Mario Coppo
Elena Lodi
G. Michele Pinna (E

LNCS 3701

Theoretic
Compute

9th Italian Conference, ICTC
Siena, Italy, October 2005
Proceedings

Mario Coppo Elena Lodi
G. Michele Pinna (Eds.)

Theoretical Computer Science

9th Italian Conference, ICTCS 2005
Siena, Italy, October 12-14, 2005
Proceedings

Springer

Volume Editors

Mario Coppo
Università di Torino, Dipartimento di Informatica
C. Svizzera 185, 10149 Torino, Italy
E-mail: coppo@di.unito.it

Elena Lodi
Università di Siena
Dipartimento di Scienze Matematiche e Informatiche
Pian dei Mantellini 44, 53100 Siena, Italy
E-mail: lodi@unisi.it

G. Michele Pinna
Università di Siena
Dipartimento di Scienze Matematiche e Informatiche
Pian dei Mantellini 44, 53100 Siena, Italy
and Università di Cagliari
Dipartimento di Informatica e Matematica
Via Ospedale 72, 09124 Cagliari, Italy
E-mail: gmpinna@unica.it

Library of Congress Control Number: 2005933156

CR Subject Classification (1998): F, E.1, G.1-2

ISSN	0302-9743
ISBN-10	3-540-29106-7 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-29106-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11560586 06/3142 5 4 3 2 1 0

Preface

The 9th Italian Conference on Theoretical Computer Science (ICTCS 2005) was held at the Certosa di Pontignano, Siena, Italy, on October 12–14 2005. The Certosa di Pontignano is the conference center of the University of Siena; it is located 8 km away from the town and it is in the Chianti region. The Certosa is a place full of history (founded in the 15th century, it was set on fire a century later and reconstructed) and of valuable artworks, like frescoes of the Scuola Senese.

Previous conferences took place in Pisa (1972), Mantova (1974 and 1989), L'Aquila (1992), Ravello (1995), Prato (1998), Turin (2001) and Bertinoro (2003).

The conference aims at bringing together computer scientists, especially young researchers, to foster cooperation, exchange of ideas and results. Great efforts have been made to attract researchers from all over the world. The main topics of the conference cover all the fields of theoretical computer science and include analysis and design of algorithms, computability, computational complexity, cryptography, formal languages and automata, foundations of programming languages and program analysis, foundations of artificial intelligence and knowledge representation, foundations of web programming, natural computing paradigms (quantum computing, bioinformatics), parallel and distributed computation, program specification and verification, term rewriting, theory of concurrency, theory of data bases, theory of logical design and layout, type theory, security, and symbolic and algebraic computation.

The Program Committee, consisting of 16 members, considered 83 papers and selected 29 for presentation. These papers were selected, after a careful reviewing process, on the basis of their originality, quality and relevance to theoretical computer science. In keeping with the previous conferences, ICTCS 2005 was characterized by the high number of submissions and by the number of different countries represented. These proceedings contain the revised versions of the 29 accepted papers together with the invited talks by Luca Cardelli (Biological Systems as Reactive Systems, abstract), Giuseppe Castagna (Semantic Subtyping: Challenges, Perspectives, and Open Problems) and Nicola Santoro (Mobile Agents Computing: Security Issues and Algorithmic Solutions).

Due to the high quality of the submissions, paper selection was a difficult and challenging task. Each submission was reviewed by at least three reviewers. We thank all the Program Committee members and the additional reviewers for their accurate work and for spending so much time in the reviewing process. We apologize for any inadvertent omission in the list of reviewers.

Following the example of the last ICTCS edition, we encouraged authors to submit their papers in electronic format. Special thanks are due to Simone Donetti for setting up a very friendly Web site for this purpose.

Finally, we would like to thank all the authors who submitted papers and all the conference participants.

October 2005

Mario Coppo
Elena Lodi
G. Michele Pinna

ICTCS 2005
October 12–14, Certosa di Pontignano, Siena, Italy

Program Co-chairs

Mario Coppo	Università di Torino
Elena Lodi	Università di Siena

Program Committee

Michele Bugliesi	Università di Venezia
Mario Coppo	Università di Torino (Co-chair)
Pierluigi Crescenzi	Università di Firenze
Giulia Galbiati	Università di Pavia
Luisa Gargano	Università di Salerno
Giorgio Ghelli	Università di Pisa
Roberto Grossi	Università di Pisa
Benedetto Intrigila	Università di L'Aquila
Nicola Leone	Università di Cosenza
Elena Lodi	Università di Siena, (Co-chair)
Flaminia Luccio	Università di Trieste
Andrea Masini	Università di Verona
Giancarlo Mauri	Università di Milano-Bicocca
Corrado Priami	Università di Trento
Geppino Pucci	Università di Padova
Davide Sangiorgi	Università di Bologna

Organizing Committee

G. Michele Pinna	Università di Cagliari (Chair)
Sara Brunetti	Università di Siena
Elisa Tiezzi	Università di Siena

Sponsoring Institutions

European Association for Theoretical Computer Science (EATCS)
Dipartimento di Matematica e Informatica "R. Magari", Università di Siena
Dipartimento di Informatica, Università di Torino
Università degli Studi di Siena
Monte dei Paschi di Siena

