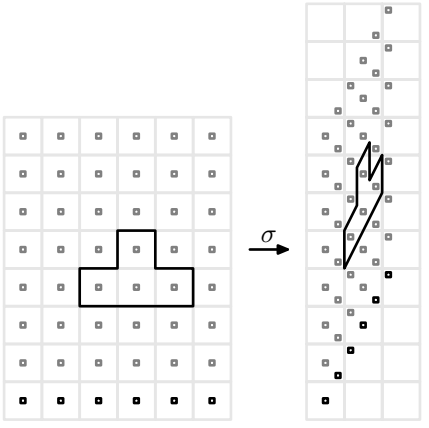


Fig. 3. Effects of σ on the right part of the space-time diagram

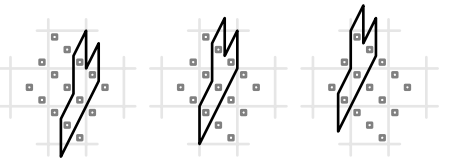


Fig. 4. The 3 possible cases when trying to apply the transition rule of $\mathcal{A}$ after compression by σ

receives, after compression, the states of 3 cells before compression (except for the origin that keeps one single state). We have alse represented on the figure the neighborhood of a cell and its successor, both before and after compression.

All we have to do now is show that $\mathcal{A}'$, on input w, can simulate the behavior of $\mathcal{A}$ by constructing in real-time the image by σ of the right part of the space-time diagram of $\mathcal{A}$.

How the Simulation Works. Since we have assumed that for all $x \in \mathbb{N}$, $h(x) \geq 6x$, the function $h/3$ is time-constructible (proposition 3) which means that the origin (still with shifted coordinates) can be marked at time $h(|w|)/3$ which means we can mark the cell $h(|w|)/3$ at time $2h(|w|)/3$ by sending signals as illustrated on figure 2 (the black dashed line is a signal that moves at speed 1 whereas the grey dashed line is a signal moving at speed 1/2). The space-time point $(h(|w|)/3, 2h(|w|)/3)$ is exactly the image by σ of the point $(h(|w|), 0)$ that happens to be the last cell in state # in the initial configuration of $\mathcal{A}$.

Therefore, even if $\mathcal{A}'$ does not have all the information held by ω in its initial configuration, it can construct the image by σ of this initial configuration. Indeed, by propagating a signal to the right at speed 1/2 from the shifted origin (the right border of the input word) it can write # symbols on the segment that is the image of the # segment in the initial configuration of $\mathcal{A}$ and we have just seen that $\mathcal{A}'$ is capable of deciding exactly where this segment must end.

We now have to check that $\mathcal{A}'$ can apply the transition rule of $\mathcal{A}$ to the information that is held by the cells after applying σ to the space-time diagram.

The states on the space-time diagram of $\mathcal{A}$ can be in one of 3 different situations after compression by σ as shown by figure 3: upper-left, center or lower-right. Figure 4 illustrates the situation for each of these cases.

In each part of the figure we have represented which states must be known in order to compute the next state on a given position. We see that the information needed to compute the "next state" on a lower-right position is located on the current cell and its right neighbor. To compute the next state on a central position, the cell must look at her right neighbor and also know the next state on the lower-right position, which, as we have seen, can be computed with the information that is accessible to the cell. Finally, computing a state in the upper-left position requires to see a state held by the left neighbor of the cell, and also computing the next state on the central position.

In all three cases, we have shown that if the space-time diagram of $\mathcal{A}'$ contains the image by σ of the configuration of $\mathcal{A}$ at time t then $\mathcal{A}'$ can compute the image by σ of the configuration of $\mathcal{A}$ at time $(t+1)$. Since the image of the initial configuration of $\mathcal{A}$ can be recreated by $\mathcal{A}'$, it is possible to compute correctly everything that happens on the right part of the space-time diagram of $\mathcal{A}$ (on the right of the last letter of the input word).

3.3 End of the Computation

We have seen how $\mathcal{A}'$ can simulate the computation of $\mathcal{A}$ on the area that is on top of the input word without any distortion if it can compute correctly all the states on the column $|w|$ (back on the original coordinates system), and how this column can be computed by compressing the space-time diagram of $\mathcal{A}$ (but the compression does not affect the column $|w|$).

The automaton $\mathcal{A}$ on input w can therefore simulate completely the behavior of $\mathcal{A}$ on input $\omega = w\#^{h(|w|)-|w|}$, which means that at time $g(|\omega|) = g \circ h(|w|)$ it can decide whether or not ω is in $\widetilde{L}$ and hence whether or not w is in L.

The language L is recognized in time $g \circ h$, which concludes the proof of theorem 1.

4 Linear Time, Linear Space

4.1 The Open Questions

As in the Turing case, it is very hard on cellular automata to prove lower bounds of complexity. Moreover, very little is known about comparing space and time complexities. If it is easy to show that for any function f, $\mathrm{CATIME}(f) \subseteq \mathrm{CASPACE}(f)$, showing that the inclusion is strict (or that it is an equality) is much harder.

In particular, extensive work has been made on the lower complexity classes:

- Real time: CATIME(Id);
- Linear time: $\bigcup_{k\in\mathbb{N}} \mathrm{CATIME}(k.\,\mathrm{Id})$;
- Linear space: $\bigcup_{k\in\mathbb{N}} \mathrm{CASPACE}(k.\,\mathrm{Id})$;

Space and time speed-up theorems show that the linear class time is equal to the class CATIME(2 Id) and that the linear time class is equal to CASPACE(Id).

There is also an immediate sequence of inclusion between these three classes but it is still unknown if any of these inclusions is strict.

The questions have been first stated by A. R. Smith III in 1972 [10] and have remained open since. Some significant improvements have been made though, for example relating the equality of complexity classes to their closure properties [6] or working on weaker versions of cellular automata [1,2,3]. These questions are in many ways similar to the well known open question "P = PSPACE ?".

In this section we will use the theorem 1 to show some consequences of the assumptions that CATIME(Id) = CASPACE(Id) or CATIME(2 Id) = CASPACE (Id).

4.2 Stronger Version of the Main Theorem

We have the following results:

Lemma 2. *Under the hypothesis that* CATIME(Id) = CASPACE(Id)*, every function f that is space-constructible is also time-constructible.*

Proof. We show that the language $L_f = \{1^x\#^{f(x)-x} \mid x \in \mathbb{N}\}$ is in CASPACE(Id) and thus in CATIME(Id). Then we can make a cellular automaton that, on input 1^x will work as if all symbols on the right of the last '1' were # and continue until the first (and only) word of the form $1^k\#^*$ is accepted (at time t the automaton considers that it has received all the relevant information, and therefore knows whether or not the word $1^x\#^{t-x}$ is in L_f). It will happen at time $f(x)$, so f is time-constructible.

Proposition 7. *If* CATIME(Id) = CASPACE(Id) *then for any space-constructible function $h : \mathbb{N} \to \mathbb{N}$ we have*

$$\text{CASPACE}(h) \subseteq \text{CATIME}(h)$$

Proof. The proof of this proposition is very similar to that of theorem 1. The only difference is that the hypothesis on h is now weaker because of lemma 2.

4.3 Equivalence of Cellular and Turing Models

For a given function $f : \mathbb{N} \to \mathbb{N}$, we will denote as DTIME(f) the class of languages recognizable on a deterministic one-tape Turing machine in time f, and DSPACE(f) the class of languages recognizable on a deterministic one-tape Turing machine in space f. It is known that DSPACE(f) = CASPACE(f). Here we will only consider deterministic one-tape Turing machines, that we will simply call Turing machines.

If CASPACE(Id) $\subseteq$ CATIME(2 Id) then, because of Theorem 1 and Proposition 4, for every time-constructible function f every language L that is recognized in space f (in the usual Turing sense since space complexities are the same for CA and Turing machines) can be recognized in time f^2, in other words :

$$\text{DTIME}(f) \subseteq \text{DSPACE}(f) \subseteq \text{CATIME}(2f) \subseteq \text{DTIME}(f^2)$$

Whether there exists or not a function f that contradicts these inclusions is still an open question. An important consequence of this is that P = PSPACE. Of course, the fact that CASPACE(Id) $\subseteq$ CATIME(2 Id) implies P = PSPACE is not new since it is an immediate consequence of the PSPACE-completeness of TQBF, that is known to be in CASPACE(Id), but the proof we have here does not require the existence of PSPACE-complete problems.

Remark. For any function f such that $\forall x, f(x) \geq 2x$, we have CATIME(f) = CATIME($2f$) (because of a linear acceleration theorem on CA).

However, the most interesting consequence of the inclusion CASPACE(Id) $\subseteq$ CATIME(2 Id) is a very strong equivalence between the cellular and the Turing computing models obtained with the use of a theorem by M. Paterson:

Theorem 2 (Paterson [8]). *For every function $f : \mathbb{N} \to \mathbb{N}$ such that $\forall x$, $f(x) \geq x$, we have* DTIME(f^2) $\subseteq$ DSPACE(f).

If CASPACE(Id) $\subseteq$ CATIME(2 Id), then we already know that for every time-constructible function f,

$$\text{DSPACE}(f) = \text{CASPACE}(f) \subseteq \text{CATIME}(2f) \subseteq \text{DTIME}(f^2)$$

and from the above theorem, we get

$$\text{CATIME}(2f) = \text{DTIME}(f^2) = \text{DSPACE}(f) = \text{CASPACE}(f)$$

This would mean that any sequential algorithm (on a single-tape Turing machine) can be speed-up by a quadratic factor when considering it on a parallel model (cellular automaton). Since we know that this speed-up factor is optimal (what can be done in time f on a CA can be done in time f^2 on a TM), it would mean that the two models are very tightly equivalent in terms of time complexity, since knowing the exact complexity of a problem on one model would give the exact complexity on the other model.

Conversely, to show that CASPACE(Id) $\not\subset$ CATIME(2 Id), it would be enough to find a language that can be recognized in time f^2 (for some time-constructible function f such that $\forall x, f(x) \geq x$) on a one-tape Turing machine but not in time $2f$ on a cellular automaton. However, no such example is known as of today.

5 Conclusion

We have shown in this paper how padding techniques that are broadly known to work on Turing machines can also work on cellular automata, by transforming the space-time diagram so that the automaton has enough time during the computation to "recreate" the information that was held by the "padded" cells. Using this technique we have shown the cellular equivalent to the transfer theorem.

This theorem enabled us to establish an important link between problems of separation of space and time complexity classes and problems of parallel computation (if CASPACE(Id) $\subseteq$ CATIME(2 Id) then every sequential algorithm can be parallelized with a quadratic speed-up).

Even though both questions remain open, the implications that we have seem to indicate that the inclusions do not hold, and open new possibilities to prove it.

References

1. Choffrut, C., Čulik II, K: On real-time cellular automata and trellis automata. Acta Informatica 21, 393–407 (1984)
2. Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions on Computers C18, 349–365 (1969)
3. Čulik II, K., Gruska, J., Salomaa, A.: Systolic automata for VLSI on balanced trees. Acta Inf. 18, 335–344 (1982)
4. Čulik II, K.: Variations of the firing squad problem and applications. Inf. Process. Lett. 30, 153–157 (1989)
5. Fischer, P.C.: Generation of primes by one-dimensional real-time iterative array. Journal of the Assoc. Comput. Mach. 12, 388–394 (1965)
6. Ibarra, O., Jiang, I.: Relating the power of cellular arrays to their closure properties. Theoretical Computer Science 57, 225–238 (1988)
7. La Torre, S., Napoli, M., Parente, D.: Synchronization of a line of identical processors at a given time. Fundamenta Informaticae 34, 103–128 (1998)
8. Paterson, M.S.: Tape bounds for time bounded Turing machines. JCSS 6, 116–124 (1972)
9. Smith III, A.R.: Simple computation-universal cellular spaces. J. ACM 18, 339–353 (1971)
10. Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. Journal of the Assoc. Comput. Mach. 6, 233–253 (1972)

Kolmogorov Complexity, Lovász Local Lemma and Critical Exponents

Andrey Yu. Rumyantsev

Moscow State Lomonosov University, Mathematics Dept.,
Logic and Algorithms Theory Division
azrumyan@mail.ru

Abstract. D. Krieger and J. Shallit have proved that every real number greater than 1 is a critical exponent of some sequence [1]. We show how this result can be derived from some general statements about sequences whose subsequences have (almost) maximal Kolmogorov complexity. In this way one can also construct a sequence that has no "approximate" fractional powers with exponent that exceeds a given value.

1 Kolmogorov Complexity of Subsequences

Let $\mathbf{w} = w_0 w_1 \ldots$ be an infinite binary sequence. For any finite set $A \subset \mathbb{N}$ let $\mathbf{w}(A)$ be a binary string of length $\#A$ formed by w_i with $i \in A$ (in the same order as in $\mathbf{w}$). We want to construct a sequence $\mathbf{w}$ such that strings $\mathbf{w}(A)$ have high Kolmogorov complexity for all simple A. (Informally speaking, Kolmogorov complexity of a string z is the minimal length of a program that generates z for a fixed optimal (universal) programming language. See [3] for the definition and properties of Kolmogorov complexity. We use prefix complexity that assumes that all programs are self-delimiting, and denote it by K, but plain complexity can also be used with minimal changes.)

Theorem 1. *Let γ be a positive real number less than* 1. *Then there exists a 0-1-sequence* $\mathbf{w}$ *and an integer c such that for any finite set A of cardinality at least c the inequality*

$$K(A, \mathbf{w}(A)|t) \geqslant \gamma \cdot \#A$$

holds for some $t \in A$.

Here $K(A, \mathbf{w}(A)|t)$ is conditional Kolmogorov complexity of a pair $(A, \mathbf{w}(A))$ relative to t.

Proof. This result is a consequence of Lovász local lemma (see, e.g., [4] for its proof):

Lemma. Assume that a finite sequence of events $E_1, \ldots, E_k$ is given, for each i some subset $N(i) \subset \{1, \ldots, k\}$ of "neighbors" is fixed, positive reals $\varepsilon_1, \ldots, \varepsilon_k < 1$ are chosen in such a way that

$$\Pr[E_i] \leqslant \varepsilon_i \prod_{j \in N(i), j \neq i} (1 - \varepsilon_j)$$

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 349–355, 2007.

and for every i the event E_i is independent of the family of all E_j with $j \notin N(i), j \neq i$. Then the probability of the event "not E_1 and not E_2 and... and not E_k" is at least $(1-\varepsilon_1)\cdot\ldots\cdot(1-\varepsilon_k)$.

The standard compactness argument shows that it is enough to construct an arbitrarily long finite sequence $\mathbf{w}$ that satisfies the statement of Theorem 1 for some fixed value of c (the choice of c will be explained later). Let L be the desired length of this (long) sequence. For any set A of indices not exceeding L and any binary string z of length $\#A$ such that $K(A,z|t) < \gamma\cdot\#A$ for all $t \in A$ consider the event $E_{A,z} = \{w \in \{0,1\}^L \mid w(A) = z\}$ in the probability space that considers w as a sequence of L independent random bits. The set A is called the *support* of the event $E_{A,z}$. We have to prove that the complements of these events (used as $E_1, E_2 \ldots$ in the Lovász local lemma) have non-empty intersection.

This is done by using Lovász lemma. Let us choose some β between γ and 1. Let ε_i be $2^{-\beta s_i}$ where s_i is the size of support of ith event. For each event $E_{A,z}$ the neighbor events are events $E_{A',z'}$ such that the supports A and A' have nonempty intersection. Let us check the assumptions of Lovász lemma.

First, an event $E_{A,z}$ is independent of any family of events whose supports do not intersect A.

Second, consider an event $E_{A,z}$ (i.e., $w(A) = z$) and let n be the cardinality of A. The probability of this event is 2^{-n}. We have to check that 2^{-n} does not exceed $2^{-\beta n}$ multiplied by the product of $(1-2^{-\beta m})$ factors for all neighbor events (where m is the size of the support of the corresponding events):

$$2^{-n} \leqslant 2^{-\beta n}\prod_i (1-2^{-\beta m_i})$$

where $m_0, m_1, \ldots$ are the sizes of the supports of $E_{A,z}$'s neighboring events.

Each factor corresponds to some neighbor event, i.e., an event whose support has a non-empty intersection with A. Fix some intersection point for each neigbor event. Then group the factors in the product according to the intersection points: each group corresponds to some $t \in A$ and is formed by events whose supports contain t.

For any non-negative integer m there are at most $2^{\gamma m}$ factors that belong to the t-group and have size m, since there exist at most $2^{\gamma m}$ objects that have complexity less than γm (relative to a given t). Then we take a product over all $m \geqslant 0$ and multiply the results for all $t \in A$ (there are n of them). The condition of Lovász lemma (that we need to check) now has the form

$$2^{-n} \leqslant 2^{-\beta n}\prod_{m>c} (1-2^{-\beta m})^{2^{\gamma m} n}$$

or (after we remove the common exponent n)

$$2^{\beta-1} \leqslant \prod_{m>c} (1-2^{-\beta m})^{2^{\gamma m}}$$

Bernoulli inequality $(1-h)^u \geqslant 1-hu$ guarantees that this is true if

$$2^{\beta-1} \leqslant 1 - \sum_{m>c} 2^{\gamma m} 2^{-\beta m}$$

Since the left hand side is less than 1 and the geometric series converges, this inequality is true for a suitable c. (Let us repeat how the proof goes: we start with $\beta \in (\gamma, 1)$, then we choose c using the convergence of the series, then for any finite number of events we apply Lovász lemma, and then we use compactness.) □

The inequality established in this theorem has an useful corollary:

$$K(\mathbf{w}(A)|t) \geqslant \gamma \cdot \#A - K(A|t) - O(1),$$

since $K(A, \mathbf{w}(A)|t) \leqslant K(A|t) + K(\mathbf{w}(A)|t) + O(1)$. For example, if A is an interval, then $K(A|t) = o(\#A)$, so this term (as well as an additive constant $O(1)$) can be absorbed by a small change in γ and we obtain the following corollary ("Levin's lemma", see [2] for a discussion and further references): *for any $\gamma < 1$ there exists a sequence* $\mathbf{w}$ *such that all its substrings of sufficiently large length n have complexity at least γn.*

2 Critical Exponents

Let x be a string over some alphabet, and let y be a prefix of x. Then the string $z = x \ldots xy$ is called a *fractional power* of x and the ratio $|z|/|x|$ is its *exponent*. A *critical exponent* of an infinite sequnce $\mathbf{w}$ is the least upper bound of all exponents of fractional powers that are substrings of $\mathbf{w}$. D. Krieger and J. Shallit [1] have proved the following result:

Theorem 2. *For any real $\alpha > 1$ there exists an infinite sequence* (*in some alphabet*) *that has critical exponent α.*

Informally speaking, when constructing such a sequence, we need to achieve two goals. First, we have to guarantee (for rational numbers r less than α but arbitrarily close to α) that our sequence contains r-powers; second, we have to guarantee that it does not contain q-powers for $q > \alpha$. Each goal is easy to achieve when considered separately. For the first one, we can just insert some r-power for every rational $r < \alpha$. For the second goal we can use the sequence with complex substrings: since every q-power has complexity about $1/q$ of its length (the number of free bits in it), Levin's sequence does not contain long q-powers if $q > 1/\gamma$. (Due to the asymptotic nature of Kolmogorov complexity, this guarantees only the absense of sufficiently long powers; short powers should be treated separately by extending the alphabet. It is easy to see that the alphabet size necessarily tends to infinity as $\alpha \to 1$.)

The real problem is to combine these two goals. We use the following scheme. First we fix some repetition pattern to ensure the first requirement (i.e., decide which bits in a sequence coincide). After that we choose the values of the "free" bits (each free bit appears in the sequence several times according to the repetition pattern) in such a way that no other (significant) repetitions arise. To implement this scheme, let us first prove some general statement about Kolmogorov complexity of subsequences in the case when some bits are repeated.

3 Complexity for Sequences with Repetitions

Let $\sim$ be an equivalence relation on $\mathbb{N}$. We assume that all equivalence classes are finite and the relation itself is computable; moreover, we assume that for a given $x \in \mathbb{N}$ one

can effectively list x's equivalence class. This relation is used as a repetition pattern: we consider only sequences $\mathbf{w}$ that follows $\sim$, i.e., only sequences $\mathbf{w}$ such that $w_i = w_j$ if $i \sim j$. For any set $A \subset \mathbb{N}$ we consider the *number of free bits in* A, i.e., the number of equivalence classes that have a non-empty intersection with A; it is denoted $\#_f A$ in the sequel.

There are countably many equivalence classes. Let us assign natural numbers to them (say, by listing them in the increasing order of minimal elements) and let $c(i)$ be the index of the equivalence class that contains i. Then every sequence $\mathbf{w}$ that follows the repetition pattern $\sim$ has the form $w_i = v_{c(i)}$ for some sequence $\mathbf{v} = v_0 v_1 v_2 \ldots$.

Theorem 3. *Let $\sim$ be an equivalence relation on $\mathbb{N}$ (as explained above) and let $\gamma \in (0,1)$ be a real number. There exists a sequence $\mathbf{w}$ that follows the pattern $\sim$ and an integer c with the following properties: for every finite set A with $\#_f A \geqslant c$ there exists $t \in A$ such that*

$$K(\mathbf{w}(A)|t) \geqslant \gamma \cdot \#_f A - K(A|t) - \log m(t)$$

where $m(t)$ is the "multiplicity" of t, i.e., the number of bits in its equivalence class.

(Note that if all equivalence classes are singletons, then $\log m(t)$ disappears, $\#_f A$ is the cardinality of A and we get an already mentioned corollary.)

Proof. Let $w_i = v_{c(i)}$ where $\mathbf{v} = v_0 v_1 \ldots$ is a sequence that satisfies the statement of Theorem 1 (with the same γ). For any finite $A \subset \mathbb{N}$ let $B \subset \mathbb{N}$ be the set of all $c(i)$ for $i \in A$. Then $\#B = \#_f A$. Theorem 1 guarantees that $K(B, \mathbf{v}(B)|u) \geqslant \gamma \cdot \#B$ for some $u \in B$. Since $u \in B$, there exists some $t \in A$ such that $c(t) = u$. To specify t when u is known, we need $\log m(t)$ bits, so $K(t|u) \leqslant \log m(t) + O(1)$. After t is known, we need $K(A|t)$ additional bits to specify A and $K(\mathbf{w}(A)|t)$ bits to specify $\mathbf{w}(A)$. Knowing A and $\mathbf{w}(A)$, we then reconstruct B and $\mathbf{v}(B)$. Therefore,

$$\gamma \cdot \#B \leqslant K(B, \mathbf{v}(B)|u) \leqslant \log m(t) + K(A|t) + K(\mathbf{w}(A)|t) + O(1),$$

which implies the desired inequality (with additional term $O(1)$, which can be compensated by a small change in γ).

4 Construction

We need to prove (for any real $\alpha > 1$) the existence of a sequence that has fractional powers of exponents less than α (and arbitrarily close to α) but does not have fractional powers of exponents greater than α. We start with an easier task. Assume that two real numbers α and β such that $1 < \alpha < \beta$ are given. Let us prove first that there exists a binary sequence $\mathbf{w}$ that contains fractional powers of exponents smaller than α and arbitrarily close to α but does not contain *long* fractional powers of exponents greater than β. (Later we make two improvements: first, we assure that the sequence has no fractional powers of exponent greater than β and any length; this is achieved by extending the alphabet. Second, we show how to make α and β equal.)

To construct such a sequence, let $r_1, r_2, \ldots$ be a sequence of rational numbers between 1 and α that converges to α. For each $r_i = p_i/q_i$ we insert a fractional power of exponent

r_i in the sequence: we select some interval of length p_i and decide that this interval should be a fractional power of some string of length q_i (and exponent r_i). This means that we declare two indices in this interval equivalent if they differ by a multiple of q_i. (The intervals for different i are disjoint.) We call these intervals *active intervals*. We assume that distance between two active intervals is much bigger than the lengths of both intervals (see below why this is useful and how long this distance should be).

Fig. 1. Two fractional powers of exponent r_1 and r_2 are implanted; y_i is a prefix of x_i (in this example the exponents are less than 2, so only one full period is shown)

Evidently, any sequence that follows this repetition pattern has critical exponent at least α.

Let us choose some γ between α/β and 1 ($\alpha/\beta < \gamma < 1$). and apply Theorem 3 with this γ to the pattern explained above. We get a bit sequence; let us prove that it does not contain *long* fractional powers of exponent greater than β. Indeed, it is easy to see that density of free bits in this pattern is at least $1/\alpha$, i.e., for any interval A of length l the number of free bits in it, $\#_f A$, is at least l/α. Indeed, if A intersects two or more active intervals, then all bits between them are free, and the distance between the intervals is large compared to interval sizes. Then we may assume that A intersects only one active interval. All subintervals of the active interval have the same repetitions period, and the density of free bits is minimal when A is maximal, i.e., coincides with the entire active interval. The bits outside the active interval are free (no equivalences), so they can only increase the fraction of free bits.