Referees

Tetsuo Asano
Vincenzo Auletta
Benjamin Aziz
Paolo Baldan
Anindya Banerjee
Stefano Baratella
Franco Barbanera
Massimo Bartoletti
Marie-Pierre Béal
Stefano Berardi
Marco Bernardo
Alberto Bertoni
Daniela Besozzi
Chiara Bodei
Paolo Boldi
Hanifa Boucheneb
Linda Brodo
Sara Brunetti
Francesco Buccafurri
Pasquale Caianiello
Tiziana Calamoneri
Felice Cardone
Dario Catalano
Marco Cesati
Federica Ciocchetta
Valentina Ciriani
Antonio Cisternino
Dario Colazzo
Andrea Clementi
Carlo Combi
Paolo D'Arco
Ottavio D'Antona
Christophe Decanniere
Pierpaolo Degano
Giuseppe Della Penna
Gianluca Della Vedova
Gianluca De Marco
Maria Rita Di Berardini
Pietro Di Giannantonio
Susanna Donatelli
Riccardo Dondi
Claudio Eccher
Wolfgang Faber
Riccardo Focardi
Paola Flocchini
Andrea Frosini
Clemente Galdi
Dora Giammarresi
Paola Giannini
Laura Giordano
Gianluigi Greco
Massimiliano Goldwurm
Stefano Guerrini
Giovambattista Ianni
Amos Korman
Ruggero Lanotte
Paola Lecca
Alberto Leporati
Renato Locigno
Fabrizio Luccio
Veli Mäkinen
Alessio Malizia
Stefano Mancini
Alberto Marchetti Spaccamela
Radu Mardare
Luciano Margara
Ines Margaria
Carlo Mereghetti
Alberto Martelli
Donatella Merlini
Filippo Mignosi
Alberto Momigliano
Manuela Montangero
Elisa Mori
Ian Munro
Aniello Murano
Venkatesh Mysore
Matthias Neubauer
Monica Nesi
Mitsunori Ogihara
Linda Pagli
Beatrice Palano
Luigi Palopoli
David Peleg
Paolo Penna
Simona Perri
Carla Piazza
Adolfo Piperno
Nadia Pisanti
Katerina Pokozy
Roberto Posenato
Davide Prandi
Giuseppe Prencipe
Orazio Puglisi
Paola Quaglia
Sven Rahmann
Adele Rescigno
Antonio Restivo
Simone Rinaldi
Lorenzo Robbiano
Simona Ronchi della Rocca
Gianluca Rossi
Alehandro Russo
Giancarlo Ruffo
Ivano Salvo
Francesco Scarcello
Debora Schuch
Roberto Segala
Andrea Sgarro
Riccardo Silvestri
Maria Simi
Robert Spalek
Renzo Sprugnoli
Frank Christian Stephan
Giorgio Terracina
Elisa Tiezzi
Mauro Torelli
Andrea Torsello
Emilio Tuosto
Ugo Vaccaro
Leonardo Vanneschi
Stefano Varricchio
Maria Cecilia Verri
Bob Walters
Damiano Zanardini
Claudio Zandron
Nicola Zannone

Table of Contents

Semantic Subtyping: Challenges, Perspectives, and Open Problems

Giuseppe Castagna

CNRS, École Normale Supérieure de Paris, France

Based on joint work with: Véronique Benzaken, Rocco De Nicola, Mariangiola Dezani, Alain Frisch, Haruo Hosoya, Daniele Varacca

Abstract. Semantic subtyping is a relatively new approach to define subtyping relations where types are interpreted as sets and union, intersection and negation types have the corresponding set-theoretic interpretation. In this lecture we outline the approach, give an aperçu of its expressiveness and generality by applying it to the λ-calculus with recursive and product types and to the π-calculus. We then discuss in detail the new challenges and research perspectives that the approach brings forth.

1 Introduction to the Semantic Subtyping

Many recent type systems rely on a subtyping relation. Its definition generally depends on the type algebra, and on its intended use. We can distinguish two main approaches for defining subtyping: the *syntactic* approach and the *semantic* one. The syntactic approach—by far the more used—consists in defining the subtyping relation by axiomatising it in a formal system (a set of inductive or coinductive rules); in the semantic approach (for instance, [AW93, Dam94]), instead, one starts with a model of the language and an interpretation of types as subsets of the model, then defines the subtyping relation as the inclusion of denoted sets, and, finally, when the relation is decidable, derives a subtyping algorithm from the semantic definition.

The semantic approach has several advantages (see [CF05] for an overview) but it is also more constraining. Finding an interpretation in which types can be interpreted as subsets of a model may be a hard task. A solution to this problem was given by Haruo Hosoya and Benjamin Pierce [HP01, Hos01, HP03], who noticed that in order to define subtyping all is needed is a set theoretic interpretation of types, not a model of the terms. In particular, they propose to interpret a type as the set of all values that have that type. So if we use $\mathcal{V}$ to denote the set of all values, then we can define the following set-theoretic interpretation for types $[\![t]\!]_{\mathcal{V}} = \{v \in \mathcal{V} \mid\ \vdash v : t\}$ which induces the following subtyping relation:

$$s \leq_{\mathcal{V}} t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathcal{V}} \subseteq [\![t]\!]_{\mathcal{V}} \tag{1}$$

This works for Hosoya and Pierce because the set of values they consider can be defined independently from the typing relation.[1] But in general in order to state when a value has a given type (the "$\vdash v : t$" in the previous definition) one needs the subtyping

[1] Their values are XML documents, and they can be defined as regular trees. The typing relation, then, becomes recognition of a regular tree language.

relation. This yields a circularity: we are building a model to define the subtyping relation, and the definition of this model needs the subtyping relation. This circularity is patent in both the examples we discuss below: in λ-calculus (Section 2) values are λ-abstractions and to type them (in particular, to type applications that may occur in their body) subtyping is needed; in π-calculus (Section 3) the covariance and contravariance of read-only and write-only channel types make the subtyping relation necessary to type channels.

In order to avoid this circularity and still interpret types as set of values, we resort to a bootstrapping technique. The general ideas of this technique are informally exposed in [CF05], while the technical development can be found in [FCB02, Fri04]. For the aims of this article, the process of defining semantic subtyping can be roughly summarised in the following steps:

1. Take a bunch of type *constructors* (e.g., $\rightarrow$, $\times$, $ch()$, ...) and extend the type algebra with the following *boolean combinators*: union $\vee$, intersection $\wedge$, and negation $\neg$.
2. Give a *set-theoretic model* of the type algebra, namely define a function $[\![\,]\!]_{\mathcal{D}}$: **Types** $\rightarrow \mathcal{P}(\mathcal{D})$, for some domain $\mathcal{D}$ (where $\mathcal{P}(\mathcal{D})$ denotes the powerset of $\mathcal{D}$). In such a model, the combinators must be interpreted in a set-theoretic way (that is, $[\![s\wedge t]\!]_{\mathcal{D}} = [\![s]\!]_{\mathcal{D}} \cap [\![t]\!]_{\mathcal{D}}$, $[\![s\vee t]\!]_{\mathcal{D}} = [\![s]\!]_{\mathcal{D}} \cup [\![t]\!]_{\mathcal{D}}$, and $[\![\neg t]\!]_{\mathcal{D}} = \mathcal{D} \setminus [\![t]\!]_{\mathcal{D}}$), and the definition of the model must capture the essence of the type constructors.

   There might be several models, and each of them induces a specific subtyping relation on the type algebra. We only need to prove that there exists at least one model and then pick one that we call the *bootstrap model*. If its associated interpretation function is $[\![\,]\!]_{\mathcal{B}}$, then it induces the following subtyping relation:

$$s \leq_{\mathcal{B}} t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathcal{B}} \subseteq [\![t]\!]_{\mathcal{B}} \tag{2}$$

3. Now that we defined a subtyping relation for our types, find a subtyping algorithm that decides (or semi-decides) the relation. This step is not mandatory but highly advisable if we want to use our types in practise.
4. Now that we have a (hopefully) suitable subtyping relation available, we can focus on the language itself, consider its typing rules, use the new subtyping relation to type the terms of the language, and deduce $\Gamma \vdash_{\mathcal{B}} e : t$. In particular this means to use in the subsumptionrule the bootstrap subtyping relation $\leq_{\mathcal{B}}$ we defined in step 2.
5. The typing judgement for the language now allows us to define a *new* natural set-theoretic interpretation of types, the one based on values $[\![t]\!]_{\mathcal{V}} = \{v \in \mathcal{V} \mid \vdash_{\mathcal{B}} v : t\}$, and then define a "new" subtyping relation as in equation (1). This relation might be different from $\leq_{\mathcal{B}}$ we started from. However, if the definitions of the model, of the language, and of the typing rules have been carefully chosen, then the two subtyping relations coincide

$$s \leq_{\mathcal{B}} t \quad \Longleftrightarrow \quad s \leq_{\mathcal{V}} t$$

   and this closes the circularity. Then, the rest of the story is standard (reduction relation, subject reduction, type-checking algorithm, etc ...).

The accomplishment of this is process is far from being straightforward. In point 2 it may be quite difficult to capture the semantics of the type constructors (e.g., it is quite hard to define a set-theoretic semantics for arrow types); in point 3 defining a model may go from tricky to impossible (e.g., because of bizarre interactions with recursive types); point 4 may fail for the inability of devising a subtyping algorithm (cf. the subtyping algorithm for $\mathbb{C}\pi$ in [CNV05]); finally the last step is the most critical one since it may require a consistent rewriting of the language and/or of the typing rules to "close the circle" ... if possible at all. We will give examples of all these problems in the rest of this document.

In the next two sections we are going to show how to apply this process to λ-like and π-like calculi. The presentation will be sketchy and presuppose from the reader some knowledge of the λ-calculus, of the π-calculus, and of their type systems. Also, the calculi we are going to present are very simplified versions of the actual ones whose detailed descriptions can be found in [FCB02] and [CNV05], respectively.

2 Semantic λ-Calculus: $\mathbb{C}$Duce

As a first example of application of the semantic subtyping 5-steps technique, let us take the λ-calculus with products.

Step 1. The first step consists in taking some type constructors, in this case products and arrows, and adding boolean combinators to them:

$$t ::= \mathbb{0} \mid \mathbb{1} \mid t \to t \mid t \times t \mid \neg t \mid t \vee t \mid t \wedge t$$

where $\mathbb{0}$ and $\mathbb{1}$ correspond, respectively, to the empty and the universal types. For more generality we consider also recursive types. Thus, our types are the regular trees generated by the grammar above and satisfying the standard contractivity condition that every infinite branch has infinitely many occurrences of the $\times$ or of the $\to$ constructors (this rules out meaningless expressions such as $t \wedge (t \wedge (t \wedge (\dots))))$.

Step 2. The second step is, in this case, the hard one as it requires to define a set-theoretic interpretation $[\![\,]\!]_{\mathcal{D}} : \mathbf{Types} \to \mathcal{P}(\mathcal{D})$. But, how can we give a set theoretic interpretation to the arrow type? The set theoretic intuition we have of $t \to s$ is that it is the set of all functions (of our language) that when applied to a value of type t either diverge or return a result of type s. If we interpret functions as binary relations on $\mathcal{D}$, then $[\![t \to s]\!]$ is the set of binary relations in which if the first projection is in (the interpretation of) t then the second projection is in (the interpretation of) s, namely $\mathcal{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$, where the overline denotes set complement.[2] However, setting $[\![t \to s]\!] = \mathcal{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ is impossible since, for cardinality reasons, we cannot have $\mathcal{P}(\mathcal{D}^2) \subseteq \mathcal{D}$. Note though, that we do not define the interpretation $[\![\,]\!]$ in order to formally state what the syntactic types *mean* but, more simply, we define it in order to state how they are *related*. Therefore, even if the interpretation does not capture the intended semantics of types, all we need is that it captures the containment relation induced by this semantics. That is, roughly, it suffices to our aims that the interpretation function satisfies

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathcal{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathcal{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}}) \tag{3}$$

[2] This is just one of the possible interpretations. See [CF05] for a discussion about the implications of such a choice and [Fri04] for examples of different interpretations.