On the other hand, a fractional power of exponent β and length l has complexity at most $l/\beta + O(\log l)$ (we specify the length of the string and l/β bits that form the period). For long enough strings we then get a contradiction with the statement of Theorem 3 since $\alpha/\beta < \gamma$. Indeed, the inequalities

$$K(\mathbf{w}(A)|t) \geqslant \gamma \cdot \#_f A - K(A|t) - \log m(t) \geqslant (\gamma/\alpha) \cdot \#A - K(A|t) - O(1)$$

(note that the multiplicities are bounded) and

$$K(\mathbf{w}(A)) \leqslant \#A/\beta + O(\log \#A)$$

(here A is an interval) are inconsistent for long intervals A since $\gamma/\alpha > 1/\beta$.

Now we get rid of short fractional powers of exponent greater than β. For this we add an additional layer of symbols. i.e., extend our alphabet from $\{0,1\}$ to $\{0,1\} \times \Sigma$ for large Σ and take a Cartesian product of our sequence and some auxiliary one. Roughly speaking, we need an auxiliary sequence that follows (almost) the same repetition pattern but has no other repetitions (not prescribed by the pattern) on short distances. More formally, we may assume without loss of generality that q_i is a multiple of $i!$. Then we consider a periodic sequence with any large period m made of m different letters; adding it will destroy all periods that are not multiple of m, including all short periods,

but only finitely many of q_i (and the latter does not change the critical exponent). The Cartesian product of these two sequences (ith letter is a pair formed by ith letters of both sequences) has critical exponent between α and β.

In fact, we get even a stronger result:

Theorem 4. *For any α and β such that $1 < \alpha < \beta$ there exist a sequence* **w** *that has fractional powers of exponents r for $r < \alpha$ arbitrarily close to α but does not have "approximate fractional powers" of exponent β or more: there exists some $\varepsilon > 0$ such that any substring of length n is εn-far from any fractional power in terms of Hamming distance (we need to change at least εn symbols of the sequence to get a fractional power of length n).*

Indeed, a change of an ε-fraction of all the bits in a sequence of length n increases its complexity at most by $H(\varepsilon)n + O(\log n)$ where

$$H(\varepsilon) = -\varepsilon \log \varepsilon - (1-\varepsilon)\log(1-\varepsilon)$$

is Shannon entropy function. Note that $H(\varepsilon) \to 0$ as $\varepsilon \to 0$. Therefore, we need to change a constant fraction of bits to compensate for the difference in complexities (between the lower bound guaranteed by Theorem 3 and the upper bound due to approximate periodicity). □

5 Critical Exponent: Exact Bound

The same construction (with some refinement) can be used to get a sequence with a given critical exponent.

Theorem 5. (*Krieger – Shallit*) *For any real number $\alpha > 1$ there exists a sequence that has critical exponent α.*

(This proof follows the suggestions of D. Krieger who informed the author about the problem and suggested to apply Theorem 1 to it. See [1] for the original proof. The author thanks D. Krieger for the explanations and both authors of [1] for the permission to cite their paper.)

Again, let us consider repetition pattern that guarantees all exponents less than α and apply Theorem 3 with some γ close to 1. This (as we have seen) prevents powers with exponents greater that α/γ; the problem is how to get rid of intermediate exponents.

To do this, we should distinguish between two possibilities: (a) an unwanted power is an extension of the prescribed one (has the same period that unexpectedly has more repetitions) and (b) an unwanted power is not an extension. The first type of unwanted powers can be prevented by adding brackets around each active interval (in an additional layer: we take a Cartesian product of the sequence and this layer).

It remains to explain why unwanted repetitions of the second type do not exist (for γ close enough to 1). Consider any fractional power with exponent greater than α. There are two possibilities:

(1) It intersects at least two active intervals. Then it contains all free bits between these intervals, and (since we assume that the distances are large compared to the length of intervals) the density of free bits is close to 1, so exponent greater than α is impossible.

(2) It intersects only one active interval. The same argument (about density of free bits) shows that if the endpoints of this fractional power deviate significantly from the endpoints of the active interval, then the density of free bits is significantly greater than $1/\alpha$ and we again get a contradiction. Therefore, taking γ close to 1 we may guarantee that the distance between endpoints of the fractional power and endpoints of the corresponding active interval is a small fraction of the length of the active interval. Then we get *two different periods* in the region that is the intersection of fractional power and active interval. One ("old") is inherited from the repetition pattern; the second one ("new") is due to the fact that we consider a fractional power. (The periods are different, otherwise we are in the case (1).) The period lengths are close to each other. Indeed, if the new period is significantly longer, then the exponent is less than α; if the new period is significantly shorter, then the complexity bound decreases and we again get a contradiction.

Now note that two periods t_1 and t_2 in a string guarantee the period $t_1 - t_2$ near the endpoints of this string (at the distance equal to the difference between string length and minimal of these periods). Therefore we get a period that is a small fraction of the string length at an interval whose length is a non-negligible fraction of the string length. This again significantly decreases the complexity of the string, and this contradicts the lower bound of the complexity.

The short fractional powers are prevented exactly as explained above, by an additional layer with a periodic sequence with large enough period made of different letters. □

Remark. This proof uses some parameters that have to be chosen properly. For a given α we choose γ that is close enough to 1 and makes the arguments about "sufficiently small" and "significantly different" things in the last paragraph valid for long strings. Then we choose the repetition pattern where length of active intervals are multiples of factorials and the distances between them grow much faster than the lengths of active intervals. Then we apply Theorem 3 for this pattern. Finally, we look at the length c provided by this theorem and prevent all shorter periods by an additional layer. Another layer is used for brackets. These layers destroy unwanted short periods but only finitely many of prescribed patterns.

Note also that the proof is not really "constructive": though a repetition pattern (equivalence relation) can be explicitly described, the values of free bits are chosen in a non-constructive way.

Acknowlegments. The author thanks the anonymous referee for an extremely careful reading and suggesting many improvements.

References

1. Krieger, D., Shallit, J.: Every real number greater than 1 is a critical exponent, Preprint (2007)
2. Yu. Rumyantsev, A., Ushakov, M.A.: Forbidden Substrings, Kolmogorov Complexity and Almost Periodic Sequences. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 396–407. Springer, Heidelberg (2006)
3. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, N.Y (1997)
4. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press, New York (1995)

Generic Complexity of Presburger Arithmetic

Alexander N. Rybalov*

Omsk Branch of the Institute of Mathematics of SB RAS,
Russia, Omsk 644099, Pevtsova, 13

Abstract. Fischer and Rabin proved in [4] that Presburger Arithmetic has at least double exponential worse-case complexity. In [6] a theory of generic-case complexity was developed, where algorithmic problems are studied on "most" inputs instead of all set of inputs. An interesting question rises about existing of more efficient (say, polynomial) generic algorithm deciding Presburger Arithmetic on some "large" set of closed formulas. We prove, however, that there is no even exponential generic algorithm working correctly on arbitrary "very large" sets of inputs (so-called strongly generic sets).

1 Introduction

In [6] a theory of generic-case complexity was developed. Generic-case approach considers an algorithmic problem on "most" of inputs instead of all domain and ignores its behaviour on the rest of inputs.

This approach is close to well-known average-case complexity (see survey [2]) traditionally used in cryptography, where cryptosystems must be based on problems from NP which are hard on almost all (random) instances. Moreover, as noted in [6], a polynomially decidable on average problem (with some additional condition – so called polinomiality on μ-average on spheres) will be generically polynomially decidable. On the other hand in [7] it was considered when generic feasibility of a problem in NP implies its feasibility on average.

But these two approaches are not equivalent. Generic-case complexity can be applicable to undecidable problems as well as to decidable problems while average case complexity studies only decidable problems. For example, in [5] it was shown that classical undecidable Halting Problem for Turing machines with one-way tape is decidable (even quickly decidable) on a set of almost all instances with respect to some natural measure of Turing machines. Moreover it is known that there are NP-complete on average problems which are generically easily decidable. On the other hand in [7] examples of problems are given which are generically hard but easy on average.

Generic complexity is very close in spirit to errorless heuristic case complexity [2]. Like the heuristic algorithms, generic algorithms can output incorrect answers on rare inputs. But generic algorithms must detect when an answer is wrong (see definitions below). In fact, any generic algorithm is a heuristic algorithm that can control its correctness.

* Supported by the Russian Foundation for Basic Research, grant 07-01-00392.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 356–361, 2007.

In this paper we prove that Presburger Arithmetic remains generically hard on "very large" sets of formulas (so-called strongly generic sets – see definition below) with respect to a natural representation of closed formulas by labelled trees.

Now we give basic definitions of generic-case complexity from [6]. Let I be the set of all inputs and I_n be the set of all inputs of size n (sphere of radius n). For a subset $S \subseteq I$ define the following sequence

$$\rho_n(S) = \frac{|S \cap I_n|}{|I_n|}, \ n = 1, 2, 3, \ldots$$

The value $\rho_n(S)$ is probability to generate an input from S during random and uniform generation of inputs from sphere I_n. The asymptotic density of S is the following limit (if it exists)

$$\rho(S) = \lim_{n \to \infty} \rho_n(S).$$

S is called *generic* if $\rho(S) = 1$ and *negligible* if $\rho(S) = 0$. Clearly, S is generic if and only if its complement in I is negligible.

Following [6] we call a set S *strongly negligible* if sequence $\rho_n(S)$ exponentially fast converges to 0, i.e. there are constants $0 < \sigma < 1$ and $C > 0$ such that for every n

$$\rho_n(S) < C\sigma^n.$$

Now S is called *strongly generic* if its complement is strongly negligible.

A set $S \subseteq I$ is *generically decidable (within polynomial time, exponential time, etc.)* if there exists a set $G \subseteq I$ such that

1. G is generic,
2. G is decidable (within polynomial, exponential time, etc.),
3. $S \cap G$ is decidable (within polynomial, exponential time, etc.).

If G is strongly generic, then S is called *strongly generically decidable (within polynomial, exponential time, etc.)* A generic algorithm $\mathcal{A}$ for S works on an input $x \in I$ in the following way. At first $\mathcal{A}$ decides whether $x \in G$. If $x \in G$ then $\mathcal{A}$ can decide S on G, else $\mathcal{A}$ says "I don't know". So $\mathcal{A}$ correctly decides S on "almost all" inputs (inputs from generic set).

2 Representation of Formulas

Recall that Presburger Arithmetic is the first-order theory of structure $\langle \mathbb{N}, + \rangle$. We will consider only closed formulas Φ (further just formulas) satisfying the following conditions

1. Φ has the natural prenex form, i.e.

$$\Phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi,$$

where $Q_i \in \{\exists, \forall\}$ and ϕ is a quantifier-free formula;

2. quantifier-free part ϕ of formula Φ is a Boolean combination of conjunctions and disjunctions of the following simple atomic formulas

   (a) $x_i = x_j$,
   (b) $x_i \neq x_j$,
   (c) $x_i = x_j + x_k$,
   (d) $x_i \neq x_j + x_k$.

It is known that any formula over $\langle \mathbb{N}, + \rangle$ can be effectively transformed to equivalent formula of such form.

Let Φ be a formula with quantifier-free part ϕ. One can naturally associate with ϕ a binary tree T_ϕ that presents the construction of ϕ from simple atomic formulas by means of disjunctions and conjunctions. The internal vertices of T_ϕ are labeled by symbols $\vee$ and $\wedge$, and the leafs of T_ϕ are labeled by simple atomic formulas. Conversely, given a binary tree T like that one can uniquely reconstruct a quantifier-free formula ϕ_T of the form described above. This gives a one-to-one representation of the quantifier free formulas ϕ by the binary trees T_ϕ. If T_ϕ has n leafs then at most $3n$ variables may occur in T_ϕ, so we may assume from the beginning that all variables in T_ϕ belong to the set $x_1, \ldots, x_{3n}$ and ϕ depends on these $3n$ variables (some of them may be fictitious).

Now a *representation* of the formula Φ consists of a binary tree T_ϕ and a quantifier prefix on all $3n$ variables of ϕ. We will identify formula Φ with its representation. The *size* of Φ is the number of leafs in the tree T_ϕ.

Denote the set of all formulas by $\mathcal{F}$ and the set of all formulas of size n by $\mathcal{F}_n$.

Lemma 1

$$|\mathcal{F}_n| = 2^{5n-1}(9n^2 + 27n^3)^n C_n,$$

where $C_n = \frac{1}{n+1}\binom{2n}{n}$ *— n-th Catalan number.*

Proof. Any formula of size n consists of a quantifier prefix of length $3n$ and a binary tree with n leafs and $n-1$ internal vertices. There are 2^{3n} variants to choose a quantifier prefix. There are C_n non-labeled binary trees with n leafs, where

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

is n-th Catalan number. Every internal vertex of the tree can be labeled by $\vee$ or by $\wedge$, and every leaf can be labeled by any simple atomic formula of $3n$ variables (there are exactly $2((3n)^3 + (3n)^2)$ such formulas). So we have

$$2^{3n} \times 2^{n-1} \times 2^n \times (9n^2 + 27n^3)^n C_n$$

different formulas of size n. □

For any formula Φ define the sets

$$AND(\Phi) = \{\Phi \wedge \Psi,\ \Psi \text{ — arbitrary formula}\},$$

$$OR(\Phi) = \{\Phi \vee \Psi,\ \Psi \text{ — arbitrary formula}\},$$

and the sets

$$AND(\Phi)^+ = \{\Phi \wedge \Psi,\ \Psi \text{ — arbitrary true formula}\},$$

$$OR(\Phi)^- = \{\Phi \vee \Psi,\ \Psi \text{ — arbitrary false formula}\}.$$

Lemma 2. *For any Φ sets $AND(\Phi)^+$ and $OR(\Phi)^-$ are not strongly negligible. Moreover, there is a constant $C > 0$ such that*

$$\frac{|AND(\Phi)^+ \cap \mathcal{F}_n|}{|\mathcal{F}_n|} > \frac{C}{(16n)^{3k}}$$

for any $n > k$, where k is the size of Φ. The same bound is true for the set $OR(\Phi)^-$.

Proof. We will prove it for $AND(\Phi)^+$, for $OR(\Phi)^-$ it is proving analogously. Let a formula Φ has a size k. Fix a size $n > k$. Consider all formulas of the following form

$$Q_1 x_1 \dots Q_{3n} x_{3n} (\varphi \wedge \psi), \tag{1}$$

where φ is the quantifier-free part of Φ with $x_1, \dots, x_{3k}$ are replaced by $3k$ variables from the set $x_1, \dots, x_{3n}$, and quantifiers on these variables are the same as in Φ, and ψ — arbitrary quantifier-free formula of size $n - k$ not containing variables from φ. Since sets of variables of φ and ψ are different, we can rewrite every formula of type (1) as $\Phi \wedge \Psi \in AND(\Phi)$. Besides size of formula (1) is n. So the set of such formulas S is a subset of $AND(\Phi)_n$.

The number of formulas of type (1) is the number of all possible quantifier-free parts ψ and quantifier prefixes on $3(n-k)$ variables of ψ (quantifiers on variables of φ are fixed). It is can be shown analogously to proof of Lemma 1 that

$$|S| = 2^{5n-4k-1}(9n^2 + 27n^3)^{n-k} C_{n-k}.$$

Bound the $\rho_n(AND(\Phi)_n)$:

$$\frac{|AND(\Phi)_n|}{|\mathcal{F}_n|} \geq \frac{|S \cap AND(\Phi)_n|}{|\mathcal{F}_n|} = \frac{|S|}{|\mathcal{F}_n|} =$$

$$= \frac{2^{5n-4k-1}(9n^2 + 27n^3)^{n-k} C_{n-k}}{2^{5n-1}(9n^2 + 27n^3)^n C_n} =$$

$$= \frac{C_{n-k}}{2^{4k}(9n^2 + 27n^3)^k C_n} > \frac{1}{(8n)^{3k}} \times \frac{C_{n-k}}{C_n}.$$

Since

$$\frac{C_{n-k}}{C_n} = \frac{n+1}{n-k+1} \times \frac{\binom{2(n-k)}{n-k}}{\binom{2n}{n}} > \frac{n!}{(n-k)!} \cdot \frac{2(n-k)\dots(n-k+1)}{2n\dots(n+1)} =$$

$$= \frac{(n\dots(n-k+1))^2}{2n\dots(2(n-k)+1)} > \left(\frac{n\dots(n-k+1)}{2n\dots(2n-k+1)}\right)^2 > \frac{1}{2^{2k}}$$

we have the following lower bound

$$\frac{|AND(\Phi)_n|}{|\mathcal{F}_n|} > \frac{C}{(16n)^{3k}}$$

with some universal constant $C > 0$.

Note now that for every formula $\Phi \wedge \Psi \in AND(\Phi)_n$ the formula $\Phi \wedge \neg\Psi$ belongs to $AND(\Phi)_n$. Indeed, if

$$\Psi = Q_1 x_{i_1} \dots Q_{3n} x_{i_{3(n-k)}} \psi(x_{i_1}, \dots, x_{i_{3(n-k)}}),$$

then

$$\neg\Psi = \overline{Q}_1 x_{i_1} \dots \overline{Q}_{3(n-k)} x_{i_{3(n-k)}} \neg\psi(x_{i_1}, \dots, x_{i_{3(n-k)}}),$$

where $\overline{\exists} = \forall$ and $\overline{\forall} = \exists$. A tree for $\neg\psi$ can be got from a tree for ψ by replacing of $\wedge$ and $\vee$ in the internal vertices, and by replacing of simple atomic formulas on its negations in the leafs. The size of the tree remains the same. It means that for every formula from $AND(\Phi)_n^+$ there is a unique formula from $AND(\Phi)_n^-$, where

$$AND(\Phi)^- = \{\Phi \wedge \Psi,\ \Psi \text{ — arbitrary false formula}\},$$

and overwise. So

$$|AND(\Phi)_n^+| = \frac{1}{2}|AND(\Phi)_n|.$$

This implies the lower bound on $\frac{|AND(\Phi)_n^+|}{|\mathcal{F}_n|}$ which we need. □

3 Main Result

To prove the main result we construct a generic reduction that transforms any exponential-time algorithm correctly deciding PA on most formulas into an exponential-time algorithm that decides PA correctly on every formula. Such type of reductions for problems in NP and polynomial algorithms was considered in average-case complexity, for example [1], [3].

Theorem 1. *There is no strongly generic set of formulas on which Presburger Arithmetic is decidable in exponential time.*

Proof. Suppose we have a generic exponential algorithm $\mathcal{A}$ deciding Presburger Arithmetic on some strongly generic set of formulas S (recognizable in exponential time). There are constants $D > 0$ and $\alpha > 0$ such that

$$\frac{|(\mathcal{F} \setminus S) \cap \mathcal{F}_n|}{|\mathcal{F}_n|} < \frac{D}{2^{\alpha n}}$$

for every n. We can find a polynomial $p()$ such that

$$\frac{C}{(16n)^{3k}} > \frac{D}{2^{\alpha n}}$$

for all $n \geq p(k)$. Here constant C is from Lemma 2.

Now we can design an algorithm $\mathcal{B}$ working in time $2^{q(n)}$ with some polynomial $q(n)$ and deciding Presburger Arithmetic on all set of formulas, in contrary to the result of Fischer and Rabin. This algorithm $\mathcal{B}$ works on a formula Φ in the following way. For every formula from the sets $AND(\Phi)_n$ and $OR(\Phi)_n$, where $n = p(k)$ and k is the size of Φ, algorithm $\mathcal{B}$ checks whether it belongs to S. It can be done in time $2^{r(k)}$, where $r()$ is some polynomial, by Lemma 1. If some disjunction $\Phi \vee \Psi$ belongs to S, then $\mathcal{B}$ launches $\mathcal{A}$ on it. If $\Phi \vee \Psi$ is false, then Φ is false too and $\mathcal{B}$ outputs the correct answer. Analogously, if some conjunction $\Phi \wedge \Psi$ hits in S and it is true, then Φ is true too and $\mathcal{B}$ can determine this by using of $\mathcal{A}$.

Prove that the described algorithm works correctly on any formula Φ. Suppose Φ is true. By Lemma 2 and by choice of the polynomial $p()$ in the set $AND(\Phi)^+_n$ there is a conjunction from S on which $\mathcal{A}$ can decide its truthness. Analogously the case of false Φ is considered. □

References

1. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. SIAM Journal on Computing 13, 850–864 (1984)
2. Bogdanov, A., Trevisan, L.: Average-Case Complexity. Electronic Colloquium on Computational Complexity, Report No. 73 (2006)
3. Bogdanov, A., Trevisan, L.: On Worst-Case to Average-Case Reductions for NP Problems. In: Proceedings of 44th FOCS, pp. 308–317. IEEE, Los Alamitos (2003)
4. Fischer, M.J., Rabin, M.O.: Super-Exponential Complexity of Presburger Arithmetic. In: Proceedings of the SIAM-AMS Symposium in Applied Mathematics, vol. 7, pp. 27–41 (1974)
5. Hamkins, J.D., Miasnikov, A.: The halting problem is decidable on a set of asymptotic probability one. Notre Dame Journal of Formal Logic 47(4), 515–524 (2006)
6. Kapovich, I., Myasnikov, A., Schupp, P., Shpilrain, V.: Generic-case complexity, decision problems in group theory and random walks. J. Algebra 264(2), 665–694 (2003)
7. Kapovich, I., Myasnikov, A., Schupp, P., Shpilrain, V.: Average-case complexity for the word and membership problems in group theory. Advances in Mathematics 190, 343–359 (2005)

Everywhere α-Repetitive Sequences and Sturmian Words

Kalle Saari*

Department of Mathematics and Turku Centre for Computer Science
University of Turku
20014 Turku, Finland
kasaar@utu.fi

Abstract. Local constraints on an infinite sequence that imply global regularity are of general interest in combinatorics on words. We consider this topic by studying *everywhere α-repetitive sequences*, sequences in which every position has an occurrence of a repetition of order $\alpha \geq 1$ of bounded length. The number of minimal such repetitions, called *minimal α-powers*, is then finite. A natural question regarding global regularity is to determine the least number of minimal α-powers such that an α-repetitive sequence is not necessarily ultimately periodic. We solve this question for $1 \leq \alpha \leq 17/8$. We also show that Sturmian words are among the optimal 2- and 2^+-repetitive sequences.

Keywords: Everywhere α-repetitive sequence, α-power, squareful sequence, overlapful sequence, Sturmian word.

1 Introduction

Let us start with the following observation: Each position in the Fibonacci word

$$\mathbf{f} = 01001010010010100101001001010010010100101001001010010100101\ldots$$

starts a square. More precisely, each position starts a square of period at most 5. The squares are

$$00, \quad 0101, \quad 010010, \quad 1010, \quad 100100, \quad 1001010010.$$

Notice that the number of the minimal squares is 6. In fact, all Sturmian words start a square at each position, and the number of minimal squares always equals 6, which we will show in this paper. What happens if a sequence with squares in every position has at most five minimal squares? We will show that the sequence is then ultimately periodic. Therefore Sturmian words have optimal properties also in this sense.

Relations between local regularity and global regularity in infinite sequences have been considered from various perspectives [5, Ch. 8]. A fundamental result

* Supported by the Finnish Academy under grant 8206039.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 362–372, 2007.

by Mignosi, Restivo, and Salemi [6] characterizes ultimately periodic sequences by considering left repetitions at each position. They proved that a sequence is ultimately periodic if and only if each sufficiently long prefix of the sequence contains a repetition of order $1+\varphi$ as a suffix, where φ denotes the golden ratio $(1+\sqrt{5})/2$. Karhumäki, Lepistö, and Plandowski [4] considered the same situation, but with an additional condition imposed to the length of the repetitions in question. They called a sequence (ρ, l)-*repetitive*, where $\rho > 1$ is real and $l \geq 1$ an integer, if all sufficiently long prefixes of the sequence have a suffix of the form v^σ with $|v| \leq l$ and $\sigma \geq \rho$. They showed that a $(2,4)$-repetitive sequence is ultimately periodic, while there exist aperiodic $(2,5)$-repetitive sequences. Here we say that a sequence is *aperiodic* if it is not ultimately periodic.

In this paper we investigate local and global regularity not by restricting the length of the shortest repetitions, but instead by restricting the number of distinct such repetitions. Also, as with the example in the beginning of this paper, we consider right repetitions, instead of left repetitions. Unlike with left repetitions, the mere existence of right repetitions of large order occurring at each position does not guarantee ultimate periodicity. Indeed, it is not difficult to construct a sequence with, say, cubes starting from every position that is not ultimately periodic. However, if the number of distinct shortest repetitions is bounded, global regularity emerges. For example, if a sequence has squares occurring at every position and there are only five different minimal squares, then the sequence is ultimately periodic. This is the observation that gave rise to the notion of an *everywhere* α*-repetitive* sequence, a sequence whose every position starts an α-power and the number of distinct minimal α-powers occurring in the sequence is finite (see the definition of a minimal α-power in the next section). A little modification of a fundamental result by Mignosi et al. [6] reveals that any everywhere α-repetitive sequence with $\alpha \geq 1+\varphi$ is ultimately periodic, see Theorem 1. For the sake of brevity, we will sometimes write "α-repetitive" instead of "everywhere α-repetitive."

The notion of an everywhere α-repetitive sequence can be approached from two perspectives. One perspective is to fix an infinite sequence, and ask whether it is α-repetitive. It is trivial that any sequence is α-repetitive for $\alpha = 1$. Therefore we may consider the supremum of all real numbers $\alpha \geq 1$ such that the sequence is everywhere α-repetitive. If the sequence is aperiodic, then the supremum of such numbers is at most $1+\varphi$. Given a real number α strictly less than the supremum, the set of minimal α-powers that are factors of the sequence is finite. This leads to the question of how many minimal α-powers there are in the sequence.

The other perspective is to fix a real number $\alpha \geq 1$, and consider the sequences that are everywhere α-repetitive. In particular, what is the smallest possible number of minimal α-powers in an *aperiodic* everywhere α-repetitive sequence? Denote this value by $M(\alpha)$. For instance, by what was said in the beginning of this introduction, we have $M(2) = 6$. Another question then arises: What is the structure of aperiodic everywhere α-repetitive sequences with precisely $M(\alpha)$ minimal α-powers? We characterize such sequences for $\alpha = 2$.

In this paper we study these questions as follows. In Section 2, we present some auxiliary definitions and results. In Section 3, we show that the Thue–Morse word is everywhere 5/3-repetitive, and that the 5/3 is optimal. In Section 4, we consider everywhere 2-repetitive sequences, which we call *squareful*. We show that an aperiodic squareful sequence must have at least six minimal squares, and we characterize the structure of the aperiodic squareful sequences with exactly six minimal squares. We show that Sturmian words are among these sequences. We also mention some corresponding results in the case of 2^+-repetitive sequences, but they are proved only in the full version of this extended abstract. In Section 5, we determine the values $M(\alpha)$ for $1 \leq \alpha \leq 17/8$.

2 Preliminaries

Here we present the necessary definitions and notations, as well as two auxiliary results. The reader should consult [3] or [5] for any notions left undefined.

Let $\alpha \geq 1$ be a real number. A word w is called an *α-power* if

$$\frac{|w|}{p(w)} \geq \alpha, \tag{1}$$

where $|w|$ denotes the length of w and $p(w)$ denotes the least period of w. The word w is called an *α^+-power* if the inequality in (1) is strict. For example, the word 010101 is a 3-power, and in fact, an α-power for any $1 \leq \alpha \leq 3$. Furthermore, it is an α^+-power for $1 \leq \alpha < 3$. With this terminology, a word is a 2^+-power precisely when it is an overlap.

The word w is a *minimal* α-power (resp. α^+-power) if w itself is an α-power (resp. α^+-power) and no proper prefix of w is an α-power (resp. α^+-power). The word 010101 is an α-power, but not minimal, for $1 \leq \alpha \leq 5/2$, because 01010 is a 5/2-power. It is minimal for $5/2 < \alpha \leq 3$, however.

Let z denote an infinite sequence. We say that z is an *everywhere α-repetitive* sequence if the following condition holds: Each position in z has an occurrence of an α-power, and the set of minimal α-powers that are factors of z is finite. Equivalently, the sequence z is everywhere α-repetitive if and only if there exists an integer $N \geq 1$ such that any factor of z of length N has a prefix that is an α-power. Trivially, any infinite word is everywhere 1-repetitive. If a sequence is uniformly recurrent, it is everywhere α-repetitive for some $\alpha > 1$. Any purely periodic word is everywhere α-repetitive for all $\alpha \geq 1$. The sequence z is called *everywhere α^+-repetitive* if it is everywhere $\alpha + \epsilon$-repetitive for some $\epsilon > 0$. Two special cases deserve more succinct names: We call everywhere 2-repetitive sequences *squareful*, and everywhere 2^+-repetitive sequences *overlapful*. Again, we sometimes call everywhere α-repetitive sequences just α-repetitive.

We use two infinite fixed points of morphisms in this paper. The Fibonacci word, denoted by $\mathbf{f}$, is the fixed point of the morphism $f\colon 0 \mapsto 01, 1 \mapsto 0$. The "ubiquitous" Thue–Morse word [2] is the infinite fixed point of the morphism

$\mu\colon 0 \mapsto 01, 1 \mapsto 10$ that begins with the letter 0. We denote it by $\mathbf{t}$. Hence,

$$\mathbf{f} = \lim_{n\to\infty} f^n(0) = 0100101001001010010100100101001001010010100101001\ldots;$$
$$\mathbf{t} = \lim_{n\to\infty} \mu^n(0) = 0110100110010110100101100110100110010110011 0\ldots.$$

Note that we sometimes call a sequence an (infinite) word, and vice versa; both terms are commonly used in the literature.

The following theorem is a little modification of the results in [6]. Details can be found in the full version of this paper.

Theorem 1. *If an infinite sequence is everywhere α-repetitive with $\alpha \geq 1+\varphi$, then it is ultimately periodic. The Fibonacci word is α-repetitive for $\alpha < 1+\varphi$.*

For a real number $1 \leq \alpha < 1+\varphi$, we denote by $M(\alpha)$ the *least* positive integer k such that there exists an aperiodic everywhere α-repetitive sequence with k distinct minimal α-powers. By the previous theorem, the quantity $M(\alpha)$ always exists. An aperiodic α-repetitive sequence with exactly $M(\alpha)$ minimal α-powers is called *optimal*.

The next lemma may seem self-evident, but it is an essential fact that we will use later in this paper. It is proved in the full version of this paper.

Lemma 1. *For $1 \leq \alpha < 1+\varphi$, there exists an optimal α-repetitive sequence over a two-letter alphabet.*

If z is an infinite sequence, we let $P(z)$ denote the supremum of the real numbers $\alpha \geq 1$ for which the sequence z is everywhere α-repetitive. If z is purely periodic, then $P(z)$ clearly equals ∞. If z is aperiodic, then Theorem 1 implies that $1 \leq P(z) < 1+\varphi$. In the next section we will show that $P(\mathbf{t}) = 5/3$, where $\mathbf{t}$ denotes the Thue–Morse word.

Sturmian words have several equivalent definitions, but for our purpose the most useful definition is via the balance property: An aperiodic infinite word z over the alphabet $\{0,1\}$ is *Sturmian* if and only if, for all factors x and y of the same length, the number of letters 0 in x differs from the number of letters 0 in y by at most 1.

So-called characteristic words form a very important subclass of Sturmian words. They can be defined as follows. Let $(d_n)_{n\geq 1}$ be a sequence of integers with $d_1 \geq 0$ and $d_n \geq 1$ for $n \geq 2$. The sequence $(d_n)_{n\geq 1}$ is called *directive*. We define auxiliary words s_n by

$$s_{-1} = 1, \quad s_0 = 0, \qquad s_n = s_{n-1}^{d_n} s_{n-2} \quad (n \geq 1).$$

Then a *characteristic word* is the infinite word z such that s_n is a prefix of z for all $n \geq 1$.

Among aperiodic infinite words, Sturmian words have many optimal properties, for instance, in terms of subword complexity, or number of return words. In this paper, we will show that Sturmian words are optimal squareful and overlapful. For the many beautiful properties of Sturmian and characteristic words, we refer to [1,5].

3 The Thue–Morse Word Is Everywhere 5/3-Repetitive

To familiarize and motivate the notions defined in the previous section, we show here that the Thue–Morse word is everywhere 5/3-repetitive, and that the quantity 5/3 is optimal.

Theorem 2. *The Thue-Morse word is everywhere* 5/3*-repetitive. Its minimal* 5/3*-powers are listed below.*

$$00, 01001, 0101, 0110010110, 0110011, 01101,$$
$$11, 10110, 1010, 1001101001, 1001100, 10010.$$

Proof. It is readily verified that each of the words listed above is a 5/3-power. It can also be verified that each factor of length 10 of $\mathbf{t}$ has a prefix in the list above. Indeed, the function that, for all integers $n \geq 0$, gives the length of the shortest prefix of $\mathbf{t}$ that contains an occurrence of each factor of length n of $\mathbf{t}$ is known, see [1, Sec. 10.10]. Therefore we know that each factor of length 10 occurs in the prefix of length 57 of $\mathbf{t}$. This completes the proof.

Next proposition shows that the 5/3 in the previous proposition is optimal.

Theorem 3. *The Thue–Morse word is not everywhere* $(5/3)^+$*-repetitive. Therefore,*

$$P(\mathbf{t}) = \frac{5}{3}.$$

Proof. It suffices to show that the Thue–Morse word $\mathbf{t}$ does not have a $(5/3)^+$-power as a prefix. To do that, we assume the contrary. Let uu' be the shortest prefix of $\mathbf{t}$ such that u' is a prefix of u and

$$\frac{|uu'|}{|u|} > \frac{5}{3}.$$

Since the word 011 is a prefix of $\mathbf{t}$, we have $|u| \geq 3$. Since uu' is the shortest $(5/3)^+$-power that is a prefix of $\mathbf{t}$, it follows that u can be written in the form $u = \mu(x)1$, and u' can be written in the form $u' = 0\mu(y)$, where x and y are some finite words. The word u' is a prefix of u, so we either have that 011 is a prefix of u', or u' is a prefix of 01. In the first case, the word 11 is a prefix of $\mu(y)$, which is impossible for any word y. In the second case, we have

$$\frac{|uu'|}{|u|} \leq \frac{|u|+2}{|u|} \leq \frac{5}{3}.$$

This contradiction completes the proof.

4 Squareful Sequences

In this section, we consider some properties of optimal squareful sequences. We also state some corresponding results on optimal overlapful sequences.

4.1 The Number of Minimal Squares

In the next two lemmas we assume that z is an aperiodic squareful sequence.

Lemma 2. *If the word uu is a minimal square in z, then there exists a minimal square vv, different from uu, in z such that u is a prefix of vv.*

Proof. Consider an occurrence of the word uu in z. The latter u in uu starts a minimal square in z, say xx. The minimality of uu imply that xx is not a prefix of u. Hence either $x = u$ or u is a prefix of xx. The sequence z is aperiodic, so for some occurrence of uu, the second case must hold.

Lemma 3. *For each letter b occurring in z, there exists at least three distinct minimal squares in z that start with the letter b.*

Proof. Let uu denote a shortest minimal square that starts with the letter b. Lemma 2 implies that there exists another minimal square vv such that u, and thus the letter b, is a prefix of vv. The choice of uu implies that u is actually a prefix of v, and hence we can write $v = ut$ for some nonempty word t.

To derive a contradiction, suppose that the two minimal squares uu and vv are the only ones starting with the letter b. Then there exists a position in z that has an occurrence of both words v and uu. Since vv is a minimal square, it follows that v is a prefix of uu, and further that t is a proper prefix of u. Hence we can write $v = ts$ for some nonempty word s. But now $vv = utts$, and we see that the word tt is a square in z. Since t is a prefix of u, it starts with the letter b. Therefore either the square tt or one of its prefixes is a minimal square starting with the letter b, and it is strictly shorter than uu, a contradiction.

Theorem 4. *Any aperiodic squareful sequence has at least six minimal squares, and six is the optimal lower bound.*

Proof. If a squareful sequence is aperiodic, it has at least two distinct letters, so Lemma 3 implies that it must have at least six distinct minimal squares. That the quantity six is optimal follows from the fact that the Fibonacci word is squareful, and it has precisely six minimal squares, as is readily verified by considering its eleven factors of length 10.

Note that, by the proof of the previous theorem, any optimal squareful sequence is over a two-letter alphabet.

The following theorem is proved in the full version of this paper.

Theorem 5. *An aperiodic overlapful sequence has at least twelve minimal overlaps, and twelve is the optimal lower bound.*

4.2 Characterization of Optimal Squareful Sequences

In what follows, the symbol z denotes an optimal squareful sequence; we may suppose that z consists of letters 0 and 1. Since z is aperiodic, it follows that

one of the letters, say 0, occurs in blocks of at least two distinct lengths. That is, there exist integers i and j with $j > i > 0$ such that the words 10^i1 and 10^j1 are factors of z. Suppose further that i and j are the least integers with these properties.

We start with a series of lemmas. We will often use Lemma 3 implying that both letters 0 and 1 correspond to exactly three distinct minimal squares in z.

Lemma 4. *The word* 11 *does not occur in* z.

Proof. There exists one minimal square with a prefix 10^i1. Since $j > i$, Lemma 2 implies that there exist two distinct minimal squares with a prefix 10^j. Therefore, the word 11 cannot occur in z because otherwise 11 would be a fourth minimal square in z with a prefix 1.

Lemma 5. *Let* $n \geq 1$ *be an integer. The word* 10^n1 *is a factor of* z *if and only if* n *equals either* i *or* j.

Proof. Suppose that, contrary to what we want to prove, the sequence z has a factor 10^n1 with $n \geq 1$ distinct from i and j. Then z has three distinct minimal squares with prefixes $10^i1, 10^j1$, and 10^n1. In addition, since $n > j > i$, Lemma 2 implies that there exists another minimal square with prefix 10^n. Hence we have altogether at least four minimal squares starting with the letter 1, a contradiction.

Lemma 6. *We have* $j = i + 1$.

Proof. The words 00 and $010^{i-1}010^{i-1}$ are minimal squares in z. There is also a third minimal square, denote it by uu, in z such that 010^{j-1} is a prefix of u. It follows from Lemma 2 that u, and hence also 010^{j-1}, is a prefix of $010^{i-1}010^{i-1}$. Since $j > i$, this is possible only if $j = i + 1$.

According to the previous lemma, we can write $i + 1$ in place of j.

Lemma 7. *There exist an integer* $k \geq 0$ *such that the word*

$$10^{i+1}(10^i)^n10^{i+1} \tag{2}$$

is a factor of z *if and only if* n *equals either* k *or* $k + 1$.

Proof. By the previous three lemmas, the suffix of z that starts from the first occurrence of the letter 1 in z can be factorized into words 10^i and 10^{i+1}. The aperiodicity of z implies, therefore, that the word of the form (2) occurs in z for at least two distinct values of n. Let k and m denote two such distinct integers with k the smallest possible. We will show that $m = k + 1$.

Observe first that the minimal squares

$$10^i10^i \qquad \text{and} \qquad 10^{i+1}(10^i)^k10^{i+1}(10^i)^k$$

are factors of z. Since the word

$$10^{i+1}(10^i)^m10^{i+1}$$

is a factor of z, it follows from Lemma 2 that there exist at least two minimal squares with a prefix $10^{i+1}(10^i)^m$. Since there exist only three minimal squares with prefix 1, the word $10^{i+1}(10^i)^m$ must be a prefix of

$$10^{i+1}(10^i)^k 10^{i+1}(10^i)^k.$$

Since $m > k$, this is possible only if $m = k + 1$. The proof is complete.

In the following theorem, the *root* of a square uu means the word u.

Theorem 6. *If a squareful sequence is optimal, then there exist integers $i \geq 1$ and $k \geq 0$ such that the roots of the minimal squares of the sequence are, up to renaming the letters,*

$$0, \quad 010^{i-1}, \quad 010^i, \quad 10^i, \quad 10^{i+1}(10^i)^k, \quad \textit{and} \quad 10^{i+1}(10^i)^{k+1}. \tag{3}$$

Proof. Let z denote the optimal squareful sequence we have been considering in this subsection. That the first four words listed in (3) are roots of minimal squares in z follows from the observation that the two words 010^i10^i and $010^{i+1}10^i$ are factors of z. Since the suffix of z starting from the first occurrence of the letter 1 in z can be factorized into words 10^i and 10^{i+1}, it follows from Lemma 7 that the minimal squares

$$10^{i+1}(10^i)^k 10^{i+1}(10^i)^k \qquad \text{and} \qquad 10^{i+1}(10^i)^{k+1}10^{i+1}(10^i)^{k+1}$$

both occur in z (the latter minimal square may be obtained in two ways). Therefore, the last two words listed in (3) are roots of minimal squares in z. Since z is optimal, there exist precisely six minimal squares in z, and so we have found all of them. The proof is complete.

Theorem 7. *Suppose that z is an aperiodic sequence. Then z is squareful and optimal if and only if, up to renaming letters, there exist integers $i \geq 1$ and $k \geq 0$ such that z is an element of the language*

$$0^*(10^i)^*\left(10^{i+1}(10^i)^k + 10^{i+1}(10^i)^{k+1}\right)^\omega. \tag{4}$$

Proof. If z is an optimal squareful sequence, then there exist integers $i \geq 1$ and $k \geq 0$ such that the roots of the minimal squares are the ones listed in (3). Write $z = uv$, where u and v are chosen so that the block 10^{i+1} occurs for the first time in z as a prefix of v. Then u is in the language $0^*(10^i)^*$. Since the word 10^{i+1} is a prefix of v, Lemma 7 implies that v factorizes over the words $10^{i+1}(10^i)^k$ and $10^{i+1}(10^i)^{k+1}$. Therefore, z is in the language (4).

Conversely, if z is an infinite word in the language (4), it is readily verified that the sequence z is squareful. Furthermore, z has exactly six minimal squares, their roots are the ones listed in (3), and so z is also optimal. This completes the proof.

4.3 Sturmian Words Are Optimal Squareful and Overlapful Sequences

Now we will show that Sturmian words are optimal squareful sequences. Note, however, that this property does not characterize Sturmian words, unlike most other well-known properties of Sturmian words.

Theorem 8. *Sturmian words are optimal squareful sequences.*

Proof. Let z denote a Sturmian word. Without loss of generality, we may suppose that the word 11 is not a factor of z. Let i denote the least integer j such that the word 10^j1 is a factor of z. Since 11 does not occur in z, we have $i \geq 1$. Furthermore, since z is balanced, it follows that the maximal integer j such that 10^j1 is a factor of z equals $i+1$. Therefore z is in the language $0^*(10^i + 10^{i+1})^\omega$

First, consider a position of z that is occupied by the letter 0. Since i is minimal, the letter 0 extends either to 00, to 010^i10^i, or to 010^i010^i. Hence, in z, each position occupied by the letter 0 starts a square, and there are at most three minimal squares with prefix 0.

Second, consider a position of z that is occupied by the letter 1. Now the letter 1 extends either to 10^i10^i or to $10^{i+1}1$. The first is a minimal square; the second extension has to be analyzed further.

Let k denote the least integer $n \geq 0$ such that the word

$$10^{i+1}(10^i)^n10^{i+1} \tag{5}$$

is a factor of z. Since z is balanced, the maximal integer n such that the word in (5) occurs in z equals $k+1$. Indeed, otherwise both words

$$0^{i+1}(10^i)^k10^{i+1} \qquad \text{and} \qquad (10^i)^{k+2}1$$

are factors of z, have the same length, but differ in the number of letters 1 by 2, which contradicts the balance property of z.

Hence the word $10^{i+1}1$ extends either to

$$10^{i+1}(10^i)^k10^{i+1}(10^i)^k \qquad \text{or to} \qquad 10^{i+1}(10^i)^{k+1}10^{i+1}(10^i)^{k+1}$$

(again, the latter word can be obtained in two ways). Both words are minimal squares. Hence each position occupied by the letter 1 starts a square, and there are at most three minimal squares with prefix 1.

We have shown that the Sturmian word z is squareful with at most six minimal squares. Since Sturmian words are aperiodic, Theorem 4 implies that z has exactly six minimal squares, and thus z is optimal. This completes the proof.

The following theorem is proved in the full version of this paper.

Theorem 9. *Sturmian words are optimal overlapful sequences.*

5 Values of the Function $M(\alpha)$

Recall that the symbol $M(\alpha)$ denotes the least integer $k \geq 1$ such that there exists an aperiodic everywhere α-repetitive sequence with k minimal α-powers. In this section, we establish the values of $M(\alpha)$ for $1 \leq \alpha \leq 17/8$. It is clear that $M(1) = 2$.

Theorem 10. *For $1 < \alpha \leq 3/2$, we have $M(\alpha) = 4$.*

Proof. Let $\alpha > 1$, and let z be an optimal α-repetitive sequence. By Lemma 1, we may assume that z is over the alphabet $\{0, 1\}$. Since z is aperiodic, it has at least two distinct minimal α-powers that start with the letter 0. The same holds true for the letter 1, and therefore $M(\alpha) \geq 4$.

Next suppose that $1 < \alpha \leq 3/2$. We will show that $M(\alpha) \leq 4$. Since any 3/2-repetitive sequence is also α-repetitive, it now suffices to find a 3/2-repetitive sequence with four minimal 3/2-powers. But the Fibonacci word $\mathbf{f}$ has this property. Indeed, each factor of $\mathbf{f}$ of length 5 has a 3/2-power as a prefix, which is readily verified; the minimal 3/2-powers are $00, 010, 10010, 101$. The proof is complete.

Theorem 11. *For $3/2 < \alpha < 2$, we have $M(\alpha) = 5$.*

Proof. Let $\alpha > 3/2$, and let z be an optimal α-repetitive sequence. By Lemma 1, we may assume that the sequence z is over the alphabet $\{0, 1\}$. Since z is aperiodic, one of the words 00 or 11 occurs in z; without loss of generality, we may suppose that 00 occurs. Let w denote a minimal α-power in z with a prefix 01. Since $\alpha > 3/2$, the word w does not equal 010, and hence w is of the form $w = 01u01v$, where u and v are finite, possibly empty, words. This implies that there exists another minimal α-power in z with a prefix 01. Noticing that 00 is a minimal α-power in z, we have shown that z has at least three minimal α-powers with prefix 0. Also, z must have at least two minimal α-powers with prefix 1. Therefore, z has altogether at least five minimal α-powers, and consequently $M(\alpha) \geq 5$.

Next we will show that, for all α with $3/2 < \alpha < 2$, there exists an aperiodic α-repetitive sequence with exactly five minimal α-powers. To this end, choose an integer $i \geq 1$ such that

$$\alpha \leq \frac{2i+3}{i+2}.$$

Let z be the characteristic word with the directive sequence $(d_n)_{n\geq 1}$, where $d_1 = i$ and $d_n = 1$ for $n \geq 2$. The sequence z being Sturmian, it has six minimal squares. We have

$$s_5 = 0^i 10^{i+1} 10^i 10^{i+1} 10^{i+1} 1,$$

and from s_5 we can find the six minimal squares of z. They are

$$00, \quad 010^i 10^{i-1}, \quad 010^{i+1} 10^i, \quad 10^i 10^i, \quad 10^{i+1} 10^{i+1}, \quad 10^{i+1} 10^i 10^{i+1} 10^i.$$

Now we see that each position of z has an occurrence of one of the words

$$00, \quad 010^i10^{i-1}, \quad 010^{i+1}10^i, \quad 10^i10^i, \quad 10^{i+1}10^i.$$

All but the last one of these words are squares. The last word is a $(2i+3)/(i+2)$-power. Therefore z is an aperiodic α-repetitive sequence with five minimal α-powers. This shows that $M(\alpha) = 5$, and the proof is complete.

Next theorem is only a rephrasing of Theorem 4.

Theorem 12. *We have* $M(2) = 6$.

The details of the proof of our next theorem can be found in the full version of this paper.

Theorem 13. *For* $2 < \alpha \leq 17/8$, *we have* $M(\alpha) = 12$.

Proof (Sketch). If z is an optimal overlapful sequence over the alphabet $\{0, 1\}$ with a factor 00, it can be shown that there exists two integers i and j with $j > i \geq 1$ such that each word 01, 001, 10^i1, and 10^j1 corresponds to at least three distinct minimal overlaps. Therefore the number of minimal overlaps is at least 12. Conversely, the Fibonacci word is everywhere 17/8-repetitive with exactly twelve minimal 17/8-powers.

Acknowledgments

The author is grateful to J. Karhumäki for several discussions and comments. Thanks also to J. Cassaigne for inspiring discussions about the topic of this paper, and to D. Krieger for the discussion about the value $P(\mathbf{t})$. Finally, the author wants to thank the anonymous reviewers for their comments and remarks.