Note that $\mathcal{P}_f(X) \subseteq \mathcal{P}_f(Y)$ if and only if $\mathcal{P}(X) \subseteq \overline{\mathcal{P}(Y)}$ (where $\mathcal{P}_f$ denotes the finite powerset). Therefore, if we set $[\![t \rightarrow s]\!] = \mathcal{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$, this interpretation satisfies (3). In other words, we can use as bootstrap model $\mathcal{B}$ the least solution of the equation $X = X^2 + \mathcal{P}_f(X^2)$ and the following interpretation function[3] $[\![\ ]\!]_{\mathcal{B}}$: **Types** $\rightarrow \mathcal{P}(\mathcal{B})$:

$$[\![\mathbb{0}]\!]_{\mathcal{B}} = \varnothing \quad [\![\mathbb{1}]\!]_{\mathcal{B}} = \mathcal{B} \qquad [\![s \vee t]\!]_{\mathcal{B}} = [\![s]\!]_{\mathcal{B}} \cup [\![t]\!]_{\mathcal{B}} \qquad [\![s \wedge t]\!]_{\mathcal{B}} = [\![s]\!]_{\mathcal{B}} \cap [\![t]\!]_{\mathcal{B}}$$
$$[\![\neg t]\!]_{\mathcal{B}} = \mathcal{B} \backslash [\![t]\!]_{\mathcal{B}} \qquad [\![s \times t]\!]_{\mathcal{B}} = [\![s]\!] \times [\![t]\!] \qquad [\![t \rightarrow s]\!]_{\mathcal{B}} = \mathcal{P}_f(\overline{[\![t]\!]_{\mathcal{B}} \times \overline{[\![s]\!]_{\mathcal{B}}}})$$

The model we have chosen can represent only finite graph functions, therefore it is not rich enough to give semantics to a λ-calculus (even the simply typed one). However since this model satisfies equation (3), it is able to express the containment relation induced by the semantic intuition we have of the type $t \rightarrow s$ (namely that it represents $\mathcal{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$, which is all we need.

Step 3. We can use the definition of subtyping as given by equation (2) to deduce some interesting relations: for instance, according to (2) the type $(t_1 \rightarrow s_1) \wedge (t_2 \rightarrow s_2)$ is a subtype of $(t_1 \wedge t_2) \rightarrow (s_1 \wedge s_2)$, of $(t_1 \vee t_2) \rightarrow (s_1 \vee s_2)$, of their intersection and, in general, all these inclusions are strict.

Apart from these examples, the point of course is to devise an algorithm to decide inclusion between any pair of types. Deciding subtyping for arbitrary types is equivalent to decide whether a type is equivalent to (that is, it has the same interpretation as) $\mathbb{0}$:

$$s \leq_{\mathcal{B}} t \Leftrightarrow [\![s]\!]_{\mathcal{B}} \subseteq [\![t]\!]_{\mathcal{B}} \Leftrightarrow [\![s]\!]_{\mathcal{B}} \cap \overline{[\![t]\!]_{\mathcal{B}}} = \varnothing \Leftrightarrow [\![s \wedge \neg t]\!]_{\mathcal{B}} = \varnothing \Leftrightarrow s \wedge \neg t = \mathbb{0}.$$

By using the definition of $\mathcal{B}[\![\,]\!]$, we can show that every type is equivalent to a finite union where each summand is either of the form:

$$\Big(\bigwedge_{s \times t \in P} s \times t\Big) \wedge \Big(\bigwedge_{s \times t \in N} \neg(s \times t)\Big) \tag{4}$$

or of the form

$$\Big(\bigwedge_{s \rightarrow t \in P} s \rightarrow t\Big) \wedge \Big(\bigwedge_{s \rightarrow t \in N} \neg(s \rightarrow t)\Big) \tag{5}$$

Put $s \wedge \neg t$ in this form. Since it is a finite union, then it is equivalent to $\mathbb{0}$ if and only if each summand is so. So the decision of $s \leq_{\mathcal{B}} t$ is reduced to the problem of deciding whether the types in (4) and (5) are empty. The subtyping algorithm, then, has to work coinductively, decomposing these problems into simpler subproblems where the topmost type constructors have disappeared. In particular, in [Fri04] it is proved that the type in (4) is equivalent to $\mathbb{0}$ if and only if for every $N' \subseteq N$:

$$\left(\bigwedge_{(t \times s) \in P} t \ \wedge \bigwedge_{(t' \times s') \in N'} \neg t'\right) \simeq \mathbb{0} \ \text{ or } \ \left(\bigwedge_{(t \times s) \in P} s \ \wedge \bigwedge_{(t' \times s') \in N \backslash N'} \neg s'\right) \simeq \mathbb{0}; \tag{6}$$

while the type in (5) is equal to zero if and only if there exists some $(t' \rightarrow s') \in N$ such that for every $P' \subseteq P$:

$$\left(t' \wedge \bigwedge_{(t \rightarrow s) \in P'} \neg t\right) \simeq \mathbb{0} \ \text{ or } \ \left(\bigwedge_{(t \rightarrow s) \in P \backslash P'} s \wedge \neg s'\right) \simeq \mathbb{0}. \tag{7}$$

[3] For the details of the definition of the interpretation in the presence of recursive types, the reader is invited to consult [Fri04] and [FCB02]. The construction is also outlined in [CF05].

By applying these decompositions the algorithm can decide the subtyping relation. Its termination is ensured by the regularity and contractivity of the types.

Step 4. We have just defined a decidable subtyping relation for our types. We now want to apply it to type the terms of a language. We do not present here a complete language: the reader can find plenty of details in [CF05, FCB02]. Instead, we concentrate on the definition and the typing of the terms that are the most interesting for the development of this article, namely λ-abstractions. These in the rest of this paper will have the form $\lambda^{\wedge_{i\in I}s_i\to t_i}x.e$, that is we index them by an intersection type. This index instructs the type checker to verify that the abstraction is in the given intersection, namely, that it has all the types composing it, as implemented by the following rule:

$$\frac{\overset{t\equiv(\bigwedge_{i=1..n}s_i\to t_i)\wedge(\bigwedge_{j=1..m}\neg(s'_j\to t'_j))\neq\mathbb{0}}{(\forall i)\ \ \Gamma,x:s_i\vdash_{\mathscr{B}} e:t_i}}{\Gamma\vdash_{\mathscr{B}}\lambda^{\wedge_{i\in I}s_i\to t_i}x.e:t}\ \text{(abstr)}$$