References

1. Allouche, J.-P., Shallit, J.: Automatic sequences. Theory, applications, generalizations. Cambridge University Press, Cambridge (2003)
2. Allouche, J.-P., Shallit, J.: The ubiquitous Prouhet-Thue-Morse sequence. In: Ding, C., Helleseth, T., Niederreiter, H. (eds.) Sequences and their applications, Proceedings of SETA'98, pp. 1–16. Springer Verlag, Heidelberg (1999)
3. Choffrut, C., Karhumäki, J.: Combinatorics of words. Handbook of formal languages 1, 329–438 (1997)
4. Karhumäki, J., Lepistö, A., Plandowski, W.: Locally periodic versus globally periodic infinite words. J. Combin. Theory Ser. A 100(2), 250–264 (2002)
5. Lothaire, M.: Algebraic Combinatorics on Words. Encyclopedia of Mathematics and its Applications, p. 90. Cambridge University Press, Cambridge (2002)
6. Mignosi, F., Restivo, A., Salemi, S.: Periodicity and the golden ratio. Theoret. Comput. Sci. 204(1-2), 153–167 (1998)

Timed Traces and Strand Spaces

Robin Sharp and Michael R. Hansen

Informatics and Math. Modelling, Technical University of Denmark

Abstract. This paper presents an approach to the analysis of real-time properties of security protocols, based on the Strand Space formalism for describing the behaviour of the participants in the protocol. The approach is compared with a trace-based analysis introduced by Pilegaard et al. [14]. Interval Logic with durations is used to express and reason about temporal phenomena. Strand Spaces were chosen as the starting point for our approach, since the causalities between important events in protocols are revealed in an illustrative manner by this formalism. The advantage of the trace-based approach is that it supports inductive reasoning in connection with the analysis of untimed properties. Interval Logic is chosen as the real-time formalism, as timing requirements and timing properties of security protocols are often expressible as interval properties. As an example, the Kerberos authentication protocol, which is based on concepts like timestamps and lifetimes, and which requires freshness of certain messages, is analysed.

Keywords: Security protocols, real-time, interval logic, verification.

1 Introduction

Many security protocols only work correctly to provide the degree of security expected by their users if certain temporal constraints on their operation are fulfilled. A typical requirement is that the exchange of messages involved in the protocol must be completed within a given time. If this does not happen, the protocol exchange is considered invalid, and the secure service which the protocol is intended to implement cannot be made available to the user. To analyse and reason about this type of requirement, we need to use timed models which can describe both trusted, concurrently executing agents attempting to execute the protocol, and untrusted intruders trying actively to disturb the execution of the protocol by deleting, modifying or inserting messages.

Paulson [12] has shown that an approach to the analysis of security protocols based on inductively defined traces is effective when the goals to be shown are not time dependent. In [14], Pilgaard et al. reported an attempt at combining Paulson's approach with Interval Logics in order to be able to analyse temporal properties of security protocols. Interval logics with the concept of durations have been shown (see for example [24,3,21,9]) to be a framework which is well-suited to reasoning about the temporal properties of systems with concurrent activities, shared resources and various scheduling disciplines. In [14], these ideas were extended to deal with concurrently executing agents which construct messages,

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 373–386, 2007.

transmit them via a shared network and check the received messages in accordance with given protocols, in the presence of hostile intruders, thus making it possible to analyse active attacks and availability questions.

In this paper, we consider how this approach can be based on the formalism of Strand Spaces introduced in [19] instead of traces. In relation to traces, Strand Spaces only consider causal sequences of events in the behaviour of the protocol participants, thus reducing the number of possible event sequences and making it possible to introduce an algebra of events which are permitted according to the protocol. Our intention here is to show how this powerful formalism can be combined with temporal reasoning using an interval logic.

2 General Ideas

The fundamental aim of a security protocol is that active *agents* should exchange *messages* via a network in a manner which achieves some security target, such as confidentiality or authentication, even in the presence of hostile intruders. We initially describe protocols by Alice-and-Bob specifications. Each step in such a specification has the form: $A \rightarrow B : \ M$, with the meaning that message M is sent from agent A to agent B via the network. The specification is a *schema* in the sense that M may contain instantiable variables and that A and B can be instantiated to different agents in different executions of the protocol.

As an example, an Alice-and-Bob specification of (a simplified version) of the Kerberos Version 5 authentication protocol [6] is shown in Fig. 1. This protocol

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{K_{sess}, N_A, \mathcal{L}, B\}_{K_{AS}}$
3. $A \rightarrow B : \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{A, \mathcal{T}_A, K_{Asub}\}_{K_{sess}}$
4. $B \rightarrow A : \{\mathcal{T}_A, K_{Bsub}\}_{K_{sess}}$

Fig. 1. The Kerberos v5 authentication protocol

enables two parties, A and B, to convince one another, by exchange of information with a trusted server S, that they know one another's identities and know a shared secret *session key*, K_{sess}, determined by S. It also allows them if required to evaluate a shared integer valued secret key *not* known to S from the pair of subkeys (K_{Asub}, K_{Bsub}). Here and elsewhere we use the notation that $A, B, \ldots$ are agents, N_i is a nonce produced by agent i, $\mathcal{T}_i$ is a timestamp produced by agent i, K_{ij} is a key known only to agents i and j, and $\{M\}_K$ represents a message M encrypted with key K.

In the Kerberos protocol, $\{K_{sess}, A, \mathcal{L}\}_{K_{BS}}$ is known as A's *ticket* for B. This ticket is sent by A to B in order to authenticate A to B. It is characteristic of the protocol that the ticket has a finite period of validity, often known as its *lifetime*, in Fig. 1 designated by $\mathcal{L}$. The lifetime is defined in terms of an end time and optional start time by the server S, and is incorporated in the ticket which is passed to A in step 2 of the protocol. In step 3, A passes the ticket

on to B, together with the *authenticator* $\{A, \mathcal{T}_A, K_{Asub}\}_{K_{sess}}$, which contains a timestamp $\mathcal{T}_A$ inserted by A. By decrypting the ticket, B can obtain the session key K_{sess} and the lifetime $\mathcal{L}$, and can then use the session key to decrypt the authenticator and recover the timestamp. The authentication process terminates correctly if the timestamp lies within the limits specified by $\mathcal{L}$.

3 The Kerberos Protocol in a Strand Space Formalism

Strand Spaces, introduced in [19], offer a framework for describing protocol behaviour and proving the correctness of protcols with respect to security specifications. A *strand* is a sequence of events, which can be performed by a legitimate participant in a protocol or by an intruder (in [19] a *penetrator*). A *strand space* is a set of strands together with a graph structure describing causal interactions. A portion of a strand space which describes a particular (desired or undesired) execution of a protocol is known as a *bundle*. An example of a bundle showing the desired behaviour of the Kerberos protocol is shown in Fig. 2(a). The nodes of the graphs denote events, and the arrows denote causal relationships, with single arrows indicating exchange of messages, and double arrows causal relationships within the strands for the individual participants A, B and S. In the

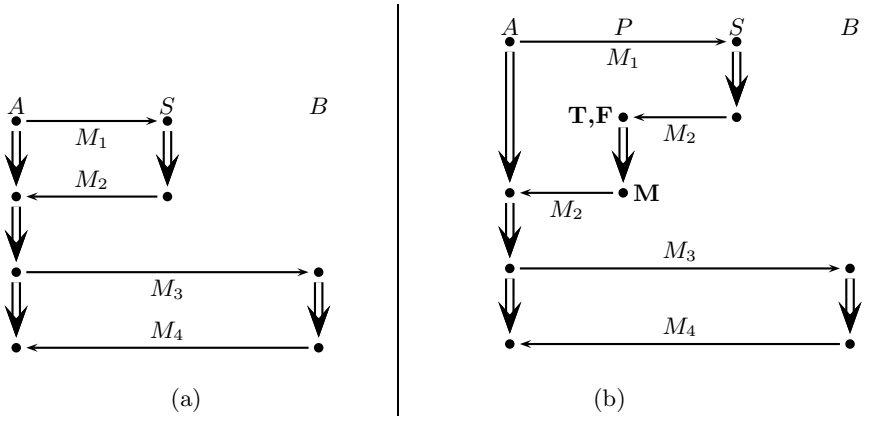


Fig. 2. (a) A bundle for the desired behaviour of the Kerberos authentication protocol (b) A bundle for an undesired behaviour of the Kerberos authentication protocol

strand space approach, it is assumed that an intruder possess a certain set of encryption keys, K_P, and can perform the atomic actions shown in Fig. 3 An intruder strand consists of a sequence of such actions. Fig. 2(b) shows a bundle with intruder strands, which illustrates an *undesired* behaviour of the Kerberos protocol. The intruder P receives message M_2, which is intended for A, and keeps it for a certain period of time before passing it on to A. In the untimed model of Strand Spaces, this behaviour looks innocent. However, it may result in the ticket enclosed in message M_2 arriving at A so late that it is no longer

M: Introduce a text *Message*, $t \in \mathcal{T}$, into the network.
F: *Flush* (remove) a message from the network.
T: Receive a message and *Duplicate* it.
C: *Concatenate* two received messages into one and send it.
S: *Separate* a composed message into its components and send them.
K: Send a *Key*, $k \in K_P$.
E: Receive a key and a message, *Encrypt* the message with the key and send it.
D: Receive a key and a message, *Decrypt* the message with the key and send it.

Fig. 3. Atomic actions which can be performed by intruders

valid, in which case P has successfully carried out a denial-of-service attack. This undesired scenario can be detected in a timed framework like that described in [14]. In the following sections we shall establish a connection between that timed framework and strand spaces.

4 Untimed Model of Network Behaviour

We start by presenting an untimed model for network behaviour, first presented in [14]. Since the aim of the work described there was to achieve a model of security protocols which was suited for verification using theorem provers, the untimed model is based on Paulson's inductive approach to verification [12].

4.1 Messages

We base the formalization of the notions *agents*, *keys* and *messages* on [12]. A message is either an agent, a natural number, a nonce, a key, a message hash value, a message pair or an encrypted message, modelled as follows:

```
datatype msg = Agent agent | Number nat | Nonce nat | Key key
             | Hash msg | MPair msg msg | Crypt key msg
```

where an `agent` is either an intruder (in Paulson's notation "a spy"), a friend or a server. Paulson assumes an infinite set of cryptographic keys, where each key has an inverse that reverses its cryptographic effect. Each part of the declaration can be considered a rule for generating messages. For example, if K is a key and M a message, then (`Crypt` K M) is a new message obtained by encryption of M using K. The datatype `msg` corresponds to the free algebra of *terms* in strand spaces. Note, however, that algebraic properties of terms may be introduced in strand spaces, but this topic will not be pursued any further in this paper.

4.2 Network Packets

Alice-and-Bob specifications are schemata in several senses. They specify a collection of protocol instances, where for example A, B and S can be instantiated differently in the various runs of the protocol. But they also specify a desired run (with no attack) of the protocol, while allowing undesired runs during which

attacks occur [8]. In order to distinguish desired from undesired protocol runs, the notion of a *packet* was introduced in [14]. A packet models what is in fact sent between two agents: the *sender* and *receiver* of the packet. A packet is a message with information about the *names* of the sender and receiver. Malicious agents may lie about their names. In order to take this into account, a name is modelled by a pair (p, i), where p is denoted the *persona* and i the (true) *identity* of the agent. If $p \neq i$, then the persona is a *fake persona*, and the name is a *fake name*. A network *packet* has the form: $(\!|s \rightarrow r : M|\!)$, where M is a message, s is the name of the sender and r is the name of the receiver. If either s or r are fake names, then the packet is a *fake packet*.

4.3 Events

A protocol step $A \rightarrow B : M$ typically involves multiple activities: Firstly, the *events* that agent A *sends* a packet to agent B containing M and that B *receives* a packet from A containing M. And secondly, the event – known as a *block* event – that B, after receiving the packet, removes it from the network. Thus, for a packet $q = (\!|(p_A, i_A) \rightarrow (p_B, i_B) : M|\!)$, we have the following three kinds of event:

$$\begin{aligned} &\textsc{Snd}_A(q) : A \text{ sends packet } q \\ &\textsc{Rcv}_B(q) : B \text{ receives packet } q \\ &\textsc{Blk}_B(q) : B \text{ blocks packet } q \end{aligned}$$

In a send event, it is a requirement that the identity of the message's sender must be that of the agent. Apart from this, neither the identities nor the personas appearing in the packet need to correspond to the agent, since events may involve fake packets or be received or blocked by agents for which they were not intended. All such events will be denoted *fake events*.

A desired run of $A \rightarrow B : M$ is described by the following event sequence:

$$\textsc{Snd}_A(\!|(A, A) \rightarrow (B, B) : M|\!)\ \textsc{Rcv}_B(\!|(A, A) \rightarrow (B, B) : M|\!)\ \textsc{Blk}_B(\!|(A, A) \rightarrow (B, B) : M|\!)$$

In [14] it was shown that with these types of events one can also model the standard types of active and passive attacks, such as message interception, interruption, theft, replaying, modification and spoofing. It was also made clear that they are adequate in the sense that all the atomic actions of the strand space formalism can be expressed.

4.4 Traces

Initially, let us consider a trace based model of a network's behaviour. The traces are inductively defined as the least inductive closure of a set of rules, using the approach of Roscoe, Lowe and Paulson [18,7,12].

Let $[e_1, \ldots, e_n]$ denote a list with $n > 0$ elements, $[\,]$ the empty list, and $e\#t$ the list formed by adding element e to the list t. In the list $[e_1, \ldots, e_n]$, e_1 is the most recent event which has happened and e_n is the oldest event.

A trace should capture the idea that a packet cannot be received before it is sent or after it is blocked. Therefore, we introduce the notion of a visible packet.

A packet q is *visible* in an event list h, written q visible h, if q occurs in the list and q is not blocked by a more recent element in h.

The set of *traces*, $trace \subseteq event$ list, is then the least set generated by the following rules:

$$\frac{}{[\,] \in trace} \quad \frac{h \in trace}{\textsc{Snd}_n(q)\#h \in trace} \quad \frac{h \in trace \quad q \text{ visible } h}{\textsc{Rcv}_n(q)\#h \in trace} \quad \frac{h \in trace \quad q \text{ visible } h}{\textsc{Blk}_n(q)\#h \in trace}$$

A trace corresponds to a bundle in strand spaces, as in a trace one can see the interaction of the various agents.

4.5 Ideal Traces

The set *trace* includes all possible traces, both desired and undesired ones. In order to discuss possible attacks, we introduce the notion of *ideal traces*, where fake events do not occur, and we require that an agent who receives a packet will block it immediately afterwards.

The ideal traces are generated inductively in a manner similar to the definition of *trace* above. However, with ideal traces we also need to have a concrete protocol in mind in order to be able to give meaning to the concepts of "desired" and "undesired" traces. In what follows, we shall refer to the Kerberos protocol whose specification was given in Fig. 1.

It is convenient to introduce some auxiliary functions:

$$\begin{aligned} \mathsf{hd} &: \quad event \text{ list} \to event \\ \mathsf{used} &: event \text{ list} \to \mathcal{P}(msg) \\ \downarrow &: \quad event \text{ list} \to agent \to event \text{ list} \end{aligned}$$

which respectively give the first element in a non-empty list of events, the set of messages occurring in an event list, and the restriction of an event list to a given agent. $\downarrow$ is an infix function. Thus $(evs \downarrow A)$ gives the *projection* of evs onto A, i.e. the event list obtained from evs by deleting events in which A does not participate, in the sense that A does not send, receive or block the packet.

The set KER of ideal traces for the Kerberos protocol is a set of event lists:

$$KER : \mathcal{P}(event \text{ list})$$

which is generated by the rules below. The first two rules express the fact that generation starts from the empty event list and that a visible packet is received and blocked in "one step":

$$\frac{}{[\,] \in KER}$$

$$\frac{evs \in KER \quad (\!|(A,A) \to (B,B) : M|\!) \text{ visible } evs}{\textsc{Blk}_B\,(\!|(A,A) \to (B,B) : M|\!)\#\textsc{Rcv}_B\,(\!|(A,A) \to (B,B) : M|\!)\#evs \in KER}$$

Each of the next four rules is a direct encoding of a step of the Alice-and-Bob specification. The predicate $N_A \notin \mathsf{used}\ evs$ assures the freshness of the nonce N_A, and by inspection of the last step of A using $\mathsf{hd}(evs \downarrow A) = \cdots$ the next protocol step of A is known. Similarly for B and S.

$$\frac{evs \in KER \quad N_A \notin \mathsf{used}\ evs}{\textsc{Snd}_A\ (\!|(A,A) \rightarrow (S,S) :\ A, B, N_A|\!) \# evs \in KER}$$

$$\frac{evs \in KER \quad \mathsf{hd}(evs \downarrow S) = \textsc{Blk}_S\ (\!|(A,A) \rightarrow (S,S) :\ A, B, N_A|\!)}{\textsc{Snd}_S\ (\!|(S,S) \rightarrow (A,A) :\ \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{K_{sess}, N_A, \mathcal{L}, B\}_{K_{AS}}|\!) \# evs \in KER}$$

$$\frac{evs \in KER \quad \mathsf{hd}(evs \downarrow A) = \textsc{Blk}_A\ (\!|(S,S) \rightarrow (A,A) :\ \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{K_{sess}, N_A, \mathcal{L}, B\}_{K_{AS}}|\!)}{\textsc{Snd}_A\ (\!|(A,A) \rightarrow (B,B) :\ \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{A, \mathcal{T}_A, K_{Asub}\}_{K_{sess}}|\!) \# evs \in KER}$$

$$\frac{evs \in KER \quad \mathsf{hd}(evs \downarrow B) = \textsc{Blk}_B\ (\!|(A,A) \rightarrow (B,B) :\ \{K_{sess}, A, \mathcal{L}\}_{K_{BS}}, \{A, \mathcal{T}_A, K_{Asub}\}_{K_{sess}}|\!)}{\textsc{Snd}_B\ (\!|(B,B) \rightarrow (A,A) :\ \{\mathcal{T}_A, K_{Bsub}\}_{K_{sess}}|\!) \# evs \in KER}$$

Repeated runs of the protocol, with different instantiations of A, B and S, are modelled by ideal traces generated by the above rules. Fig. 4(a) shows an ideal trace corresponding to a single run of the entire Kerberos protocol.

Fig. 4(b) shows a trace in the presence of the attack illustrated in Fig. 2(b).

(a)	(b)
$t_1 = [\ \textsc{Snd}_A\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$	$t_2 = [\ \textsc{Snd}_A\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$
$\textsc{Rcv}_S\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$	$\textsc{Rcv}_S\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$
$\textsc{Blk}_S\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$	$\textsc{Blk}_S\ (\!\|(A,A) \rightarrow (S,S) :\ M_1\|\!),$
$\textsc{Snd}_S\ (\!\|(S,S) \rightarrow (A,A) :\ M_2\|\!),$	$\textsc{Snd}_S\ (\!\|(S,S) \rightarrow (A,A) :\ M_2\|\!),$
	$\textsc{Rcv}_P\ (\!\|(S,S) \rightarrow (A,P) :\ M_2\|\!),$
	$\textsc{Blk}_P\ (\!\|(S,S) \rightarrow (A,P) :\ M_2\|\!),$
	$\textsc{Snd}_P\ (\!\|(S,P) \rightarrow (A,A) :\ M_2\|\!),$
$\textsc{Rcv}_A\ (\!\|(S,S) \rightarrow (A,A) :\ M_2\|\!),$	$\textsc{Rcv}_A\ (\!\|(S,P) \rightarrow (A,A) :\ M_2\|\!),$
$\textsc{Blk}_A\ (\!\|(S,S) \rightarrow (A,A) :\ M_2\|\!),$	$\textsc{Blk}_A\ (\!\|(S,P) \rightarrow (A,A) :\ M_2\|\!),$
$\textsc{Snd}_A\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$	$\textsc{Snd}_A\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$
$\textsc{Rcv}_B\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$	$\textsc{Rcv}_B\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$
$\textsc{Blk}_B\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$	$\textsc{Blk}_B\ (\!\|(A,A) \rightarrow (B,B) :\ M_3\|\!),$
$\textsc{Snd}_B\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!),$	$\textsc{Snd}_B\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!),$
$\textsc{Rcv}_A\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!),$	$\textsc{Rcv}_A\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!),$
$\textsc{Blk}_A\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!)\ ]$	$\textsc{Blk}_A\ (\!\|(B,B) \rightarrow (A,A) :\ M_4\|\!)\ ]$

Fig. 4. (a) An ideal trace of the Kerberos protocol and (b) A trace in the presence of the attack by intruder P illustrated in Figure 2(b)

4.6 Attacks

Given the set of all possible traces and the set of desired traces of a given protocol, we can formalize what constitutes an active attack on a given agent in the trace-based formalism. The main ideas are as follows; a more detailed presentation can be found in [13,14]:

Schematic Projections: For a given event list *evs* and agent A, we define the list of protocol steps of *evs* as they are "seen by" A, in the sense that A just sees the personas occurring in names (which may be forged), but not the identities. This list of protocol steps is denoted by: $evs \uparrow A$.

Consistent traces: Event list *evs* is called consistent with respect to an ideal set of traces *ideal* for given agent A, if there exists some trace $evs_i \in ideal$ which is identical to *evs* as seen by A, i.e. $evs \uparrow A = evs_i \uparrow A$.

Active attacks: Event list *evs* contains an active attack on a "good agent" A, given an ideal set of traces *ideal*, if *evs* is consistent with *ideal* for A, and A takes part in a fake event in *evs*.

More formally, a schematic projection is a function which for a given event list, *evs*, and a given agent, a, can extract a list of protocol steps which that agent takes part in. The list of steps is denoted a *schematic protocol trace*, and has the form:

$$[\,(\!|a_1 \rightarrow b_1 : M_1|\!), (\!|a_2 \rightarrow b_2 : M_2|\!), \ldots, (\!|a_q \rightarrow b_q : M_q|\!)\,]$$

where a_i, b_i are agent personas and M_i is a message, for $1 \leq i \leq q$. The schematic protocol trace for A corresponds to A's strand.

For example, the schematic projection onto A of the ideal trace t_1 shown in Fig. 4(a), written $t_1 \uparrow A$, is:

$$[\,(\!|A \rightarrow S : M_1|\!), (\!|S \rightarrow A : M_2|\!), (\!|A \rightarrow B : M_3|\!), (\!|B \rightarrow A : M_4|\!)\,]$$

Thus the result for a given agent is exactly the corresponding schematic Alice-and-Bob specification for that agent. When applied to multiple protocol executions, with or without interleaving, the resulting schematic trace is always *consistent* with the Alice-and-Bob specification.

A trace is *consistent* with respect to a protocol if, for all good agents participating in the trace, the corresponding schematic projections are equal to that of some ideal execution. If $(GAgent\ evs)$ identifies the good participants (i.e. those agents which are not intruders) in a list of events *evs*, then the notion of *consistency* of an event list *evs* with respect to a set of ideal traces *ideal* is formalized by the infix function Ξ, defined by:

$$evs\ \Xi\ ideal \mathrel{\hat{=}} evs \in trace \wedge \forall A \in (GAgent\ evs) \cdot (\exists evs_i \in ideal \cdot (evs_i \uparrow A = evs \uparrow A))$$

We note for example that the trace t_2 shown in Fig. 4(b) is consistent with the Kerberos protocol, since $(t_2 \uparrow a_i = t_1 \uparrow a_i)$ for all $a_i \in \{A, B, S\}$. However, since A takes part in the fake event

$$\textsc{Rcv}_A\ (\!|(S, P) \rightarrow (A, A) : M_2|\!)$$

in t_2, then t_2 contains an active attack.

5 Interval Logics

To reason about temporal phenomena, we use the Duration Calculus (DC), which is a real-time logic introduced in [25] as an extension to Interval Temporal Logic [4]. In DC [22], time is modelled as real numbers $\mathbb{T}ime \mathrel{\hat{=}} \mathbb{R}$ and a real-time system is modelled by Boolean-valued functions of time

$$P : \mathbb{T}ime \to \{0, 1\}\,.$$

Such functions are also called *state variables*. The intuition is that the system is in state P at time t iff $P(t) = 1$. State variables can be combined using the Boolean connectives to form *state expressions*. For example, $P \wedge \neg Q$ is a state expression and the system is in this (combined) state at time t iff it is in state P and not in state Q at time t.

Timing properties are expressed in a modal logic, where the possible worlds are *bounded* and *closed time intervals*:

$$\mathbb{I}ntv \;=\; \{\,[b, e] \mid b, e \in \mathbb{T}ime \wedge b \leq e\,\}$$

From a state expression S a *term* called the *duration* of S, denoted $\int S$, can be formed. For a given interval $[b, e]$ this denotes the accumulated time where the system is in state S in the interval:

$$\int_b^e S(t)dt$$

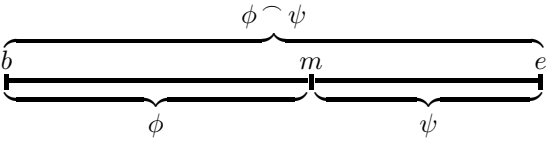
Furthermore, we introduce the terms ℓ and TIME, where for a given interval $[b, e]$, ℓ denotes the *length* $e - b$ of the interval, and TIME denotes the time e at the right-hand end point.