To understand this rule consider, as a first approximation, the case for $m = 0$, that is, when the type t assigned to the function is exactly the index. The rule verifies that the function has indeed all the $s_i \to t_i$ types: for every type $s_i \to t_i$ of the intersection it checks that the body e has type t_i under the assumption that the parameter x has type s_i. The rule actually is (and must be) more general since it allows the type checker to infer for the function a type t strictly smaller than the one at the index, since the rule states that it is possible to subtract from the index any finite number of arrow types, provided that t remains non-empty.[4] This is necessary to step 5 of our process. But before moving to the next step, note that the intersection of arrows can be used to type overloaded functions. Indeed, our framework is compatible with overloading, since the following containment

$$[\![(t_1\vee t_2)\to(s_1\wedge s_2)]\!]\subsetneq[\![(t_1\to s_1)\wedge(t_2\to s_2)]\!] \tag{8}$$

is strict. So the semantic model authorises the language to define functions that return different results (e.g. one in $s_1\backslash s_2$ and the other in $s_2\backslash s_1$) according to whether their argument is of type t_1 or t_2. If the model had instead induced an equality between the two types above then the function could not have different behaviours for different types but should uniformly behave on them.[5]

Step 5. The last step consists in verifying whether the model of values induces that same subtyping relation as the bootstrap one. This holds only if

$$\vdash_{\mathscr{B}} v:t \quad\Longleftrightarrow\quad \not\vdash_{\mathscr{B}} v:\neg t \tag{9}$$

which holds true (and, together with the fact that no value has the empty type, makes the two subtyping relations coincide) thanks to the fact that the (abstr) rule deduces negated arrow types for lambda abstractions. Without it the difference of two arrow types (which

[4] Equivalently, it states that we can deduce for a λ-abstraction every non-empty type t obtained by intersecting the type indexing the abstraction with a finite number of negated arrow types that do not already contain the index.

[5] Overloading requires the addition of a type-case in the language. Without it intersection of arrows can just be used to give more specific behaviour, as for $\lambda^{Odd\to Odd\wedge Even\to Even}x.x$ which is more precise than $\lambda^{Int\to Int}x.x$.

in general is non-empty) might be not inhabited by a value, since the only way to deduce for an abstraction a negated arrow would be the subsumption rule. To put it otherwise, without the negated arrows in (abstr) property (9) would fail since, for instance, both $\not\vdash \lambda^{Int\to Int}x.(x+3) : \neg(Bool \to Bool)$, and $\not\vdash \lambda^{Int\to Int}x.(x+3) : Bool \to Bool$ would hold.

3 Semantic π-Calculus: $\mathbb{C}\pi$

In this section we repeat the 5 steps process for the π-calculus.

Step 1. The types we are going to consider are the following ones

$$t \quad ::= \quad \mathbb{0} \mid \mathbb{1} \mid ch^{+}(t) \mid ch^{-}(t) \mid ch(t) \mid \neg t \mid t \vee t \mid t \wedge t$$

without any recursion. As customary $ch^{+}(t)$ is the type of channels on which one can expect to receive values of type t, $ch^{-}(t)$ is the type of channels on which one is allowed to send values of type t, while $ch(t)$ is the type of channels on which one can send and expect to receive values of type t.

Step 2. The set-theoretic intuition of the above types is that they denote sets of channels. In turn a channel can be seen as a box that is tightly associated to the type of the objects it can transport. So $ch(t)$ will be the set of all boxes for objects of type t, $ch^{-}(t)$ the set of all boxes in which one can put something of type t while $ch^{+}(t)$ will be the set of boxes in which one expects to find something of type t. This yields the following interpretation

$$\begin{aligned}
[\![ch(t)]\!] &= \{c \mid c \text{ is a box for objects in } [\![t]\!]\} \\
[\![ch^{+}(t)]\!] &= \{c \mid c \text{ is a box for objects in } [\![s]\!] \text{ with } s \leq t\} \\
[\![ch^{-}(t)]\!] &= \{c \mid c \text{ is a box for objects in } [\![t]\!] \text{ with } s \geq t\}.
\end{aligned}$$

Given the above semantic interpretation, from the viewpoint of types all the boxes of one given type t are indistinguishable, because either they all belong to the interpretation of one type or they all do not. This implies that the subtyping relation is insensitive to the actual number of boxes of a given type. We can therefore assume that for every equivalence class of types, there is only one such box, which may as well be identified with $[\![t]\!]$, so that the intended semantics of channel types will be

$$[\![ch^{+}(t)]\!] = \left\{[\![s]\!] \mid s \leq t\right\} \qquad\qquad [\![ch^{-}(t)]\!] = \left\{[\![s]\!] \mid s \geq t\right\} \tag{10}$$

while the invariant channel type $ch(t)$ will be interpreted as the singleton $\{[\![t]\!]\}$. Of course, there is a circularity in the definitions in (10), since the subtyping relation is not defined, yet. So we rather use the following interpretations $[\![ch^{+}(t)]\!] = \{[\![s]\!] \mid [\![s]\!] \subseteq [\![t]\!]\}$, $[\![ch^{-}(t)]\!] = \{[\![s]\!] \mid [\![s]\!] \supseteq [\![t]\!]\}$, which require to have a domain that satisfies $\{[\![t]\!] \mid t \in \textbf{Types}\} \subseteq \mathcal{D}$. This is not straightforward but doable, as shown in [CNV05].

Step 3. As for the λ-calculus we can use the definition of the model given by (10) to deduce some interesting relations. First of all, the reader may have already noticed that

$$ch(t) = ch^{+}(t) \wedge ch^{-}(t)$$

thus, strictly speaking, the ch constructor is nothing but syntactic sugar for the intersection above (henceforth, we will no longer consider this constructor and concentrate on the covariant and contravariant channel type constructors). Besides this relation, far

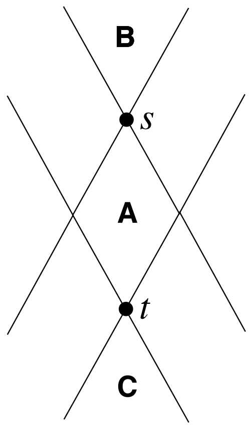


Fig. 1. Deciding atomicity

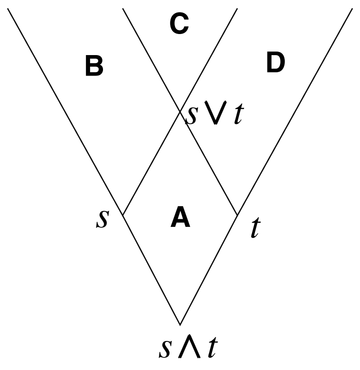


Fig. 2. Some equations

more interesting relations can be deduced and, quite remarkably, in many case this can be done graphically. Consider the definitions in (10): they tell us that the interpretation of $ch^{+}(t)$ is the set of the interpretations of all the types smaller than or equal to t. As such, it can be represented by the downward cone starting from t. Similarly, the upward cone starting from t represents $ch^{-}(t)$. This illustrated in Figure 1 where the upward cone B represents $ch^{-}(s)$ and the downward cone C represents $ch^{+}(t)$. If we now pass on Figure 2 we see that $ch^{-}(s)$ is the upward cone B+C and $ch^{-}(t)$ is the upward cone C+D. Their intersection is the cone C, that is the upward cone starting from the least upper bound of s and t which yields the following equation

$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t) \ . \tag{11}$$

Similarly, note that the union of $ch^{-}(s)$ and $ch^{-}(t)$ is given by B+C+D and that this is strictly contained in the upward cone starting from $s \wedge t$, since the latter also contains the region A, whence the strictness of the following containment:

$$ch^{-}(s) \vee ch^{-}(t) \lneq ch^{-}(s \wedge t) \ . \tag{12}$$

Actually, the difference of the two types in the above inequality is the region A which represents $ch^{+}(s \vee t) \wedge ch^{-}(s \wedge t)$, from which we deduce

$$ch^{-}(s \wedge t) = ch^{-}(s) \vee ch^{-}(t) \vee (ch^{+}(s \vee t) \wedge ch^{-}(s \wedge t)) \ .$$

We could continue to devise such equations, but the real challenge is to decide whether two generic types are one subtype of the other. As in the case for λ-calculus we can reduce the problem of subtyping two types to the decision of the emptiness of a type (the difference of the two types). If we put this type in disjunctive normal form, then it comes to decide whether $\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}$, that is whether $\bigwedge_{i \in P} t_i \leq \bigvee_{j \in N} t'_j$. With the type constructors specific to $\mathbb{C}\pi$ this expands to $\bigwedge_{i \in I} ch^{+}(t_1^i) \wedge \bigwedge_{j \in J} ch^{-}(t_2^j) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$. Since intersections can always be pushed inside type constructors (we saw it in (11) for ch^{-} types, the reader can easily check it for ch^{+}), then we end up with checking the following inequality:

$$ch^{+}(t_1) \wedge ch^{-}(t_2) \quad \leq \quad \bigvee_{h\in H} ch^{+}(t_3^h) \vee \bigvee_{k\in K} ch^{-}(t_4^k)\ . \tag{13}$$

This is indeed the most difficult part of the job, since while in some cases it is easy to decide the inclusion above (for instance, when $t_2 \not\leq t_1$ since then the right-hand side is empty), in general, this requires checking whether a type is *atomic*, that is whether its only proper subtype is the empty type (for sake of simplicity the reader can think of the atomic types as the singletons of the type system[6]). The general case is treated in [CNV05], but to give an idea of the problem consider the equation above with only two types s and t with $t \lneq s$, and let us try to check if:

$$ch^{+}(s) \wedge ch^{-}(t) \leq ch^{-}(s) \vee ch^{+}(t)\ .$$

Once more, a graphic representation is quite useful. The situation is represented in Figure 1 where the region A represents the left-hand side of the inequality, while the region B+C is the right hand side. So to check the subtyping above we have to check whether A is contained in B+C. At first sight these two regions looks completely disjoint, but observe that they have at least two points in common, marked in bold in the figure (they are respectively the types $ch(s)$ and $ch(t)$). Now, the containment holds if the region A does not contain any other type besides these two. This holds true if and only if there is no other type between s and t, that is if and only if $s\backslash t$ (i.e. $s \wedge \neg t$) is an atomic type.

Step 4. The next step is to devise a π-calculus that fits the type system we have just defined. Consider the dual of equation (12):[7]

$$ch^{+}(s) \vee ch^{+}(t) \lneq ch^{+}(s \vee t) \tag{14}$$

and in particular the fact that the inclusion is strict. A suitable calculus must distinguish the two types above. The type on the left contains either channels on which we will always read s-objects or always read t-objects, while the type on the right contains channels on which objects of type s or of type t may arrive interleaved. If we use a channel with the left type and we can test the type of the first message we receive on it, then we can safely assume that all the following messages will have the same type. Clearly using in such a context a channel with the type on the right would yield a runtime type error, so the two types are observationally different. This seems to suggest that a suitable calculus should be able to test the types of the messages received on a channel, which yields to the following definition of the calculus:

$$\begin{array}{lll} \textit{Channels} & \alpha & ::= x \mid c^t \\ \textit{Processes} & P & ::= \overline{\alpha}(\alpha) \mid \sum_{i\in I} \alpha(x:t_i)P_i \mid P_1 \| P_2 \mid (\nu c^t)P \mid !P \end{array}$$

The main difference with respect to the standard π-calculus is that we have introduced channel *values*, since a type-case must not be done on open terms. Thus, c^t is a physical box that can transport objects of type t: channels are tightly connected to the type of the objects they transport. Of course, restrictions are defined on channels, rather than

[6] Nevertheless, notice that according to their definition, atomic types may be neither singletons nor finite. For instance $ch(\mathbb{0})$ is atomic, but in the model of values it is the set of all the synchronisation channels; these are just token identifiers on a denumerable alphabet, thus the type is denumerable as well.