Terms are formed from durations, ℓ, TIME, constants and variables using functions of real arithmetic. From terms, *atomic formulas* are formed using relations of real arithmetic, and *formulas* are constructed from the atomic formulas using connectives and quantifiers of predicate logic, together with *modalities* which will be introduced below. An example of a formula is

$$\textstyle\int S = \ell \;\wedge\; \ell > 0$$

This formula, commonly abbreviated to $\lceil S \rceil$, expresses the fact that "state S is present (almost) everywhere on a non-point interval". Terms and formulas in DC are all interpreted with respect to intervals, i.e. for a given interval $[b, e]$ a term θ denotes a real number and a formula ϕ is either true or false.

In Interval Temporal Logic [4,10,25,5], which was the original basis for DC, there is only one basic modality, which is known as *chop*. A formula $\phi \frown \psi$ (reads: "ϕ chop ψ") holds on $[b, e]$ iff there exists $m \in [b, e]$ such that ϕ holds on $[b, m]$ and ψ holds on $[m, e]$:

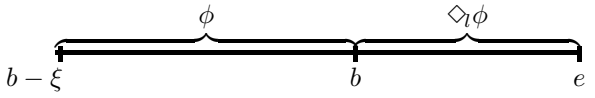


This modality is said to be a *contracting* modality, since by using chop only subintervals of the original interval can be reached. With contracting modalities only safety properties can be expressed. In order to be able to express liveness and fairness properties DC is now based on Neighbourhood Logic (NL) [23,1], which

has two basic modalities, designated $\Diamond_l$ (reads: "for some left neighbourhood") and $\Diamond_r$ (reads: "for some right neighbourhood"), defined by:

$$\Diamond_l \phi \text{ holds on } [b,e] \text{ iff there exists } \xi \geq 0 \text{ such that } \phi \text{ holds on } [b-\xi, b]$$
$$\text{and } \Diamond_r \phi \text{ holds on } [b,e] \text{ iff there exists } \xi \geq 0 \text{ such that } \phi \text{ holds on } [e, e+\xi]$$

With $\Diamond_l$ one can reach *left* neighbourhoods of the beginning point of an interval:



and with $\Diamond_r$ one can correspondingly reach *right* neighbourhood intervals. These modalities are called *expanding* modalities, as one can reach intervals outside a given interval with them. In a first order interval logic with these two modalities, one has an adequate interval logic in the sense that any other interval modality (including chop) is expressible [23]. In this paper we will also use modalities to reach sub-intervals and proper sub-intervals:

$$\Diamond \phi \;\hat{=}\; \mathsf{true} \frown \phi \frown \mathsf{true} \qquad \text{reads: "for some sub-interval: } \phi\text{"}$$
$$\diamondsuit \phi \;\hat{=}\; (\ell > 0) \frown \phi \frown (\ell > 0) \qquad \text{reads: "for some proper sub-interval: } \phi\text{"}$$

Furthermore, for each unary modality there is a dual defined in the standard way. For example $\Box_l \phi$, defined by $\neg \Diamond_l \neg \phi$, with the meaning "for all left neighbourhoods: ϕ".

6 Timed Traces and Strand Spaces

We shall connect untimed traces to DC using a technique introduced in [17], where a special trace variable Tr, called a *timed trace*, which gives the trace as a function of time, is introduced. This function is raised to a function on intervals: $\mathsf{Tr} : \mathbb{I}ntv \to trace$, such that the value of Tr on $[a,b]$ is the trace of events which is observed at the right end point b. The timed trace must satisfy a collection of properties to faithfully describe the timed behaviour of the network, e.g. the trace can only grow as time progresses and only a finite number of new events can occur in an interval. In order to express these properties and timing properties of protocols, we introduce some abbreviations.

Suppose $h \in trace$, then $\mathsf{Throughout}\, h$ is a formula which holds for a non-point interval $[a,b]$, if the trace is equal to h *inside* $[a,b]$:

$$\mathsf{Throughout}\, h \;\hat{=}\; \ell > 0 \wedge \boxdot\, \mathsf{Tr} = h\,.$$

The formula Stable expresses that no event has occurred inside an interval:

$$\mathsf{Stable} \;\hat{=}\; \exists h \in trace.\mathsf{Throughout}\, h\,.$$

We shall assume that only a finite number of events can occur in an interval, i.e. the variable Tr is *finitely variable*. Hence the following formulas are axioms:

$$\Diamond_l \mathsf{Stable} \qquad \text{and} \qquad \Diamond_r \mathsf{Stable}\,.$$

If $h_1, h_2 \in \mathit{trace}$, then h_1 is called a *prefix* of h_2 (written $h_1 \preceq h_2$), if there exist events $e_1, \ldots, e_n$ such that $e_1 \# \cdots \# e_n \# h_1 = h_2$. The trace grows monotonically with time:

$$\mathsf{Tr} = h \Rightarrow \Box_r(h \preceq \mathsf{Tr})\,.$$

Even though the trace is finitely variable, several events can occur at a given time point. To capture the occurrence of an event, we let $h_2 - h_1 = \{e_1, \ldots, e_n\}$ when $h_1 \preceq h_2$. Event e occurs "now" if e extends every earlier trace, where earlier trace are reached by the use of modalities:

$$\mathsf{Occurs}(e) \;\hat{=}\; \ell = 0 \wedge \exists h.(\mathsf{Tr} = h \wedge \Box_l\,(\ell > 0 \Rightarrow \boxdot\,(e \in h - \mathsf{Tr})))\,.$$

Note that the two occurrences of Tr refer to different intervals. The above "well-formedness" properties are general in the sense that they must hold for all possible systems. Such well-formedness properties are further discussed in [14].

On the other hand, certain properties are system specific. For example, in some systems processes may communicate in a synchronous manner, while in others asynchronous communication is the choice. Furthermore, it is system dependent whether one would have a model allowing many events to occur at a given point in time. We show that within this simple setting, we can model a variety of system properties. As an example we show how synchronous and asynchronous network communication may be described. In the synchronous case it means that packets are sent and received at the same time

$$\mathsf{Occurs}(\textsc{Snd}_A(q)) \Longleftrightarrow \mathsf{Occurs}(\textsc{Rcv}_B(q))$$

if q is a packet which is sent from A to B. Notice that we here exploit the fact that events from different agents can occur at the same time.

A minimal network delay δ can be modelled using *block* events as follows:

$$(\mathsf{Occurs}(\textsc{Rcv}_B(q)) \frown \mathsf{true} \frown \mathsf{Occurs}(\textsc{Blk}_B(q))) \;\Rightarrow\; \ell \geq \delta\,,$$

and asynchronous communication with a minimal delay may be modelled by:

$$\mathsf{Occurs}(\textsc{Rcv}_B(q)) \;\Rightarrow\; \Diamond_l(\mathsf{Occurs}(\textsc{Snd}_A(q)) \;\wedge\; \ell \geq \delta)$$

7 Timing Requirement in the Kerberos Protocol

The timing requirement of the ideal run of the protocol, described by the bundle in Fig. 2, can be expressed as follows:

$$\mathsf{Occurs}(\textsc{Snd}_B(B \rightarrow A : M_4)) \implies \begin{pmatrix} \Diamond_l(\mathsf{Occurs}(\textsc{Snd}_A(A \rightarrow B : M_2)) \frown \mathsf{true}) \\ \wedge\; \mathcal{T}_A \in [t_1, t_2] \end{pmatrix}$$

where $\mathcal{T}_A$ is the timestamp set by the clock of A, which occurs in M_3 and M_4, and the lifetime (t_1, t_2) comes from the first message M_2 of S, i.e. $\mathcal{L} = [t_1, t_2]$. The condition $\mathcal{T}_A \in [t_1, t_2]$ expresses that A must initiate the communication with B during the lifetime of the ticket, and this is all that is required in the

specification of the protocol [6]. It would, however, be natural to strengthen this formula to require that B also finishes its task within the lifetime of the ticket

$$\mathcal{T}_A \in [t_1, t_2] \wedge \mathsf{TIME} \in [t_1, t_2] \tag{1}$$

to make B less vulnerable to denial of service attacks. Notice that B can actually check this condition as B "owns" the relevant keys.

Let us suppose for a moment that the protocol instance is aborted as soon as a violation of the timing condition (1) is experienced by B.

7.1 Sensitivity with Respect to Delays

The Strand Space formalism has the advantage that bundles give an overview of the desired as well as undesired runs. Consider, for example, the bundle in Fig. 2(b). Informally, by considering the timing information, A may abort the protocol instance in the case that A's local time is later than the endpoint t_2 of $\mathcal{L}$. If this is not the case, the worst case scenario is that B aborts the protocol instance, as all good participants have wasted their time in this case.

With a model of network delays and execution times in the participants, we can extract a symbolic expression relating delays, execution times and the lifetime of tickets, which describes the cases with successful protocol termination.

The delay δ_P caused by P in the bundle in Fig. 2(b), may be described by:

$$(\mathsf{Occurs}(\textsc{Snd}_S(S \to A : M_2)) \frown \mathsf{true} \frown \mathsf{Occurs}(\textsc{Snd}_P((S, P) \to A : M_2))) \implies \ell = \delta_P$$

Other delays are expressed similarly.

Suppose that δ_S and δ_B are the internal delays in S and B caused by the preparation of M_2 and M_4, respectively, and that other delays are negligible. We would like to extract a condition for the delay caused by P in order to prevent the worst-case scenario, for example, by using a timeout mechanism in A.

Exploiting the inference system in Sect. 4, one can, in the timeless setting, derive that certain events have occurred prior to the event $e_b = \textsc{Snd}_B(B \to A : M_4)$. In this worst-case scenario, we consider the events $e_a = \textsc{Snd}_A(A \to B : M_3), e_p = \textsc{Snd}_P((S, P) \to A : M_2)$, and $e_s = \textsc{Snd}_S(S \to A : M_2))$.

In the timed setting, it is not difficult to derive formulas for delays between the events, e.g.

$$\mathsf{Occurs}(e_b) \implies \Diamond_l(\mathsf{Occurs}(e_a) \frown \ell = \delta_B)$$

One can prove that the delay allowed by P depends on the upper bound t_2 of the lifetime as follows $t_2 \geq \delta_P + \delta_B$, and A should wait no longer than $\delta_S + t_2 - \delta_B$ for the message M_2.

This simple example illustrates the way untimed reasoning is combined with reasoning about timing, and how an implementation of the protocol could take system parameters into account.

7.2 Drift of Clocks

The protocol is also vulnerable due to drift of clocks, even without the appearance of an attacker. Suppose, for example, that the local clocks of A and B are

faster than the clock of S. If this drift is not taken into account, then eventually there will occur a protocol instance, where the lifetime set up by S would end before the timestamp set by A, and from there on the protocol is useless. This problem is in practice solved by synchronizing the clocks.

To model clock synchonization, we introduce a new event Clk which can occur in traces, and to model the drift of clocks we introduce a function $d_A : \mathbb{R} \to \mathbb{R}$ for each participant A. The idea is that if x is the distance to the most recent occurrence of Clk at time t, then the local time for A, designated by $\mathsf{localTime}_A(t)$, is $t + d_A(x)$. The concept is formalized as follows:

$$\ell = 0 \wedge \mathsf{TIME} = t \implies \left(\begin{array}{l} \mathsf{localTime}_A = t + d_A(x) \\ \iff \Diamond_l((\mathsf{Occurs}(\mathsf{Clk}) \frown \ell = x) \wedge \Box\, \neg \mathsf{Occurs}(\mathsf{Clk})) \end{array} \right)$$

Hence, $\mathsf{localTime}_A$ is a temporal variable, which for point intervals denotes the local time of A.

When a participant, say A, sends a message with timing information, the local time is used, and we have a framework in which we can analyse the protocol in order to estimate the frequency of clock synchronizations, based on the protocol parameters, such as lifetime and delays, and assumptions about the clock drifts.

8 Discussion

The formalism presented in this paper appears to provide a useful framework for the analysis of temporal properties of security protocols, and supplements the Strand Spaces approach. A similar analysis is useful for other protocols whose correctness assumes freshness of nonces, such as the well-known Needham-Schroeder secret key authentication protocols originally presented in [11] but subsequently shown incorrect due to a missing freshness property.

For practical analysis, Neighbourhood Logic and Duration Calculus have been encoded in signed interval logic within the Isabelle/HOL proof assistant. The encoding, developed by Rasmussen and Pilegaard [15,16,13], is based on a labelled natural deduction system. In addition to interval logic, the tool is based on the use of inductive proof methods over traces of events of the system, inspired by the work of Paulson [12]. For further details of this work, see [14].

References

1. Barua, R., Roy, S., Chaochen, Z.: Completeness of neighbourhood logic. Journal of Logic and Computation 10(2), 271–295 (2000)
2. Bellovin, S.M., Merritt, M.: Limitations of the Kerberos authentication system. In: Proc. of the USENIX Conference, pp. 253–267. USENIX Assoc. (1991)
3. Chan, P., Hung, D.V.: Duration calculus specification of scheduling for tasks with shared resources. In: Kanchanasut, K., Levy, J.-J. (eds.) Algorithms, Concurrency and Knowledge. LNCS, vol. 1023, pp. 365–380. Springer, Heidelberg (1995)
4. Halpern, J., Moszkowski, B., Manna, Z.: A hardware semantics based on temporal intervals. In: Díaz, J. (ed.) Automata, Languages and Programming. LNCS, vol. 154, pp. 278–291. Springer, Heidelberg (1983)

5. Hansen, M.R., Chaochen, Z.: Duration Calculus: Logical foundations. Formal Aspects of Computing 9(3), 283–330 (1997)
6. Kohl, J., Neuman, C.: RFC1510: The Kerberos network authentication service (V5). IETF (September 1993)
7. Lowe, G.: Some new attacks upon security protocols. In: Proc. of the 9th IEEE Security Foundations Workshop, pp. 162–169. IEEE Computer Society Press, Los Alamitos (1996)
8. Meadows, C.: A cost-based framework for analysis of denial-of-service in networks. Journal of Computer Security 9(1/2), 143–164 (2001)
9. Mørk, S., Godskesen, J.C., Hansen, M.R., Sharp, R.: A timed semantics for SDL. In: Formal Description Techniques IX: Theory, application and tools, pp. 295–309. Chapman & Hall, Sydney, Australia (1996)
10. Moszkowski, B.: A temporal logic for multilevel reasoning about hardware. IEEE Computer 18(2), 10–19 (1985)
11. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Communications of the ACM 21(12), 993–999 (1978)
12. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128 (1998)
13. Pilegaard, H.: Modelling properties of security protocols. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark (October 2002)
14. Pilegaard, H., Hansen, M.R., Sharp, R.: An approach to analyzing availability properties of security protocols. Nordic Journal of Computing 10(4), 337–373 (2003)
15. Rasmussen, T.M.: Automated proof support for interval logics. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 317–326. Springer, Heidelberg (2001)
16. Rasmussen, T.M.: Interval Logic – Proof Theory and Theorem Proving. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark (2002)
17. Ravn, A.P., Rischel, H., Hansen, K.M.: Specifying and verifying requirements of real-time systems. IEEE Trans. on Software Engineering 19(1), 41–55 (1993)
18. Roscoe, A.W.: Modelling and verifying key-exchange protocols using CSP and FDR. In: Proc. of the 8th IEEE Security Foundations Workshop, IEEE Comp. Soc. Press, Los Alamitos (1995)
19. Thayer Fábrega, F.J., Herzog, J.C., Guttman, J.D.: Strand Spaces: Proving security protocols correct. Journal of Computer Security 7, 191–230 (1999)
20. Yu, C.-F., Gligor, V.D.: A formal specification and verification method for the prevention of denial of service. In: 1988 IEEE Symposium on Security and Privacy, pp. 187–202. IEEE Comp. Soc. Press, Los Alamitos (1988)
21. Yuhua, Z., Chaochen, Z.: A formal proof of the deadline driven scheduler. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, vol. 863, pp. 756–775. Springer, Heidelberg (1994)
22. Chaochen, Z., Hansen, M.R.: Duration Calculus: A Formal Approach to Real-Time Systems. Monographs in Theoretical Computer Science. Springer, Heidelberg (2004)
23. Chaochen, Z., Hansen, M.R.: An adequate first order interval logic. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 581–608. Springer, Heidelberg (1998)
24. Chaochen, Z., Hansen, M.R., Ravn, A.P., Rischel, H.: Duration specifications for shared processors. In: Vytopil, J. (ed.) Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, vol. 571, pp. 21–32. Springer, Heidelberg (1991)
25. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Information Processing Letters 40(5), 269–276 (1991)

On Empirical Meaning of Randomness with Respect to a Real Parameter

Vladimir V'yugin

Institute for Information Transmission Problems, Russian Academy of Sciences,
Bol'shoi Karetnyi per. 19, Moscow GSP-4, 127994, Russia

Abstract. We study the empirical meaning of randomness with respect to a family of probability distributions P_θ, where θ is a real parameter, using algorithmic randomness theory. In the case when for a computable probability distribution P_θ an effectively strongly consistent estimate exists, we show that the Levin's a priory semicomputable semimeasure of the set of all P_θ-random sequences is positive if and only if the parameter θ is a computable real number. The different methods for generating "meaningful" P_θ-random sequences with noncomputable θ are discussed.

1 Introduction

We use algorithmic randomness theory to analyze the empirical meaning of random data generated by a parametric family of probability distributions when the parameter value is noncomputable. More correctly, let a parametric family of probability distributions P_θ (θ is a real number) be given such that an effectively strongly consistent estimate exists for this family. The Bernoulli family with a real parameter θ is an example of such family. We show that in this case the Levin's a priory semicomputable semimeasure of the set of all P_θ-random sequences is positive if and only if the parameter value θ is a computable real number.

We say that a property of infinite sequences have an empirical meaning if the Levin's a priory semimeasure of the set of sequences possessing this property is positive. In this respect, the model of the biased coin with "a prespecified" probability θ of the head is invalid if θ is a noncomputable real number; noncomputable parameters θ can have empirical meaning only in their totality, i.e., as elements of some uncountable sets. For example, P_θ-random sequences with noncomputable θ can be generated by a Bayesian mixture of these P_θ using a computable prior. In this case, evidently, the semicomputable semimeasure of the set of all sequences random with respect to this mixture is positive.

We also show that the Bayesian statistical approach is insufficient to cover all possible "meaningful" cases: a probabilistic machine can be constructed, which with probability close to one outputs a random θ-Bernoulli sequence, where the parameter θ is not random with respect to each computable probability distribution.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 387–396, 2007.

2 Preliminaries

Let Ξ be the set of all finite binary sequences, Λ be the empty sequence, and Ω be the set of all infinite binary sequences. We write $x \subseteq y$ if a sequence y is an extension of a sequence x, $l(x)$ is the length of x. A real-valued function $P(x)$, where $x \in \Xi$, is called semimeasure if

$$\begin{aligned} P(\Lambda) &\leq 1, \\ P(x0) + P(x1) &\leq P(x) \end{aligned} \tag{1}$$

for all x, and the function P is semicomputable from below; this means that the set $\{(r, x) : r < P(x)\}$, where r is a rational number, is recursively enumerable. A definition of upper semicomputability is analogous.

Solomonoff proposed ideas for defining the a priori probability distribution on the basis of the general theory of algorithms. Levin [13], [3] gave a precise form of Solomonoff's ideas in a concept of a maximal semimeasure semicomputable from below (see also Li and Vitányi [7], Section 4.5, Shen et al. [9]). Levin proved that there exists a maximal to within a multiplicative positive constant factor semimeasure M semicomputable from below, i.e. for every semimeasure P semicomputable from below a positive constant c exists such that the inequality

$$cM(x) \geq P(x) \tag{2}$$

holds for all x. The semimeasure M is called *the a priory* or universal semimeasure.

A function P is a measure if (1) holds, where both inequality signs $\leq$ are replaced on $=$. Any function P satisfying (1) (with equalities) can be extended on all Borel subsets of Ω if we define $P(\Gamma_x) = P(x)$ in Ω, where $x \in \Xi$ and $\Gamma_x = \{\omega \in \Omega : x \subseteq \omega\}$; after that, we use the standard method for extending P to all Borel subsets of Ω. By simple set in Ω we mean a union of intervals Γ_x from a finite set.

A measure P is computable if it is, at one time, lower and upper semicomputable.

For technical reazons, for any semimeasure P, we consider the maximal measure $\bar{P}$ such that $\bar{P} \leq P$. This measure satisfies

$$\bar{P}(x) = \inf_n \sum_{l(y)=n, x \subseteq y} P(y).$$

In general, the measure $\bar{P}$ is noncomputable (and it is not a probability measure). By (2), for each lower semicomputable semimeasure P, the inequality $c\bar{M}(A) \geq \bar{P}(A)$ holds for every Borel set A, where c is a positive constant.

In the manner of Levin's papers [4,5,6,13] (see also [12]), we consider combinations of probabilistic and deterministic processes as the most general class of processes for generating data. With any probabilistic process some computable probability distribution can be assigned. Any deterministic process is realized by means of an algorithm. Algorithmic processes transform sequences generated

by probabilistic processes into new sequences. More precise, a probabilistic computer is a pair (P, F), where P is a computable probability distribution, and F is a Turing machine supplied with an additional input tape. In the process of computation this machine reads on this tape a sequence ω distributed according to P and produces a sequence $\omega' = F(\omega)$ (A correct definition see in [4],[7], [9],[12]). So, we can compute the probability

$$Q(x) = P\{\omega \in \Omega : x \subseteq F(\omega)\}$$

that the result $F(\omega)$ of the computation begins with a finite sequence x. It is easy to see that $Q(x)$ is a semimeasure semicomputable from below.

Generally, the semimeasure Q can be not a probability distribution in Ω, since $F(\omega)$ may be finite for some infinite ω.

The converse result is proved in Zvonkin and Levin [13]: for every semimeasure $Q(x)$ semicomputable from below a probabilistic computer (L, F) exists such that

$$Q(x) = L\{\omega | x \subseteq F(\omega)\},$$

for all x, where $L(x) = 2^{-l(x)}$ is the uniform probability distribution in the set of all binary sequences.

Therefore, by (2) $M(x)$ defines an universal upper bound of the probability of generating x by probabilistic computers.

We refer readers to Li and Vitányi [7] and to Shen et al. [9] for the theory of algorithmic randomness. We use definition of a random sequence in terms of universal probability. Let P be some computable measure in Ω. The deficiency of randomness of a sequence $\omega \in \Omega$ with respect to P is defined as

$$d(\omega|P) = \sup_n \frac{M(\omega^n)}{P(\omega^n)}, \tag{3}$$

where $\omega^n = \omega_1\omega_2 \ldots \omega_n$. This definition leads to the same class of random sequences as the original Martin-Löf [8] definition. Let R_P be the set of all infinite binary sequences random with respect to a measure P

$$R_P = \{\omega \in \Omega : d(\omega|P) < \infty\}.$$

We also consider *parametric families* of probability distributions $P_\theta(x)$, where θ is a real number; we suppose that $\theta \in [0, 1]$. An example of such family is the Bernoulli family $B_\theta(x) = \theta^k(1-\theta)^{n-k}$, where n is the length of x and k is the number of ones in it.

We associate with a binary sequence $\theta_1\theta_2 \ldots$ a real number with the binary expansion $0.\theta_1\theta_2 \ldots$. When the sequence $\theta_1\theta_2 \ldots$ is computable or random with respect to some measure we say that the number $0.\theta_1\theta_2 \ldots$ is computable or random with respect to the corresponding measure in $[0, 1]$.

We consider probability distributions P_θ computable with respect to a parameter θ. Informally, this means that there exists an algorithm enumerating all pairs of rational numbers (r_1, r_2) such that $r_1 < P_\theta(x) < r_2$. This algorithm uses

an infinite sequence θ as an additional input; if some pair (r_1, r_2) was enumerated by this algorithm then only a finite initial fragment of θ was used in the process of computation (for correct definition, see also Shen et al. [9] and Vovk and V'yugin [10]).

Analogously, we consider parametric lower semicomputable semimeasures. It can be proved that there exist a universal parametric lower semicomputable semimeasure M_θ. This means that for any parametric lower semicomputable semimeasure R_θ there exists a positive constant C such that $CM_\theta(x) \geq R_\theta(x)$ for all x and θ.

The corresponding definition of randomness with respect to a family P_θ is obtained by relativization of (3) with respect to θ

$$d_\theta(\omega) = \sup_n \frac{M_\theta(\omega^n)}{P_\theta(\omega^n)}$$

(see also [3]). This definition leads to the same class of random sequences as the original Martin-Löf [8] definition relitivized with respect to a parameter θ.