[7] To check this inequality turn Figure 2 upside down.

channel variables (since the latter could be never substituted). The type-case is then performed by the following reduction rule

$$\overline{c}_1^s(c_2^t) \parallel \textstyle\sum_{i\in I} c_1^s(x:t_i)P_i \quad \rightarrow \quad P_j[c_2^t/x] \qquad\qquad \text{if } ch(t) \leq t_j$$

This is the usual π-calculus reduction with three further constraints: (i) synchronisation takes place on channels (rather than on channel variables),(ii) it takes place only if the message is a channel value (rather than a variable), and (iii) only if the type of the message (which is $ch(t)$) matches the type t_j of the formal parameter of the selected branch of the summation. The last point is the one that implements the type-case. It is quite easy to use intersection and negation types to force the various branches of the summation to be mutually exclusive or to obey to a first match policy. We leave it as an exercise to the reader.

As usual, the type system assigns a type to messages (i.e. channels) and checks well-typing of processes. It is very compact and we report it below

$$\frac{}{\Gamma \vdash c^t : ch(t)}\ (\text{chan}) \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{var}) \qquad \frac{\Gamma \vdash \alpha : s \leq_{\mathcal{B}} t}{\Gamma \vdash \alpha : t}\ (\text{subsum})$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P}\ (\text{new}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\text{repl}) \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \| P_2}\ (\text{para})$$

$$\begin{smallmatrix} t \leq \bigvee_{i\in I} t_i \\ t_i \wedge t \neq \mathbb{0} \end{smallmatrix}\ \frac{\Gamma \vdash \alpha : ch^+(t) \quad \Gamma, x : t_i \vdash P_i}{\Gamma \vdash \sum_{i\in I} \alpha(x:t_i).P_i}\ (\text{input}) \qquad \frac{\Gamma \vdash \beta : t \quad \Gamma \vdash \alpha : ch^-(t)}{\Gamma \vdash \overline{\alpha}(\beta)}\ (\text{output})$$

The rules are mostly self explaining. The only one that deserves some comments is (input), which checks that the channel α can be used to read, and that each branch is well-typed. Note the two side conditions of the rule: they respectively require that for every t message arriving on α there must be at least one branch able to handle it (thus $t \leq \bigvee_{i\in I} t_i$ forces summands to implement exhaustive type-cases), and that for every branch there must be some message that may select it (thus the $t_i \wedge t \neq \mathbb{0}$ conditions ensure the absence of dead branches). Also note that in the subsumption rule we have used the subtyping relation induced by the bootstrap model (the one we outlined in the previous step) and that in rule (new) the environment is the same in the premise and the conclusion since we restrict channels and not variables.

Step 5. The last step consists in verifying whether the set of values induces the same subtyping relation as the bootstrap model, which is indeed the case. Note that here the only values are the channel constants and that they are typed just by two rules, (chan) and (subsum). Note also that, contrary to the λ-calculus, we do not need to explicit consider in the typing rule intersections with negated types. This point will be discussed in Section 4.6.

4 Challenges, Perspectives, and Open Problems

In the previous sections we gave a brief overview of the semantic subtyping approach and a couple of instances of its application. Even though we have done drastic simplifications to the calculi we have considered, this should have given a rough idea of the basic mechanisms and constitute a sufficient support to understand the challenges, perspectives, and open problems we discuss next.

4.1 Atomic Types

We have shown that in order to decide the subtyping relation in $\mathbb{C}\pi$ one must be able to decide the atomicy of the types (more precisely, one must be able to decide whether a type contains a finite number of atomic types and, if this is the case, to enumerate them). Quite surprisingly the same problem appears in λ-calculus (actually, in any semantic subtyping based system) as soon as we try to extend it with polymorphic types. Imagine that we embed our types with type variables $X, Y, \ldots$. Then the "natural" (semantic) extension of the subtyping relation is to quantify the interpretations over all substitutions for the type variables:

$$t_1 \leq t_2 \quad \overset{\text{def}}{\iff} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!] \,. \tag{15}$$

Consider now the following inequality (taken from [HFC05]) where t is a closed type

$$(t, X) \leq (t, \neg t) \vee (X, t). \tag{16}$$

We may need to check such an inequality when type-checking the body of a function polymorphic in X where we apply a function whose domain is the type on the right to an argument of the type on the left.

It is easy to see that this inequality holds if and only if t is atomic. If t is not atomic, then it has at least one non-empty proper subtype, and (15) does not hold when we substitute this subtype for X. If instead t is atomic, then for all X either $t \leq X$ or $t \leq \neg X$: if the second case holds then X is contained in $\neg t$, thus the left hand type of the inequality is included in the first clause on the right hand type. If X does contain t, then all the elements on the left except those in (t, t) are included by the first clause on the right, and the elements in (t, t) are included by the second clause.

Note that the example above does not use any fancy or powerful type constructor, such arrows or channels: it only uses products and type variables. So the problem we exposed is not a singularity but applies to all polymorphic extensions of semantic subtyping where, once more, deciding subtyping reduces to deciding whether some type is atomic or not.

Since these atomic types pop out so frequently a first challenge is understanding why this happens and therefore how much the semantic subtyping technique is tightened to the decidability of atomicity. In particular, we may wonder whether it is reasonable and possible to study systems in which atomic types are not denotable so that the set of subtypes of a type is dense.

4.2 Polymorphic Types

The previous section shows that the atomic types problem appears both in $\mathbb{C}\pi$ and in the study of polymorphism for $\mathbb{C}$Duce. From a theoretical viewpoint this is not a real problem: in the former case Castagna *et al.* [CNV05] show atomicity to be decidable, which implies the decidability of the subtyping relation; in the functional case Hosoya *et al.* [HFC05] argue that the subtyping relation based on the substitution interpretation can be reduced to the satisfiability of a constraint system with negative constraints.