For any θ, let

$$I_\theta = \{\omega \in \Omega : d_\theta(\omega) < \infty\}$$

be the set of all infinite binary sequences random with respect to the measure P_θ. In case of Bernoulli family, we call elements of this set *θ-Bernoulli sequences.*

3 Results

We need some statistical notions (see Cox and Hinkley [2]). Let P_θ be some computable parametric family of probability distributions. A function $\hat{\theta}(x)$ from Ξ to $[0, 1]$ is called *an estimate*. An estimate $\hat{\theta}$ is called *strongly consistent* if for any parameter value θ

$$\hat{\theta}(\omega^n) \to \theta$$

for P_θ-almost all ω. We suppose that ϵ and δ are rational numbers. An estimate $\hat{\theta}$ is called *effectively strongly consistent* if there exists a computable function $N(\epsilon, \delta)$ such that for any θ for all ϵ and δ

$$P_\theta\{\omega \in \Omega : \sup_{n \geq N(\epsilon,\delta)} |\hat{\theta}(\omega^n) - \theta| > \epsilon\} \leq \delta \tag{4}$$

The strong law of large numbers Borovkov [1] (Chapter 5)

$$B_\theta \left\{ \sup_{k \geq n} \left| \frac{1}{k} \sum_{i=1}^{k} \omega_i - \theta \right| \geq \epsilon \right\} < \frac{1}{\epsilon^4 n}$$

shows that the function $\hat{\theta}(\omega^n) = \frac{1}{n} \sum_{i=1}^{n} \omega_i$ is a computable strongly consistent estimate for the Bernoulli family B_θ.

Proposition 1. *For any effectively strongly consistent estimate* $\hat{\theta}$*,*

$$\lim_{n\to\infty} \hat{\theta}(\omega^n) = \theta$$

for each $\omega \in I_\theta$*.*

Proof. Let, for some θ, an infinite sequence ω be Martin-Löf random with respect to P_θ.

At first, we prove that $\lim_{n\to\infty} \hat{\theta}(\omega^n)$ exists. Let for $j = 1, 2, \ldots$,

$$W_j = \{\alpha \in \Omega : (\exists n, k \geq N(1/j, 2^{-(j+1)}))|\hat{\theta}(\alpha^n) - \hat{\theta}(\alpha^k)| > 1/j\}.$$

By (4) for any θ, $P_\theta(W_j) < 2^{-j}$ for all j. Define $V_i = \cup_{j>i} W_j$ for all i. By definition, for any θ, $P_\theta(V_i) < 2^{-i}$ for all i. Also, any set V_i can be represented as a recursively enumerable union of intervals of type Γ_x. To reduce this definition of Martin-Löf test to the definition of the test (3) define a sequence of uniform lower semicomputable parametric semimeasures

$$R_{\theta,i}(x) = \begin{cases} 2^i P_\theta(x) \text{ if } \Gamma_x \subseteq V_i \\ 0 \text{ otherwise} \end{cases}$$

and consider the mixture $R_\theta(x) = \sum_{i=1}^{\infty} \frac{1}{i(i+1)} R_{\theta,i}(x)$.

Suppose that the limit $\lim_{n\to\infty} \hat{\theta}(\omega^n)$ does not exist. Then for each sufficiently large j, $|\hat{\theta}(\omega^n) - \hat{\theta}(\omega^k)| > 1/j$ for infinitely many n, k. This implies that $\omega \in V_i$ for all i, and then for some positive constant c,

$$d_\theta(\omega) = \sup_n \frac{M_\theta(\omega^n)}{P_\theta(\omega^n)} \geq \sup_n \frac{R_\theta(\omega^n)}{cP_\theta(\omega^n)} = \infty,$$

i.e., ω is not Martin-Löf random with respect to P_θ.

Suppose that $\lim_{n\to\infty} \hat{\theta}(\omega^n) \neq \theta$. Then the rational numbers r_1, r_2 exist such that $r_1 < \lim_{n\to\infty} \hat{\theta}(\omega^n) < r_2$ and $\theta \notin [r_1, r_2]$. Since the estimate $\hat{\theta}$ is consistent, $P_\theta\{\alpha : r_1 < \lim_{n\to\infty} \hat{\theta}(\alpha^n) < r_2\} = 0$, and we can effectively (using θ) enumerate an infinite sequence of positive integer numbers $n_1 < n_2 < \ldots$ such that for

$$W'_j = \cup\{\Gamma_x : l(x) \geq n_j, r_1 < \hat{\theta}(x) < r_2\},$$

we have $P_\theta(W'_j) < 2^{-j}$ for all j. Define $V'_i = \cup_{j>i} W'_j$ for all i. We have $P_\theta(V'_i) \leq 2^{-i}$ and $\omega \in V'_i$ for all i. Then ω can not be Martin-Löf random with respect to P_θ. These two contradictions obtained above prove the proposition. △

The following theorem shows that, from the point of view of the philosophy explained above, P_θ-random sequences with "a prespecified" noncomputable parameter θ can not be obtained in any combinations of stochastic and deterministic processes.

Theorem 1. *Let a computable parametric family P_θ of probability distributions has an effectively strongly consistent estimate. Then for any θ, $\bar{M}(I_\theta) > 0$ if and only if θ is computable.*

Proof. If θ is computable then the probability distribution P_θ is also computable and by (2) $c\bar{M}(I_\theta) \geq P_\theta(I_\theta) = 1$, where c is a positive constant.

The proof of the converse assertion is more complicated. Let $\bar{M}(I_\theta) > 0$. There exists a simple set V (a union of a finite set of intervals) and a rational number r such that $\frac{1}{2}\bar{M}(V) < r < \bar{M}(I_\theta \cup V)$. For any finite set $X \subseteq \Xi$, let $\bar{X} = \cup_{x \in X} \Gamma_x$.

Let n be a positive integer number. Let us compute a rational approximation θ_n of θ up to $\frac{1}{2n}$. Using the exhaustive search, we find a finite set X_n of pairwise incomparable finite sequences of lengths $\geq N(1/n, 2^{-n})$ such that

$$\bar{X}_n \subseteq V, \quad \sum_{x \in X_n} M(x) > r,$$
$$|\hat{\theta}(x) - \hat{\theta}(x')| \leq \frac{1}{2n} \tag{5}$$

for all $x, x' \in X_n$. If any such set X_n will be found, we put $\theta_n = \hat{\theta}(x)$, where $x \in X_n$ is minimal with respect to some natural (lexicographic) ordering of all finite binary sequences.

Now we prove that for any n the set X_n exists. Since $\bar{M}(I_\theta \cap V) > r$, there exists a closed (in the topology defined by intervals Γ_x) set $E \subseteq I_\theta \cap V$ such that $\bar{M}(E) > r$. Consider the function

$$f_k(\omega) = \inf\{n : n \geq k, |\hat{\theta}(\omega^n) - \theta| \leq \frac{1}{4n}\}.$$

By Proposition 1 this function is continuous on Ω and, since the set E is compact, it is bounded on E. Hence, for any k, a finite set $X \subseteq \Xi$ exists consisting of pairwise incomparable sequences of length $\geq k$ such that $E \subseteq \bar{X}$ and $|\hat{\theta}(x) - \hat{\theta}(x')| \leq \frac{1}{2n}$ for any $x, x' \in X$. Since $E \subseteq \bar{X}$, we have $\sum_{x \in X} M(x) > r$. Therefore, the set X_n can be found by exhaustive search.

Lemma 1. *For any Borel set $V \subseteq \Omega$, $\bar{M}(V) > 0$ and $V \subseteq I_\theta$ imply $P_\theta(V) > 0$.*

Proof. By definition of M_θ any computable parametric measure P_θ is absolutely continuous with respect to the measure $\bar{M}_\theta$, and so, we have representation

$$P_\theta(X) = \int_X \frac{dP_\theta}{d\bar{M}_\theta}(\omega) d\bar{M}_\theta(\omega), \tag{6}$$

where $\frac{dP_\theta}{d\bar{M}_\theta}(\omega)$ is the Radon-Nicodim derivative; it exists for $\bar{M}_\theta$-almost all ω.

By definition we have for $\bar{M}_\theta$-almost all $\omega \in I_\theta$

$$\frac{dP_\theta}{d\bar{M}_\theta}(\omega) = \lim_{n \to \infty} \frac{P_\theta}{\bar{M}_\theta}(\omega^n) \geq \liminf_{n \to \infty} \frac{P_\theta}{\bar{M}_\theta}(\omega^n) \geq C_{\theta,\omega} > 0. \tag{7}$$

By definition $c_\theta \bar{M}_\theta(X) \geq \bar{M}(X)$ for all Borel sets X, where c_θ is some positive constant (depending on θ). Then by (6) and (7) the inequality $\bar{M}(X) > 0$ implies $P_\theta(X) > 0$ for each Borel set X. $\triangle$

We rewrite (4) in a form

$$E_n = \{\omega \in \Omega : \sup_{N \geq N(1/(2n), 2^{-n})} |\hat{\theta}(\omega^N) - \theta| \geq \frac{1}{2n}\} \tag{8}$$

and $P_\theta(E_n) \leq 2^{-n}$ for all n. We prove that $X_n \not\subseteq E_n$ for almost all n. Suppose that the opposite assertion holds. Then there exists an increasing infinite sequence of positive integer numbers $n_1, n_2 \ldots$ such that $X_{n_i} \subseteq E_{n_i}$ for all $i = 1, 2, \ldots$. This implies $P_\theta(X_{n_i}) \leq 2^{-n_i}$ for all i. For any k, define $U_k = \cup_{i \geq k} X_{n_i}$. Clearly, we have for all k, $\bar{M}(\bar{U}_k) > r$ and $P_\theta(\bar{U}_k) \leq \sum\limits_{i \geq k} 2^{-n_i} \leq 2^{-n_k+1}$. Let $U = \cap U_k$. Then $P_\theta(U) = 0$ and $\bar{M}(U) \geq r > \frac{1}{2}\bar{M}(V)$. From $U \subseteq V$ and $\bar{M}(I_\theta \cap V) > \frac{1}{2}\bar{M}(V)$ the inequality $\bar{M}(I_\theta \cap U) > 0$ follows. Then the set $I_\theta \cap U$ consists of P_θ-random sequences, $P_\theta(I_\theta \cap U) = 0$ and $\bar{M}(I_\theta \cap U) > 0$. This is a contradiction with Lemma 1.

Let $X_n \not\subseteq E_n$ for all $n \geq n_0$. Let also, a finite sequence $x_n \in X_n$ is defined such that

$$\Gamma_{x_n} \cap (\Omega \setminus E_n) \neq \emptyset.$$

Then from $l(x_n) \geq N(\frac{1}{2n}, 2^{-n})$ the inequality

$$\left| \frac{1}{l(x_n)} \sum_{i=1}^{l(x_n)} (x_n)_i - \theta \right| < \frac{1}{2n}$$

follows. By (5) we obtain $|\theta_n - \theta| < \frac{1}{n}$. This means that the real number θ is computable. Theorem is proved. $\triangle$

Bayesian mixture of computable (with respect to θ) probability measures P_θ using a computable prior on θ gives to P_θ-random sequences "the empirical meaning". Let Q be a computable probability distribution on θ (i.e., in the set Ω). Then the Bayesian mixture

$$P(x) = \int P_\theta(x) dQ(\theta)$$

is also computable.

Recall that R_Q is the set of all infinite sequences Martin-Löf random with respect to a computable probability measure Q. Obviously, $P(\cup_{\theta \in R_Q} I_\theta) = 1$, and then $\bar{M}(\cup_{\theta \in R_Q} I_\theta)) > 0$. Moreover, it follows from Corollary 4 of Vovk and V'yugin [10]

Theorem 2. *For any computable measure Q a sequence ω is random with respect to the Bayesian mixture P if and only if ω is random with respect to a measure P_θ for some θ random with respect to the measure Q; in other words,*

$$R_P = \cup_{\theta \in R_Q} I_\theta.$$

Notice, that any computable θ is Martin-Löf random with respect to the computable probability distribution concentrated on this sequence.

The following Theorem 3 shows that the Bayesian approach is insufficient to cover all possible "meaningful" cases: a probabilistic machine can be constructed, which with probability close to one outputs a random θ-Bernoulli sequence, where the parameter θ is not random with respect to each computable probability distribution.

Let $\mathcal{P}$ be the set of all computable probability measures in Ω, and let

$$St = \cup_{P \in \mathcal{P}} R_P$$

be the set of all sequences Martin-Löf random with respect to allcomputable probability measures. We call these sequences - *stochastic*. Its complement $NSt = \Omega \setminus St$ consists of all sequences nonrandom with respect to all computable probability measures. We call them *nonstochastic*.

We proved in V'yugin [11], [12] that $\bar{M}(NSt) > 0$. Namely, the following proposition holds[1] .

Proposition 2. *For any ϵ, $0 < \epsilon < 1$, a lower semicomputable semimeasure Q can be constructed such that*

$$\bar{Q}(\Omega) > 1 - \epsilon, \tag{9}$$

$$NSt = \cup_{Q(x)>0} \Gamma_x. \tag{10}$$

We show that this result can be extended to parameters of the Bernoulli sequences.

Theorem 3. *Let I_θ be the set of all θ-Bernoulli sequences. Then*

$$\bar{M}(\cup_{\theta \in NSt} I_\theta) > 0.$$

In terms of probabilistic computers, for any ϵ, $0 < \epsilon < 1$, a probabilistic machine (L, F) can be constructed, which with probability $\geq 1-\epsilon$ generates an θ-Bernoulli sequence, where $\theta \in NSt$ (i.e., θ is nonstochastic).

Proof. For any $\epsilon > 0$, $0 < \epsilon < 1$, we define a lower semicomputable semimeasure P such that

$$\bar{P}(\cup_{\theta \in NSt} I_\theta) > 1 - \epsilon.$$

The proof of the theorem is based on Proposition 2. The property (10) can be rewritten as: $Q(\omega^n) = 0$ for all sufficiently large n if and only if $\omega \in St$ (i.e., ω is Martin-Löf random with respect to some computable probability measure).

For the measure

$$R^-(x) = \int B_\theta(x) d\bar{Q}(\theta), \tag{11}$$

[1] We also prove in these papers that $M(\Omega \setminus \bar{R}_L) > 0$, where $\bar{R}_L$ is the set of all sequences Turing reducible to sequences from R_L random with respect to the uniform measure L. By [13] it holds $St \subseteq \bar{R}_L$. The corresponding strengthening of the Theorem 3 is: $\bar{M}(\cup_{\theta \in \Omega \setminus \bar{R}_L} I_\theta) > 0$.

where B_θ is the Bernoulli measure, by (9) we have $R^-(\Omega) > 1 - \epsilon$, and by (10) we have $R^-(\cup_{\theta \in St} I_\theta) = 0$.

Unfortunately, we can not conclude that $c\bar{M} \geq R^-$ for some constant c, since the measure R^- is not represented in the form $R^- = \bar{P}$ for some lower semicomputable semimeasure P. To overcome this problem, we consider some semicomputable approximation of this measure.

For any finite binary sequences α and x, let $B^-_\alpha(x) = (\theta^-)^K(1-\theta^+)^{N-K}$, where N is the length of x and K is the number of ones in it, θ^- is the left side of the subinterval corresponding to the sequence α and θ^+ is its right side. By definition, $B^-_\alpha(x) \leq B_\theta(x)$ for all $\theta^- \leq \theta \leq \theta^+$.

Suppose that ϵ is a rational number. Let $Q^s(x)$ be equal to the maximal rational number $r < Q(x)$ computed in s steps of enumeration of $Q(x)$ from below. Using (9), we can define for $n = 1, 2, \ldots$ and for any x of length n a computable sequence of positive integer numbers $s_x \geq n$ and a sequence of finite binary sequences $\alpha_{x,1}, \alpha_{x,2}, \ldots \alpha_{x,k_x}$ of length $\geq n$ such that the function $P(x)$ defined by

$$P(x) = \sum_{i=1}^{k_x} B^-_{\alpha_{x,i}}(x) Q^{s_x}(\alpha_{x,i}) \tag{12}$$

is a semimeasure, i.e., such that condition (1) holds for all x, and such that

$$\sum_{l(x)=n} P(x) > 1 - \epsilon \tag{13}$$

holds for all n. These sequences exist, since the limit function R^- defined by (11) is a measure satisfying $R^-(\Omega) > 1 - \epsilon$.

By definition the semimeasure $P(x)$ is lower semicomputable. Then $cM(x) \geq P(x)$ holds for all $x \in \Xi$, where c is a positive constant.

To prove that $\bar{P}(\Omega \setminus \cup_\theta I_\theta) = 0$ we consider some probability measure $Q^+ \geq Q$. Since (1) holds, it is possible to define some noncomputable measure Q^+ satisfying these properties in many different ways. Define the mixture of the Bernoulli measures with respect to Q^+

$$R^+(x) = \int B_\theta(x) dQ^+(\theta). \tag{14}$$

By definition $R^+(\Omega \setminus \cup_\theta I_\theta) = 0$. Using definitions (12) and (14), it can be easily proved that $\bar{P} \leq R^+$. Then $\bar{P}(\Omega \setminus \cup_\theta I_\theta) = 0$. By (10) we have $\bar{P}(\cup_{\theta \in St} I_\theta) = 0$. By (13) we have $\bar{P}(\Omega) > 1 - \epsilon$. Then $\bar{P}(\cup_{\theta \in NSt} I_\theta) > 0$. Therefore, $\bar{M}(\cup_{\theta \in NSt} I_\theta) > 0$. △

Acknowledgements

This work was partially supported by Russian foundation for fundamental research: 06-01-00122.

References

1. Borovkov, A.A.: Theory of Probability. Nauka. (Probability Theory. Gordon and Breach 1998) (1999)
2. Cox, D.R., Hinkley, D.V.: Theoretical Statistics. Chapman and Hall, London (1974)
3. Levin, L.A.: On the notion of random sequence. Soviet Math. Dokl. 14, 1413–1416 (1973)
4. Levin, L.A.: Laws of information conservation (non-growth) and aspects of the foundation of probability theory. Problems Inform. Transmission. 10, 206–210 (1974)
5. Levin, L.A., V'jugin, V.V.: Invariant Properties of Informational Bulks. In: Gruska, J. (ed.) Mathematical Foundations of Computer Science 1977. LNCS, vol. 53, pp. 359–364. Springer, Heidelberg (1977)
6. Levin, L.A.: Randomness conservation inequalities; information and independence in mathematical theories. Inform. and Control. 61, 15–37 (1984)
7. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, New York (1997)
8. Martin-Löf, P.: The definition of random sequences. Inform. and Control. 9, 602–619 (1966)
9. Shen, A., Uspensky, V.A., Vereshchagin, N.K.: Lecture Notes on Kolmogorov Complexity (2007), http://lpes.math.msu.su/ver/kolm-book
10. Vovk, V.G., V'yugin, V.V.: On the empirical validity of the Bayesian rule. J. R. Statist. Soc. B. 55, 317–351 (1993)
11. V'yugin, V.V.: On Turing invariant sets. Soviet Math. Dokl. 17, 1090–1094 (1976)
12. V'yugin, V.V.: The algebra of invariant properties of binary sequences. Problems Inform. Transmission. 18, 147–161 (1982)
13. Zvonkin, A.K., Levin, L.A.: The complexity of finite objects and the algorithmic concepts of information and randomness. Russ. Math. Surv. 25, 83–124 (1973)

An Efficient Algorithm for Zero-Testing of a Lacunary Polynomial at the Roots of Unity

Sergey P. Tarasov* and Mikhail N. Vyalyi**

Dorodnitsyn Computing Center of RAS,
Vavilova, 40, Moscow 119991, Russia
starasov@newmail.ru, vyalyi@mccme.ru

Abstract. We present a polynomial time algorithm for the following problem: to check whether a lacunary polynomial $f(x)$ vanishes at a given primitive nth root of unity ζ_n. A priori $f(\zeta_n)$ may be nonzero and doubly exponentially small in the input size. Only exponential algorithms were known for this problem. The existence of an efficient procedure in the case of factored n was conjectured by D. Plaisted in 1984. As a consequence we show that the problem of the divisibility testing of a lacunary polynomial by some cyclotomic polynomial belongs to the complexity class $\mathcal{NP}$.

Keywords: algorithm, cyclotomic polynomial, root of unity, sparse representation.

Let $\zeta_n = e^{2\pi i/n}$ be an nth primitive root of unity. *A vanishing sum* of roots of unity has the form

$$\sum_{j=0}^{n-1} a_j \zeta_n^j = 0 \tag{1}$$

where the coefficients a_j are integers.

There are many results on vanishing sums of roots of unity. Rédei [9] and Schoenberg [10] described the lattice of vanishing sums (see also Rédei [8], de Bruijn [1], Lam and Leung [5]). Conway and Jones [2] gave a lower bound on the size of the support set of a minimal vanishing sum with nonnegative coefficients. The paper by Lam and Leung [5] contains an exact characterization of the set of ℓ^1-norms of vectors of the coefficients of vanishing sums with nonnegative coefficients. Steinberger [11] developed a method for construction of minimal sums with large coefficients.

In this paper we examine the algorithmic aspects of zero-testing of sums of roots of unity and consider the following problem. Given an integer n, a finite *support* set J of natural numbers and a set of integer coefficients a_j, $j \in J$ check the equality

$$\sum_{j \in J} a_j \zeta_n^j = 0 \ . \tag{2}$$

* The work is supported by the RFBR grant 05–01–01019.
** The work is supported by the RFBR grants 05–01–01019, 05–01–02803 and the grant NS 5833.2006.1.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 397–406, 2007.

Due to irreducibility of cyclotomic polynomials $\Phi_n(x)$ the equality (2) is equivalent to the divisibility of a lacunary polynomial

$$f(x) = \sum_{j \in J} a_j x^j \tag{3}$$

by the cyclotomic polynomial $\Phi_n(x)$. Hereinafter we call this problem the *cyclotomic test* (CT for brevity).

Note that a linear combination of roots of unity with integer coefficients is an algebraic number. So, the straightforward way to check the equality (2) is to compute a rational approximation of its left-hand side and then to compare it with zero. An accuracy of approximation required for correctness of this algorithm is called a *separation bound*. The standard separation bound (see e.g. Mignotte [6]) is doubly exponentially small w.r.t. the input size of the problem CT. It means that the algorithm above uses exponential space and hence, exponential time. Specific separation bounds for linear combinations of roots of unity are unknown.

It is shown by Plaisted [7] that CT is in co-$\mathcal{NP}$ (the related problem is called SPARSE-POLY-NONROOT there).[1]

A related and a more general problem is studied by Filaseta and Schinzel [4]: to check the divisibility of a given lacunary polynomial by some cyclotomic polynomial. Formally stated this task consists in checking whether a lacunary polynomial $f(x) = \sum_{j \in J} a_j x^j$ represented by a finite support set J of natural numbers and a set of integer coefficients a_j, $j \in J$, is divisible by some cyclotomic polynomial. We call this problem the general cyclotomic test (GCT for brevity). The Plaisted's result mentioned above implies GCT $\in \Sigma_2$. The running time of the algorithm in [4] is subexponential

$$O(\exp[(2 + o(1))\sqrt{|J|/\log|J|}(\log|J| + \log\log\deg f)]\log H(f)),$$

where $H(f) \stackrel{\text{def}}{=} \max_{j \in J} |a_j| + 1$. As a subroutine the procedure in [4] uses a subexponential algorithm for the cyclotomic test, *provided the prime power decomposition of n is given.*

A more sophisticated algorithm for GCT is described in a more recent paper by Filaseta, Granville, and Schinzel [3]. However, it uses the same subexponential subroutine for the cyclotomic test. So, this algorithm cannot be applied to prove that GCT $\in \mathcal{NP}$.

Our contribution is a polynomial time algorithm for the cyclotomic test in the case of a general (not factored) n. As a direct consequence we show that GCT is in $\mathcal{NP}$.

Now we state our results more formally. A *sparse representation* of a polynomial $f(x) = \sum_{i=0}^{d} a_j x^j$ with integer coefficients is a list of pairs (a_j, j) for $a_j \neq 0$. Integers in a sparse representation are written in binary. A vector of the coefficients of a polynomial $f(x)$ is denoted by coef f. A *support* set supp a of a vector

[1] There is even a more resolute statement in [7, p. 132]:"The author believes he has a method for solving SPARSE-POLY-NONROOT in polynomial time if the prime factorization of M is given." To our knowledge this result is unpublished.

$a = (a_0, \ldots, a_d)^T$ is a set $\{j : a_j \neq 0\}$. So the size of a sparse representation of a polynomial $f(x)$ is $O(m(\log H(f) + \log \deg f))$, where $m = |\operatorname{supp} \operatorname{coef} f|$.

The decision problem CT is stated as follows. The input is a sparse representation of a polynomial $f(x)$ and an integer n written in binary. The output is 'yes' if $f(\zeta_n) = 0$ and 'no' otherwise. We associate with the problem CT a language $\mathrm{CT} \subset \{0,1\}^*$ in the standard manner.

Note that the cardinality of the support set m and the maximal bit length of integers contained in the input $L = \max(\log H(f), \log \deg f, \log n)$ do not exceed the input size. We will use the parameters m, L and $\log n$ in bounds of running time below.

Theorem 1. $\mathrm{CT} \in \mathcal{P}$

The decision problem GCT is stated as follows. The input is a sparse representation of a polynomial $f(x)$. The output is 'yes' if there is an integer m such that $\Phi_m(x) \mid f(x)$ and 'no' otherwise. We also denote by GCT the language associated with the problem GCT.

As a direct consequence of Theorem 1 we obtain

Theorem 2. $\mathrm{GCT} \in \mathcal{NP}$

1 Algebraic Facts About Vanishing Sums of Roots of Unity

Let

$$n = p_1^{t_1} \cdot p_2^{t_2} \cdot \ldots \cdot p_r^{t_r} \tag{4}$$

be the prime power decomposition of n. For a positive integer n let V_n be an n-dimensional vector space over the field $\mathbb{Q}$ of rationals equipped with the canonical basis $\{e_0, e_1, \ldots, e_{n-1}\}$.

Let $\varphi\colon V_n \to \mathbb{Q}[\zeta_n]$ be an evaluation map — a $\mathbb{Q}$-linear map acting on the basis vectors by the rule

$$\varphi(e_k) = \zeta_n^k \ . \tag{5}$$

We denote by X_n the kernel of φ.

We follow the paper by Lam and Leung [5] and give a description of X_n using a tensor decomposition of V_n.

At first, the Chinese remainder theorem implies a decomposition

$$V_n \cong V_{p_1^{t_1}} \otimes \cdots \otimes V_{p_r^{t_r}} \ . \tag{6}$$

The isomorphism acts on the basis vectors by the rule

$$e_i \mapsto e_{i \bmod p_1^{t_1}} \otimes \cdots \otimes e_{i \bmod p_r^{t_r}} \ . \tag{7}$$

We also split V_{p^t} by another isomorphism

$$V_{p^t} \cong V_p \otimes V_{p^{t-1}} \tag{8}$$

which is defined by

$$e_i \mapsto e_j \otimes e_k, \quad \text{where } i = p^{t-1}j + k \ . \tag{9}$$

Substituting (8) to (6) and rearranging factors we get a decomposition

$$\begin{aligned} V_n \cong & V_{p_1} \otimes \cdots \otimes V_{p_r} \otimes V_{p_1^{t_1-1}} \otimes \cdots \otimes V_{p_r^{t_r-1}} = \\ & Q_1 \otimes Q_2 \otimes \ldots Q_r \otimes Q_{r+1} \otimes \cdots \otimes Q_{2r} \ . \end{aligned} \tag{10}$$

Hereinafter we use a notation $Q_i = V_{p_i}$ and $Q_{r+i} = V_{p_i^{t_i-1}}$ for $1 \le i \le r$.

Define the vector $\hat{1} \in V_p$ by

$$\hat{1} = \sum_{j=0}^{p-1} e_j \ . \tag{11}$$

The next theorem is a weaker form of theorem 2.2 from the paper by Lam and Leung [5]. (The theorem 2.2 is attributed to Rédei, de Bruijn and Schoenberg there.)

Theorem 3. *$X_n = \operatorname{Ker} \varphi$ is a sum of subspaces X_n^i, where*

$$X_n^i = Q_1 \otimes \cdots \otimes Q_{i-1} \otimes \mathbb{Q}\hat{1} \otimes Q_{i+1} \otimes \cdots \otimes Q_{2r}, \quad 1 \le i \le r \ . \tag{12}$$

Let us use an inner product in V_n such that the canonical basis is orthonormal with respect to this product. Then the dual space to X_n can be identified with the orthogonal complement $X_n^\perp$. In order to describe a basis of $X_n^\perp$ we define 'cuboids' of dimension r (see the Steinberger's paper [11]). Let u^0, u^1 be integer sequences $(u_{j,0})$, $(u_{j,1})$ of length r such that $0 \le u_{j,\alpha} \le p_j - 1$ and $u_{j,0} \ne u_{j,1}$ for any j. A *cuboid* $Q(u^0, u^1)$ parameterized by the sequences u^0, u^1 is a vector in $Q_1 \otimes \cdots \otimes Q_r$ such that

$$Q(u^0, u^1) = \bigotimes_j \left(e_{u_{j,0}} - e_{u_{j,1}} \right) = \sum_{\alpha \in \{0,1\}^r} (-1)^{\|\alpha\|} e_{u_{1,\alpha_1}} \otimes e_{u_{2,\alpha_2}} \otimes \cdots \otimes e_{u_{r,\alpha_r}} \ , \tag{13}$$

where $\|\alpha\|$ is the Hamming norm of the Boolean vector α (the number of 1's in α).

The *cuboid space* Q is the subspace spanned by cuboids in $Q_1 \otimes \cdots \otimes Q_r$.

Theorem 4 (Steinberger, [11]). *Let n be a squarefree integer. Then $X_n^\perp = Q$.*

Remark 1. A different form of Theorem 4 is contained in papers by Rédei [9] and by Conway and Jones [2]. See also Titova and Shevchenko [12].

In general case note that the first part $Q_1 \otimes \ldots \otimes Q_r$ of the decomposition (10) corresponds to the decomposition of the maximal squarefree divisor of n. Taking into account Theorem 3 we get a generalization of Theorem 4.

Theorem 5. *$X_n^\perp = Q \otimes Q_{r+1} \otimes \cdots \otimes Q_{2r}$ for any integer n.*

An extended cuboid is a tensor product of a cuboid by any vector from the canonical basis in $Q_{r+1} \otimes \cdots \otimes Q_{2r}$. In this notation Theorem 5 states that $X_n^{\perp}$ is spanned by the extended cuboids.

Now we specify a family of bases in the cuboid space.

Lemma 1. *Let z be an integer sequence of length r such that $0 \le z_j \le p_j - 1$. Then the set $\mathcal{Q}_z$ of cuboids $Q(z,u)$ for all integer sequences $u = (u_1, \ldots, u_r)$, $0 \le u_i \le p_i - 1$, $u_i \ne z_i$, $i = 1, \ldots, r$ forms a basis in the cuboid space.*

Proof. The vector $\otimes_j e_{u_j}$, $u_j \ne z_j$, is orthogonal to a cuboid $Q(z,v)$ unless $v = u$. So, cuboids in the set $\mathcal{Q}_z$ are linearly independent. On the other hand, an arbitrary cuboid can be expressed as a linear combination of cuboids taken from $\mathcal{Q}_z$:

$$Q(x,y) = \bigotimes_j (e_{x_j} - e_{y_j}) = \bigotimes_j (-(e_{z_j} - e_{x_j}) + (e_{z_j} - e_{y_j})) = \sum_{S \subseteq \{0,1\}^r} (-1)^{|S|} \bigotimes_j (e_{z_j} - e_{u(S,j)}) \ , \tag{14}$$

where $u(S,j) = x_j$ if $j \in S$ and $u(S,j) = y_j$ otherwise. □

We use various bases $\mathcal{Q}_z$ in the proof of Theorem 1 in Sect. 3. In the next section we will restrict our consideration to the basis $\mathcal{Q}_0$ formed by cuboids $Q(0^r, u)$. We call these cuboids the *standard cuboids*.

2 An Algorithm in the Case of Factored n

In this section we present a polynomial time algorithm for the restricted problem CT^f. In the problem CT^f the input consists of a sparse representation of a polynomial $f(x)$, the binary representation of an integer n and an auxiliary string w. The output is 'yes' if $f(\zeta_n) = 0$ and the string w is the prime power decomposition of the integer n. Otherwise, the output is 'no'.

Lemma 2. $\mathrm{CT}^f \in \mathcal{P}$

Remark 2. An artificial form of the problem CT^f needs some justification. If the binary representation of n were not a part of the input then the algorithm below would not run in polynomial time since the prime power representation of an integer can be exponentially shorter than the binary representation for the same integer.

On the other hand, if n is a part of the input then it takes a polynomial time to verify whether the string w is the prime power decomposition of n. Indeed, the size of the prime power decomposition of an integer n is $O(\log n)$. For verification one needs to multiply $O(\log n)$ integers each written in $O(\log n)$ bits and to check primality of each factor. All these operations can be done in polynomial time.

Remark 3. In both problems CT and CT^f we do not require that $n \geq \deg f(x)$. A simple preprocessing procedure can be applied to guarantee this condition: it is sufficient to change each exponent $j \in \operatorname{supp} \operatorname{coef} f$ by the residue modulo n summing up similar terms. We always assume this preprocessing before running the main body of an algorithm. Note that the preprocessing requires $O(m)$ arithmetic operations with $O(L)$-bit integers.

The rest of this section contains the proof of Lemma 2.

We choose the basis of $X_n^\perp$ formed by the standard extended cuboids, i.e. the tensor products of cuboids $Q(0^r, u)$ and vectors from the canonical basis in $Q_{r+1} \otimes \cdots \otimes Q_{2r}$. Vectors in the basis are naturally indexed by two integer sequences u, v of length r such that $1 \leq u_j \leq p_j - 1$ and $0 \leq v_j \leq p_j^{t_j - 1} - 1$ for any $1 \leq j \leq r$.

To check the equality $f(\zeta_n) = 0$ the algorithm checks that the vector $a = \operatorname{coef} f$ is orthogonal to the subspace $X_n^\perp$. It is possible to compute the inner product of a by any standard extended cuboid in polynomial time since the components from the complement to $\operatorname{supp} a$ do not affect the inner product and any cuboid's component can be computed in polynomial time (see Lemma 3). But the difficulty remains as we need to check all (exponentially many) cuboids.

The idea of the algorithm is simple. Let E be an $m \times N$ matrix and a be an $m \times 1$ column, $a = (a_1, \ldots, a_m)^T$. Let $r_j(E)$ be the jth row of the matrix E, and let $c_k(E)$ be the kth column of the matrix E. The direct computation

$$\langle a, c_k(E) \rangle = \sum_j a_j E_{j,k} = \Big(\sum_j a_j r_j(E) \Big)_k \tag{15}$$

shows that $\langle a, c_k(E) \rangle = 0$ for all k iff

$$\sum_j a_j r_j(E) = 0. \tag{16}$$

The latter is equivalent to

$$\sum_j a_j \langle r_j(E), r_k(E) \rangle = 0 \quad \text{for all } k \tag{17}$$

(note that the vector $\sum_j a_j r_j(E)$ lies in the subspace spanned by the rows $r_j(E)$).

Equation (17) can be rewritten as $a^T G = 0$, where $G = (G_{j,k})$ is the *Gram matrix* for the rows of the matrix E, i.e. $G_{j,k} = \langle r_j(E), r_k(E) \rangle$. Thus the conditions (17) can be verified in polynomial time provided a polynomial time subroutine for computing the elements of the Gram matrix is given.

To apply the above observation to the problem CT^f we introduce the matrix E whose columns are restrictions of the extended standard cuboids to $\operatorname{supp} a$. The matrix elements of the matrix E are indexed by pairs $(j, (u, v))$, where $j \in \operatorname{supp} a$ and (u, v) is an indexing pair for an extended standard cuboid.

We need to show that the elements of the Gram matrix for the rows of the matrix E can be computed efficiently. We start from an explicit expression for the matrix elements $E_{j,(u,v)}$.

Let τ be a modular decomposition map corresponding to (10):

$$\tau\colon j \mapsto \tau_j = (\tau_{j,s})\Big|_{s=1}^{2r} \ , \tag{18}$$

where

$$\tau_{j,s} = \begin{cases} (j \bmod p_s^{t_s} - j \bmod p_s^{t_s-1})/p_s^{t_s-1}, & 1 \le s \le r \ , \\ j \bmod p_s^{t_s-1}, & r < s \le 2r \ . \end{cases} \tag{19}$$

Let w_j be the number of non-zero $\tau_{j,s}$ for $1 \le s \le r$.

Lemma 3. *In the notation above* $E_{j,(u,v)} = (-1)^{w_j}$ *if* τ_j *can be obtained from* (u, v) *by replacing* $r - w_j$ *elements in* u *by zeroes. Otherwise,* $E_{j,(u,v)} = 0$.

Proof. The definition (13) of a cuboid implies that all components of an extended standard cuboid are in $\{-1, 0, +1\}$. Any non-zero component can be obtained from the indexing pair (u, v) in three steps: (1) choose a subset S of $\{1, \ldots, r\}$; (2) for each index $j \in S$ replace the element u_j by 0 to get the sequence $u'v$ of length $2r$; (3) apply τ^{-1} to the sequence $u'v$. The resulting index $j = \tau^{-1}(u'v)$ points to the component whose value is $(-1)^{|S|}$. □

From Lemma 3 we get the formula for $G_{j,k} = \langle r_j(E), r_k(E)\rangle$:

$$G_{j,k} = (-1)^{w_j+w_k} n_{j,k} \ , \tag{20}$$

where $n_{j,k}$ is the number of sequences $\nu = (\nu_1, \ldots, \nu_{2r})$ such that

1. $1 \le \nu_s \le p_s - 1$ for any $1 \le s \le r$;
2. $0 \le \nu_s \le p_s^{t_s-1} - 1$ for any $r < s \le 2r$;
3. if $\tau_{j,s} \ne 0$ then $\tau_{j,s} = \nu_s$ for any $1 \le s \le r$;
4. if $\tau_{k,s} \ne 0$ then $\tau_{k,s} = \nu_s$ for any $1 \le s \le r$;
5. $\tau_{k,s} = \nu_s = \tau_{j,s}$ for any $r < s \le 2r$.

These conditions imply the formula

$$n_{j,k} = \prod_{s=1}^{r} n_{j,k,s} \prod_{s=r+1}^{2r} \delta(\tau_{j,s}, \tau_{k,s}) \ , \tag{21}$$

where

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \ , \\ 0 & \text{otherwise} \ , \end{cases} \tag{22}$$

and

$$n_{j,k,s} = \begin{cases} p_s - 1 & \text{if } \tau_{j,s} = \tau_{k,s} = 0 \ , \\ 1 & \text{if } \tau_{j,s}\tau_{k,s} = 0 \text{ and } (\tau_{j,s})^2 + (\tau_{k,s})^2 \ne 0 \ , \\ 1 & \text{if } \tau_{j,s} = \tau_{k,s} \ne 0 \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{23}$$

It follows from (21–23) that $n_{j,k} < \prod_{j=1}^{s}(p_j - 1) < n$. As in Remark 2, to find $n_{j,k}$ one needs to multiply $O(\log n)$ integers each written in $O(\log n)$ bits.

There are $O(m^2 \log n)$ arithmetic operations with $O(L)$-bit integers in the main body of the algorithm. Since by Remarks 2 and 3 the verification and the preprocessing are also polynomial we obtain an overall polynomial algorithm for CT^f.

3 Polynomial Time Algorithm for CT

To use the algorithm described in the previous section for the problem CT one needs to factorize n. It is believed that factorization is a hard computational task. Instead of the direct use of the algorithm, we modify it maintaining the same idea of the Gram matrix computation. To simplify the computation we change basis in the cuboids space.

Consider a partial prime decomposition

$$n = p_1^{t_1} \dots p_\ell^{t_\ell} q \ , \tag{24}$$

where $p_i \le m$ and all prime divisors of q are greater than m. (Recall that m is the number of monomials in the input polynomial $f(x)$.)

Let z be an integer sequence of length r such that $z_s = 0$ for $1 \le s \le \ell$ and $z_s \ne \tau_{j,s}$, $0 \le z_s \le p_s - 1$ for all $\ell < s \le r$, $j \in \operatorname{supp}\operatorname{coef} f$. Such sequence exists since we assume that prime divisors of q are greater than m and the cardinality of a set $\{\tau_{j,s}\}\big|_{j=1}^{m}$ does not exceed m for any s.

Let us change the standard basis $\mathcal{Q}_0$ by the basis $\mathcal{Q}_z$ for the sequence z.

Lemma 4. *The Gram matrix for the rows of the matrix E in the basis $\mathcal{Q}_z$ is block diagonal: if $j - k \ne 0 \pmod q$ then $G_{j,k} = 0$.*

Lemma 4 is proved by repeating the arguments from Sect. 2. Namely, substituting the sequence z for 0^r one can repeat all calculations from Sect. 2 in the basis $\mathcal{Q}_z$. An analogue of (23) implies that $n_{j,k,s} = 0$ if $j - k \not\equiv 0 \pmod{p_s^{t_s}}$, $s > \ell$. Now Lemma 4 follows from the Chinese remainder theorem.

Let S be a set $\{k :$ exists j such that $a_j \ne 0$ and $j \equiv k \pmod q\}$. By construction, $|S| \le m$. By Lemma 4 the problem CT is decomposed into $|S|$ problems of type CT^f.

The algorithm for the problem CT works in five stages:

1. Find the set $p_1, \dots, p_\ell$ of all primes in the range $\{1, \dots, m\}$.
2. Compute the partial decomposition (24).
3. Form the list of polynomials $f_k(x)$, $k \in S$. Each polynomial f_k consists of all monomials $a_i x^i$ of the input polynomial $f(x)$ such that $i \equiv k \pmod q$.
4. For each $k \in S$ solve the instance of the problem CT^f with the input consisting of the polynomial $f_k(x)$, the integer $n' = n/q$ and the string representing the prime power decomposition $n' = p_1^{t_1} \dots p_\ell^{t_\ell}$.
5. If all answers at the previous stage are 'yes' then the algorithm outputs 'yes'. Otherwise it outputs 'no'.

The algorithm described above runs in polynomial time. The first stage can be done by Eratosthenes' sieve in time polynomial in m. The straightforward realization of the second stage requires $O(m \log n)$ arithmetic operations with $O(\log n)$-bit integers. The length of the list formed at the third stage is at most m. So, the third stage requires $O(m)$ arithmetic operations with $O(L)$-bit integers. At the fourth stage the algorithm from the Sect. 2 is applied. Note that the string verification procedure of this algorithm can be omitted.

Thus the proof of Theorem 1 is complete.

4 Concluding Remarks

Recently we proved that GCT is computationally hard unless $\mathcal{NP} \subseteq \mathcal{BPP}$. This result will appear in the full version of the paper.

Our technique seems to be inapplicable to the more general problem of testing the inequality $|f(\zeta_n)| > \varepsilon$ (a rational ε is a part of the input; ε is represented by a pair of integers written in binary). The inequality testing is related to the problem of obtaining nontrivial lower bounds for a nonzero value of a lacunary polynomial at a root of unity. Recall that any implication *'if* $f(\zeta_n) \neq 0$ *then* $|f(\zeta_n)| > \exp(-|\operatorname{poly}(m, L, \log n)|)$*'* immediately leads to the straightforward polynomial algorithm for CT as well as for the inequality testing.

Another generalization of the problem involved is to represent a polynomial by an algebraic circuit (a straight-line program). This succinct representation makes the problem harder. The complexity status of this problem is also unknown.

Acknowledgments. We are grateful to Igor Shparlinski and Dima Grigoriev for the interest to the paper. We are thankful to the unknown referees for their detailed reviews which help to improve the text.

References

1. de Bruijn, N.G.: On the factorization of cyclic groups. Indag. Math. 15, 370–377 (1953)
2. Conway, J.H., Jones, A.J.: Trigonometric Diophantine equations (On vanishing sums of roots of unity). Acta Arith. 30, 229–240 (1976)
3. Filaseta, M., Granville, A., Schinzel, A.: Irreducibility and greatest common divisor algorithms for sparse polynomials, http://www.math.sc.edu//~filaseta/papers/SparsePaper.pdf
4. Filaseta, M., Schinzel, A.: On testing the divisibility of lacunary polynomials by cyclotomic polynomials. Math. Comp. 73, 957–965 (2004)
5. Lam, T.Y., Leung, K.H.: On vanishing sums of roots of unity. J. Algebra 224, 91–109 (2000)
6. Mignotte, M.: Identification of algebraic numbers. J. Algorithms 3, 197–204 (1982)
7. Plaisted, D.A.: New NP-hard and NP-complete polynomial and integer divisibility problems. Theoretical Computer Science 31, 125–138 (1984)
8. Rédei, L.: Ein Beitrag zum Problem der Faktorisation von Abelschen Gruppen. Acta Math. Acad. Sci. Hungar. 1, 197–207 (1950)

9. Rédei, L.: Natürliche Basen des Kreisteilungskörpers. Teil I. Abh. Math. Sem. Hamburg 23, 180–200 (1959)
10. Schoenberg, I.J.: A note on the cyclotomic polynomial. Mathematika 11, 131–136 (1964)
11. Steinberger, J.P.: Minimal vanishing sums of roots of unity with large coefficients, http://www.math.ucdavis.edu/~jpsteinb/vanishing.ps
12. Titova, E., Shevchenko, V.: Left and right kernels of a planar multiindex transportation problem. In: Proc. of VIII International seminar 'Discrete mathematics and applications'. MSU, Moscow, pp. 229–230 (In Russian) (2004)

Generic Complexity of Undecidable Problems

Alexei Myasnikov*

McGill University
Department of Mathematics and Statistics

Abstract. This is an extended abstract of my talk on generic complexity of undecidable problems. It turns out that some classical undecidable problems are, in fact, strongly undecidable, i.e., they are still undecidable on every strongly generic (i.e., "very very large") subset of inputs. For instance, the classical Halting Problem for Turing machines is strongly undecidable. Moreover, we prove an analog of the Rice's theorem for strongly undecidable problems, which provides plenty of examples of strongly undecidable problems. On the other hand, it has been shown recently that many of these classical undecidable problems are easily decidable on some generic (i.e., "very large") subsets of inputs. Altogether, these results lead to an interesting hierarchy of undecidable problems with respect to the size of subsets of inputs where the problems are still undecidable - a frequency analysis of hardness.

We construct here some natural super-undecidable problems, i.e., problem which are undecidable on every generic (not only strongly generic) subset of inputs. In particular, there are finitely presented semigroups with super-undecidable word problem. To construct strongly- and super-undecidable problems we introduce a method of generic amplification (an analog of the amplification in complexity theory).

1 Introduction

1.1 Motivation

In this paper we discuss algorithmic complexity of undecidable problems. We do not study them as in abstract recursion theory, say, trying to place them in the upper echelons of the hierarchy of Turing degrees. To the contrary, we approach them from a very practical computational view-point - we study their algorithmic behavior on "most" or "typical" inputs. This approach originated in asymptotic and computational algebra, and is called now "generic complexity" (see, [2,3,9,10]). Generic complexity allows one to study naturally the computational behavior of undecidable problems - impossible task in the worst-case or the average case complexities. In fact, the generic complexity provides a unifying framework for studying, and comparing, computational complexity of decidable and undecidable problems. It turned out, for example, that the famous halting

* The author was partially supported by NSF grant DMS-0405105, NSERC Discovery grant RGPIN 261898, and NSERC Canada Research Chair grant.

V. Diekert, M. Volkov, and A. Voronkov (Eds.): CSR 2007, LNCS 4649, pp. 407–417, 2007.

problem for Turing machines (with a one-way infinite tape) is easily decidable on most inputs - on the so-called "generic" sets of inputs [8]. This result generated a thorough study of several other undecidable problems, in particular, in groups [4] and semigroups [16]. Again, it has been shown that they are easy on some generic sets of inputs. Here we discuss undecidable problems that are still undecidable on arbitrary generic sets. Moreover, we give a method on how to amplify the undecidability of a given problem onto generic sets.

1.2 Generic Complexity

Let $\mathcal{D}$ be an algorithmic problem with a set of inputs I. We assume that the set I comes equipped with a "size" function $s : I \to \mathbb{N}$. Typically, the size $s(w)$ of an input $w \in I$ is the length of a description of w (that is fixed in advance). We also assume that there are only finitely many elements in I that have a given size $n \in \mathbb{N}$, so the spheres $I_n = \{w \in I \mid s(w) = n\}$, as well as the balls $B_n = \{w \in W \mid s(w) \leq n\}$, are finite. To distinguish "large" and "small" subsets of I one needs either a measure μ on I, or, better still, an ensemble $\mu = \{\mu_n\}_{n\in\mathbb{N}}$ of measures on the spheres I_n (or on the balls B_n). For a detailed discussion on generic complexity we refer to [7].

Fix an ensemble $\mu = \{\mu_n\}$ of spherical measures for I, if not said otherwise we assume that μ_n is the uniform distribution on the finite set I_n. Stratification $I = \cup_{n\in\mathbb{N}} I_n$ together with the ensemble μ allow one to measure size of subsets of I via asymptotic densities. For a subset $R \subseteq I$ the asymptotic (*spherical*) density $\rho_\mu(R)$ is defined by the following limit (if it exists)

$$\rho_\mu(R) = \lim_{n\to\infty} \mu_n(R \cap I_n)$$

Here

$$\mu_n(R \cup I_n) = \frac{|R \cap I_n|}{|I_n|}$$

is the n-th *frequency*, or probability, to hit an element from R in the sphere I_n. For uniform distributions we denote $\rho_\mu(R)$ simply by $\rho(R)$.

A subset $R \subseteq I$ is called *generic* if $\rho(R) = 1$ and *negligible* if $\rho(R) = 0$. Moreover, we say that R has asymptotic density $\rho(R)$ with a *super-polynomial convergence rate* if

$$|\rho(R) - \mu_n(R \cap I_n)| = o(n^{-k})$$

for any $k \in \mathbb{N}$. Now, a subset $R \subseteq I$ is called *strongly generic* if $\rho(R) = 1$ with the superpolynomial convergence rate. The set R is *strongly negligible* if its complement $I - R$ is strongly generic. Similarly, one can define exponential convergence rates and exponential generic (negligible) sets.

Now one can define generic complexity of algorithms. Let $\mathcal{A}$ be a partial decision algorithm for a problem $\mathcal{D}$. Denote by $T_\mathcal{A}$ the time function of the algorithm $\mathcal{A}$, so for an input $w \in I$ the value $T_\mathcal{A}(w)$ is the number of steps required for $\mathcal{A}$ to halt on w (if it is finite), or ∞ if $\mathcal{A}$ does not halt on w.

A function $f : \mathbb{N} \to \mathbb{R}$ is a *generic time upper bound* for $\mathcal{A}$ if the set

$$H_{\mathcal{A},f} = \{w \in I \mid T_{\mathcal{A}}(w) \leq f(s(w))\}$$

is generic in I with respect to the spherical asymptotic density ρ. Similarly, $f(n)$ is a *strongly generic* (*exponentially generic*, etc.) time upper bound for $\mathcal{A}$ if the set $H_{\mathcal{A},f}$ is strongly generic (has exponential convergence rate, etc.).

A partial decision algorithm $\mathcal{A}$ for $\mathcal{D}$ *generically solves* the problem $\mathcal{D}$ if the halting set $H_{\mathcal{A}}$ of $\mathcal{A}$ is generic in I with respect to the spherical asymptotic density ρ. If such a partial decision algorithm $\mathcal{A}$ exists we say that $\mathcal{D}$ is *generically decidable*.

As we have mentioned above the standard halting problem for Turing machines (with a one-way infinite tape), being undecidable, is nevertheless generically decidable. To study undecidable problems which are undecidable on "large" sets of inputs we need the following terminology.

We say that an undecidable problem $\mathcal{D}$ is *strongly undecidable* if it is undecidable on every strongly generic subset of inputs from I. More precisely, in this case for every strongly generic (with respect to ρ) subset $I' \subseteq I$ the restriction of $\mathcal{D}$ onto I' is still undecidable. Among strongly undecidable problems there are even "more undecidable" ones - we say $\mathcal{D}$ is *super-undecidable* if its undecidable on any generic subset of I. Furthermore, $\mathcal{D}$ is termed *absolutely undecidable* if it is undecidable on every non-negligible subset from I.

1.3 Overview of Results

In Section 2 we introduce the required terminology for the Halting Problem (HP) for Turing machines and indicate why HP is generically decidable when the tape is one-ended (for a complete proof see [8]).

Following [18], we show that the Halting Problem is, in fact, strongly undecidable. Then we prove a "strongly undecidable" analog of the Rice's theorem, which provides plenty of examples of strongly undecidable problems. We would like to emphasize here that these results do not depend on a particular model of Turing machines.

In Section 3 we discuss *generic amplification* - a method that allows one to construct a strongly (super-strongly) undecidable problem from a given undecidable problem, thus amplifying the hardness of the initial problem. The main tool here is termed "cloning". A *cloning* from a set I into a set J is a function $C : I \to P(J)$ from I into the set of all subsets $P(J)$ of J such that

$$\forall x, y \in I \, (x \neq y \to C(x) \cap C(y) = \emptyset) \, .$$

A *clone* $C(S)$ of a subset $S \subseteq I$ is just union of clones of members of S, so $C(S) = \cup_{x \in S} C(x)$. Similarly, if $\mathcal{D} = (L, I)$ is a decision problem in I then a decision problem $C(\mathcal{D}) = (C(L), J)$ in J is called the *clone* of $\mathcal{D}$ in J relative to C.

A cloning C from I to J is *effective* if there is an algorithm that for every $x \in I$ computes an effective enumeration of all elements in the clone $C(x)$,

and it is *non-negligible, (non-strongly-negligible)*, if for every $x \in I$ the clone $C(x)$ is non-negligible (non-strongly-negligible) in J. Theorem 4 is the main technical result of the paper. It states that if C is an effective non-strongly-negligible (non-negligible) cloning from I to J and $\mathcal{D} = (L, I)$ is undecidable problem in I then it is clone $C(\mathcal{D}) = (C(L), J)$ is strongly undecidable (super-undecidable) problem in J. In the same section we construct several effective non-negligible clonings thus providing plenty of examples of super-undecidable problems. Although these clonings are easy to construct, they often change the nature of the original decision problem, for example, the cloning of the word problem in a given group, or the decidability problem of a first-order theory, becomes a membership problem for some obscure language in a binary alphabet. In the rest of the paper we construct some particular clonings that preserve the nature of the original problem.

There are several famous finitely presented semigroups with undecidable word problem (see, for example, [12,17,20,13], or a survey [1]). However, it has been shown in [16] that their Word Problems have polynomial time generic case complexity. In Section 4 we show how one can amplify the generic complexity of finitely presented semigroups with undecidable word problem. Namely, for a finitely presented semigroup $\mathfrak{S}$ we construct a semigroup $\mathfrak{S}_x$ whose word problem is an effective non-negligible clone of the word problem of $\mathfrak{S}$. It follows that if the word problem in $\mathfrak{S}$ is undecidable then the word problem in $\mathfrak{S}_x$ is super-undecidable.

Tseitin semigroup $\mathfrak{T}$ has a presentation with 5 generators, 7 relations and undecidable word problem [20]. In this case the semigroup $\mathfrak{T}_x$ with super-undecidable word problem has 6 generators and 13 relators whose total length is equal to 49.

We would like to mention here that we do not know any examples of finitely presented groups with super-undecidable word problem.

2 Halting Problem for Turing Machines

The halting problem for Turing machines is one of the basic undecidable problems.

Recall that a Turing machine has a finite number n of states $Q_n = \{q_1, \ldots, q_n\}$, with q_1 designated as the *start* state, plus a separate designated state q_0, which is not in Q_n. We assume that all Turing machines with n states have the same set of states Q_n. A Turing machine *program* is a function

$$p : Q_n \times \{0, 1\} \to (Q_n \cup \{q_0\}) \times \{0, 1\} \times \{L, R\}.$$

The transition $p(q, i) = \langle r, j, R\rangle$, for example, directs that when the head is in state q reading symbol i, it should change to state r, write symbol j, and move one cell to the right. The computation of a program proceeds by iteratively performing the instructions of such transition rules, halting when the final state q_0 is reached.

If the machine operates on a one-way infinite tape and attempts to move left from the left-most cell, then the head falls off the tape and all computation ceases. In this case we say that the machine "breaks".

If a particular model of Turing machines is fixed (one-way infinite tape, two tapes, etc.) then we do not distinguish between Turing machines and Turing machine programs.

Definition 1. *The* Empty Tape Halting Problem *is the set H of programs p of Turing machines that halt or break when computing on a tape initially filled with 0s.*

Let P be a set of all Turing machine programs. By size of a program $p \in P$ we understand the number of states $s(p)$ in p. This gives a size function $s : P \to \mathbb{N}$. The sphere P_n consists of all programs with n states.

The most natural method for measuring the size of a set of Turing machine programs is that of asymptotic density ρ relative to the ensemble $\{\mu_n\}$ of the uniform distributions on spheres P_n. Thus, for a set $B \subseteq P$ of Turing machine programs

$$\rho(B) = \lim_{n\to\infty} \frac{|B \cap P_n|}{|P_n|},$$

if this limit exists. We define *generic* and *strongly generic* sets of programs relative to the asymptotic density ρ.

The following result was obtained in [8], it holds for the model of Turing machines with a one-way infinite tape. The proof is sensitive to this particular computational model. The question whether the result holds for arbitrary model of Turing machines is open.

Theorem 1 (Generic decidability of the Empty Tape Halting Problem). *There is a set B of Turing machine programs such that*

1) B is generic.
2) B is polynomial time decidable.
3) The restriction $H \cap B$ of the empty tape halting problem H on B is polynomial time decidable.

However, Rybalov recently showed in [18] that the classical Halting Problem is strongly undecidable, i.e., it is not decidable on any strongly generic subset. This result does not depend on a model of Turing machines. The proof is quite instructive and we sketch it here.

Let δ be an effective coding of all Turing machines by strings in the alphabet $\{1\}$, so given $\delta(M)$ one can recover M; conversely, given a machine M one can effectively compute $\delta(M)$. We also assume that for a string $x \in \{1\}^*$ one can effectively determine if $x = \delta(M)$ for some Turing machine M or not.

Definition 2. *(The classical Halting Problem) The Halting Problem is the set HP of programs p of Turing machines that halt on the input $\delta(p)$.*

By definition HP is decidable on a strongly generic subset $S \subseteq P$ if there is a partial computable function $f : \{1\}^* \to \{0,1\}$ such that $S \subseteq Dom(f)$ and if $f(\delta(M)) = 1$ then M halts on $\delta(M)$, and if $f(\delta(M)) = 0$ then M does not halt on $\delta(M)$, and also $f(x)$ is undefined if $x \neq \delta(M)$ for any Turing machine M. In this case, we say that f is a strongly generic decision function for HP. In particular, it follows that the domain $Dom(f)$ is a strongly generic recursively enumerable set on which HP is decidable.

Let M be a Turing machine with k non-final states $\{q_1, \ldots, q_k\}$. For a given $n \geq k$ one can construct a new machine M^* on n states with the following program:

$$2k \text{ fixed instructions of } M \begin{cases} (q_1, 0) \to \ldots, \\ \ldots \\ (q_k, 1) \to \ldots, \end{cases}$$

$$\text{arbitrary } 2(n-k) \text{ instructions} \begin{cases} (q_{k+1}, 0) \to \ldots, \\ \ldots \\ (q_n, 1) \to \ldots. \end{cases}$$

It is easy to see that M^* computes the same function as M does, because the new states are not attainable from the states of M, so M^* never executes any of the new instructions. Denote by $C(M)$ the set of all Turing machines M^* described above.

The following lemma is straightforward.

Lemma 1. *For any Turing machine M the set $C(M)$ is not strongly negligible.*

Theorem 2. *[18] The Halting Problem HP is strongly undecidable, i.e., it is undecidable on every strongly generic subset of programs.*

Proof. We follow here the classical proof of undecidability of the Halting Problem. Suppose, to the contrary, that there exists a strongly generic set S on which HP is decidable. Then there exists a partial decision function $f : \{1\}^* \to \{0,1\}$ for HP such that $S \subseteq Dom(f)$. We may assume that $S = Dom(f)$. It is easy to see then that the following function is partial computable

$$h(x) = \begin{cases} \text{undefined}, & \text{if } f(x) = 1, \text{ or } f(x) \text{ is undefined}, \\ 1, & \text{if } f(x) = 0. \end{cases}$$

Denote by M_h a Turing machine that computes h. Notice, that the set $P - S$ is strongly negligible, so by Lemma 1 the set $C(M_h)$ is not a subset of $P - S$, hence, there is a machine M_h^* from $S \cap C(M_h)$ computing h. Now let's look at the result of computation of M_h^* on the input $\delta(M_h^*)$. Observe, first, that $f(\delta(M_h^*))$ is defined since $M_h^* \in S$. If M_h^* halts on $\delta(M_h^*)$ then $f(\delta(M_h^*)) = 1$, hence $h(\delta(M_h^*))$ is undefined, so M_h^* does not halt on $\delta(M_h^*)$ - contradiction. If M_h^* does not halt on $\delta(M_h^*)$ then $f(\delta(M_h^*)) = 0$, so $h(\delta(M_h^*)) = 1$, which implies that M_h^* halts on $\delta(M_h^*)$ - contradiction. This shows that such S does not exists, as claimed.

It turned out that the classical Rice's theorem admits the following improvement.

Let $\mathcal{C}$ be a class of all partial computable functions of the type $f : \{1\}^* \to \{0,1\}$. A subclass $\mathcal{F}$ of $\mathcal{C}$ is proper if $\mathcal{F} \neq \emptyset$ and $\mathcal{F} \neq \mathcal{C}$.

Theorem 3. *[Myasnikov, Rybalov] Let $\mathcal{F}$ be a proper class of partial computable functions. Then the problem whether or not a given Turing machine computes a function from $\mathcal{F}$ is strongly undecidable.*

3 Generic Amplification of Undecidable Problems

In this section we discuss generic amplification - a method that allows one to construct a strongly (super-strongly) undecidable problem from a given undecidable problem, thus amplifying the hardness of the problem to an arbitrary strongly generic (generic) set. The main tool of the generic amplification method is "cloning", we described it below.

Let I and J be sets. A *cloning* of I in J is a function $C : I \to P(J)$ from I into the set of all subsets $P(J)$ of J such that

$$\forall x, y \in I \, (x \neq y \to C(x) \cap C(y) = \emptyset) \, .$$

For a subset $S \subseteq I$ its *clone* $C(S)$ is defined as union of clones of elements in S:

$$C(S) = \cup_{x \in S} C(x).$$

Similarly, if $\mathcal{D} = (L, I)$ is a decision problem in I then a decision problem $C(\mathcal{D}) = (C(L), J)$ in J is called the *clone* of $\mathcal{D}$ in J relative to C. Since $C(x) \cap C(y) = \emptyset$ for $x \neq y$ one has the following fundamental property of cloned problems:

$$x \in L \Longleftrightarrow C(x) \subseteq C(L) \Longleftrightarrow C(x) \cap C(L) \neq \emptyset \tag{1}$$

We say that a cloning C from I to J is *effective* if there is an algorithm $E(x, i)$ that for every $x \in I$ computes an effective enumeration of the clone $C(x)$, i.e.,

$$C(x) = \{E(x, 0), E(x, 1), \ldots, \}. \tag{2}$$

Among effective clonings one can consider *polynomial, super-polynomial, exponential* time clonings, or, more generally, effective clonings with a given time bound.

One can view an effective cloning C from I to J as a total computable function $E : I \times \mathbb{N} \to J$ such that $C(x) \cap C(y) = \emptyset$ for $x \neq y$ where the clone $C(x)$ of x is defined by (2).

To amplify a worst-case hardness of a given problem $\mathcal{D}$ in I into a generically hard distributional problem $C(\mathcal{D})$ in J one needs to have a notion of genericity of subsets of J. To this end, we assume that the set J is equipped with a fixed size function $s : J \to \mathbb{N}$ and generic subsets of J are defined with respect to the asymptotic density.

A cloning $C : I \to P(J)$ is called *non-negligible, (non-strongly-negligible, etc.)*, if for every $x \in I$ the clone $C(x)$ is non-negligible (non-strongly-negligible, etc.) in J.

Example 1. Let $I = J = \{0,1\}^*$. Define a function $E : I \times \mathbb{N} \to I$ by

$$E(a_1 \dots a_{n-1}a_n, i) = a_1 0 \dots a_{n-1} 0 a_n 1 bin(i),$$

where $a_k \in \{0,1\}$ and $bin(i)$ is the binary expression of the natural number i.

We claim that E gives rise to a polynomial time computable non-negligible cloning from I to I. Indeed, in this case, if $x = a_1 \dots a_{n-1}a_n$, then, according to [2],

$$C(x) = \{a_1 0 \dots a_{n-1} 0 a_n 1 bin(i) \mid i \in \mathbb{N}\}$$

so $C(x) \cap C(y) = \emptyset$ for $x \neq y$, hence C is cloning from I to I. Clearly, C is effective, moreover, E is polynomial time computable. Observe, that for the spherical uniform distribution μ_n on the sphere $I_n = \{w \in I \mid |w| = n\}$ one has for $n \geq 2|x| + 1$:

$$\mu_n(C(x) \cap I_n) = \frac{2^{n-2|x|-1}}{2^n} = \frac{1}{2^{2|x|+1}} > 0.$$

Therefore,

$$\rho(C(x)) = \frac{1}{2^{2|x|}} > 0$$

and $C(x)$ is non-negligible for any x.

The following result allows one to construct strongly and super-undecidable problems.

Theorem 4. *[Myasnikov, Rybalov] Let I, J be sets and $C : I \to P(J)$ an effective cloning. Then for every undecidable problem $\mathcal{D}$ with inputs from I the following holds:*

1) *if C is non-negligible then $C(\mathcal{D})$ is generically decidable in J then $\mathcal{D}$ is super-undecidable (undecidable on every generic subset of I).*
2) *if C is non-strongly-negligible then the clone $C(\mathcal{D})$ is strongly undecidable (undecidable on every strongly generic subset).*

Corollary 1. *Let $I = \{0,1\}^*$ and C the cloning from Example 1. Then for every undecidable problem $\mathcal{D} = (L, I)$ its clone problem $C(\mathcal{D}) = (C(L), I)$ is super-undecidable.*

4 Finitely Presented Semigroups with Super-Undecidable Word Problems

There are several famous finitely presented semigroups with undecidable word problem (see, for example, [12,20,13], or a survey [1]). However, it has been shown in [16] that their Word Problems have polynomial time generic case complexity. The main reason for this phenomenon is that to interpret a hard problem (say, the Halting Problem for Turing machines) one brings into the finite presentation of a semigroup a lot of "garbage" (extra generators and relators that serve to

simulate the Turing machine computation) that makes the Word problem easy on most inputs. In this section we show how one can amplify the generic complexity of finitely presented semigroups with undecidable word problem. Namely, we describe a general method to construct finitely presented semigroups with super-undecidable word problems. More precisely, for a finitely presented semigroup $\mathfrak{S}$ we construct a semigroup $\mathfrak{S}_x$ whose word problem is an effective non-negligible clone of the word problem of $\mathfrak{S}$.

Let

$$\mathfrak{S} = \langle a_1, \ldots, a_n | r_1 = s_1, \ldots, r_k = s_k \rangle$$

be a finitely presented semigroup with a set of generators $A = \{a_1, \ldots, a_n\}$ and a set of defining relations $R = \{r_1 = s_1, \ldots, r_k = s_k\}$.

Denote by $WP_{\mathfrak{S}}$ the word problem in the semigroup $\mathfrak{S}$, so

$$WP_{\mathfrak{S}} = \{(u, v) \in A^* \times A^* \mid u = v \text{ in } \mathfrak{S}\}.$$

For a letter $x \notin A$ put

$$\mathfrak{S}_x = \langle A, x | R, x = xa_1, \ldots, x = xa_n, x = xx \rangle.$$

Then $\mathfrak{S}_x$ is also a finitely presented semigroup. Denote $A_x = A \cup \{x\}$.

Lemma 2. *For any $w_1, w_2 \in A^*$ and $v_1, v_2 \in A_x^*$ the following hold:*

1) $w_1 = w_2$ in $\mathfrak{S} \Leftrightarrow w_1 = w_2$ in $\mathfrak{S}_x$,
2) $w_1 = w_2$ in $\mathfrak{S} \Leftrightarrow w_1 x v_1 = w_2 x v_2$ in $\mathfrak{S}_x$.

Corollary 2. *The canonical embedding $A \to A_x$ extends to an embedding of semigroups $\mathfrak{S} \to \mathfrak{S}_x$.*

Lemma 3. *Let $I = A^* \times A^*$ and $J = A_x^* \times A_x^*$. For $(u, v) \in I$ put*

$$C(u, v) = \{(uxp, vxq) \mid p, q \in A_x^*\}.$$

Then C is an effective non-negligible cloning.

Lemma 4. *For any finitely presented semigroup $\mathfrak{S}$ the word problem in $\mathfrak{S}_x$ is the C-clone of the word problem in $\mathfrak{S}$:*

$$WP_{\mathfrak{S}_x} = C(WP_{\mathfrak{S}}).$$

Proof. It follows immediately from Lemma 2.

Theorem 5 (Myasnikov, Rybalov). *If the word problem in $\mathfrak{S}$ is undecidable then the word problem in $\mathfrak{S}_x$ is super-undecidable.*

Proof. By Lemma 3 C is an effective non-negligible cloning. By Lemma 4 $WP_{\mathfrak{S}_x} = C(WP_{\mathfrak{S}})$. Now by Theorem 4 if the word problem $WP_{\mathfrak{S}}$ is undecidable then the word problem $WP_{\mathfrak{S}_x} = C(WP_{\mathfrak{S}})$ is super-undecidable, as claimed.

5 Tseitin Semigroup

In 1956, Tseitin constructed a semigroup T presented by 5 generators and 7 relations with undecidable word problem.

Theorem 6 (Tseitin). *Let T be a semigroup presented by the generators a, b, c, d, e and defining relations*

$$ac = ca, ad = da, bc = cb, bd = db, ce = eca, de = edb, cca = ccae.$$

Then the Word Problem in T is undecidable.

However, it was shown in [16] that the Word Problem in T is generically easy.

Proposition 1. *The Word Problem in Tseitin T semigroup is decidable in linear time on a generic set of inputs.*

By Theorem 5 the semigroup $\mathfrak{T}_x$ from Section 4 has super-undecidable . Observe, that $\mathfrak{T}_x$ has 6 generators and 13 relators whose total length is equal to 49.

Problem 1. Is there a finitely presented group with generically undecidable word problem?

References

1. Adjan, S.I., Durnev, V.G.: Decision problems for groups and semigroups. Russian Math. Surveys 55(2), 207–296 (2000)
2. Borovik, A., Myasnikov, A., Shpilrain, V.: Measuring sets in infinite groups. Computational and Statistical Group Theory. Amer. Math. Soc. Contemporary Math. 298, 21–42 (2002)
3. Borovik, A.V., Myasnikov, A.G., Remeslennikov, V.N.: Multiplicative measures on free groups. Internat. J. Algebra Comput. 13(6), 705–731 (2003)
4. Borovik, A.V., Myasnikov, A.G., Remeslennikov, V.N.: Algorithmic stratification of the conjugacy problem in Millers groups. International Journal of Algebra and Computation (to appear)
5. Borovik, A.V., Myasnikov, A.G., Remeslennikov, V.N.: The conjugacy problem in amalgamated products I: regular elements and black holes
6. Cooper, S.B.: Computability Theory. Chapman and Hall/CRC Mathematics (2003)
7. Gilman, R., Miasnikov, A.D., Myasnikov, A.G., Ushakov, A.: Generic Complexity. (preprint)
8. Hamkins, J.D., Miasnikov, A.: The halting problem is decidable on a set of asymptotic probability one. Notre Dame Journal of Formal Logic 47(4), 515–524 (2006)
9. Kapovich, I., Myasnikov, A., Schupp, P., Shpilrain, V.: Generic-case complexity and decision problems in group theory. J. of Algebra 264, 665–694 (2003)
10. Kapovich, I., Myasnikov, A., Schupp, P., Shpilrain, V.: Average-case complexity for the word and membership problems in group theory. Advances in Mathematics 190(2), 343–359 (2005)
11. Knuth, D.E.: The art of computer programming: Sorting and Searching, vol. 3. Addison-Wesley, Reading (1998)

12. Markov, A.A.: On the impossibility of certain algorithms in the theory of associative systems, Dokl. Akad. Nauk SSSR 55, 587–590 (1947) (French transl., C.R. (Dokl.) Acad. Sci. URSS II, 55, 583–586 (1947))
13. Yu, V.: Matiyasevich, Simple examples of undecidable associative calculi, Dokl. Akad. Nauk SSSR 173, 1264–1266 (1967) (English transl., Soviet Math. Dokl. 8, 555–557 (1967))
14. Mendelson, E.: Introduction to Mathematical Logic. Chapman and Hall/CRC (1997)
15. Myasnikov, A., Ushakov, A.: Random van Kampen Diagrams and algorithmic problems in groups.
16. Miasnikov, A., Ushakov, A., Won, D.W.: Generic complexity of the word problem in finitely presented semigroups 2006 (preprint)
17. Post, E.L.: Recursive unsolvability of a problem of Thue. J.Symbolic Logic 12(1), 1–11 (1947)
18. Rybalov, A.: On the Strongly Generic Undecidability of the Halting Problem. In: Theoretical Computer Science 2007 (to appear)
19. Savage, J.E.: The Complexity of Computing. John Wiley and Sons Inc., Chichester (1977)
20. Tseitin, G.S.: An associative system with undecidable equivalence problem. MIAN 52, 172–189 (1958)

Author Index