However, from a practical point of view the situation is way more problematic, probably not in the case of $\mathbb{C}\pi$, since it is not intended for practical immediate applications, but surely is in the case targeted by Hosoya *et al.* since the study performed there is intended to be applied to programming languages, and no practical algorithm to solve this case is known. For this reason in [HFC05] a different interpretation of polymorphic types is given. Instead of interpreting type variables as “place holders” where to perform substitutions, as stated by equation (15), they are considered as “marks” that indicate the parametrised subparts of the values. The types are then interpreted as sets of marked values (that is, usual values that are marked by type variables in the correspondence of where these variables occur in the type), and are closed by mark erasure. Once more, subtyping is then set containment (of mark-erasure closed sets of marked values) which implies that the subtyping relation must either preserve the parametrisation (i.e. the marks), or eliminate part of it in the supertype. More specifically, this requires that the type variables in the supertype are present in the same position in the subtype. This rules out critical cases such as the one of equation (16) which does not hold since no occurrence of X in the type on the left of the equation corresponds to the X occurring in the type on the right.

Now, the marking approach works because the only type constructor used in [HFC05] is the product type, which is covariant. This is enough to model XML types, and the polymorphism one obtains can be (and actually is) applied to the XDuce language. However, it is still matter of research how to implement the marking approach in the presence of contravariant type constructors: first and foremost in the presence of arrow types, as this would yield the definition of a polymorphic version of $\mathbb{C}$Duce; but also, it would be interesting to study the marking approach in the presence of the contravariant ch^{-} type constructor, to check how it combines with the checks of atomicity required by the mix with the ch^{+} constructor, and see whether markings could suggests a solution to avoid this check of atomicity.

On a different vein it would be interesting to investigate the relation between parametricity (as intended in [Rey83, ACC93, LMS93]) and the marking approach to polymorphism. According to parametricity, a function polymorphic (parametric) in a type variable X cannot look at the elements of the argument which have type X, but must return them unmodified. Thus with respect to those elements, the function is just a rearranging function and it behaves uniformly on them whatever the actual type of these elements is. Marks have more or less the same usage and pinpoint those parts of the argument that must be used unchanged to build the result (considering the substitution based definition of the subtyping relation would correspond to explore the semantic properties of the type parameters, as the example of equation (16) clearly shows). However, by the presence of “ad hoc” polymorphism (namely, the type-case construction evocated in Footnote 5 and discussed in Section 4.5) the polymorphic functions in [HFC05] can look at the type of the parametric (i.e. marked) parts of the argument, decompose it, and thus behave differently according to the actual type of the argument. Therefore, while the marking approach and parametric polymorphism share the fact that values of a variable type are never constructed, they differ in presenting a uniform behaviour whatever the type instantiating a type variable is.

Another direction that seems worth pursuing is to see if it is possible to recover part of the substitution based polymorphic subtyping as stated by equation (15), especially in $\mathbb{C}\pi$ where the test of atomicity is already necessary because of the presence of the covariant and contravariant cones.

Finally, one can explore a more programming language oriented approach and check whether it is possible to define reasonable restrictions on the definition of polymorphic functions (for instance by allowing polymorphism to appear only in top-level functions, by forbidding a type variable to occur both in covariant and in contravariant position, by constraining the use of type variables occurring in the result type of polymorphic functions, etc.) so that the resulting language provides the programmer with sufficient polymorphism for practical usage, while keeping it simple and manageable.

4.3 The Nature of Semantic Subtyping

The importance of atomic types also raises the question about the real nature of the semantic subtyping, namely, is semantic subtyping just a different way to axiomatise a subtyping relation that could be equivalently axiomatised by classic syntactic techniques, or is it something different? If we just look at the $\mathbb{C}$Duce case, then the right answer seems to be the first one. As a matter of facts, we could have probably arrived to define the same system without resorting to the bootstrapping technique and the semantic interpretation, but just finding somehow the formulae (6) and (7) and distributing them at the premises of some inference rules in which the types (4) and (5) are equated to $\mathbb{0}$. Or alternatively we could have arrived to this system by looking at the axiomatisation for the positive fragment of the $\mathbb{C}$Duce type system given in [DCFGM02], and trying to extend it to negation types.

But if we consider $\mathbb{C}\pi$, then we are no longer sure about the right answer. In Section 3 we just hinted at the fact that checking subtyping involves checking whether some types are atomic, but we did not give further details. The complete specification can be found in Theorem 2.6 of [CNV05], and involves the enumeration of all the atomic types of a finite set. Looking at that definition, it is unclear whether it can be syntactically axiomatised, or defined with a classical deduction system. Since the relation is proved to be decidable, then probably the answer is yes. But it is clear that finding such a solution without resorting to semantic techniques would have been very hard, if not impossible. And in any case one wonders whether in case of a non decidable relation this would be possible at all.

As a matter of facts, we do not know whether the semantic subtyping approach is an innovative approach that yields to the definition of brand new type systems or it is just a new way to define old systems (or rather, systems that could be also defined in the good old syntactic way). Whichever the answer is, it seems interesting trying to determine the limits of the semantic subtyping approach, that is, its degree of freedom. More precisely, the technique to “close the circle” introduced in [FCB02] and detailed in [CF05] is more general than the one presented here in the introduction. Instead of defining a particular model it is possible to characterise a class of models which are those that induce the same containment relation as the intended semantics (that is, that satisfy an equation such as (10) but customised for the type constructors at issue). This relies on the definition of an auxiliary function—called the extensional interpretation [FCB02]—

which fixes the intended semantics for the type constructors. So a more technical point is to investigate whether and to which extent it is possible to systematise the definition of the extensional interpretation. Should one start with a given model of values and refine it, or rather try to find a model and then generalise it? And what are the limits of such a definition? For instance, is it possible to define an extensional interpretation which induces a containment where the inequality (14) is an equality? And more generally is it possible to characterise, even roughly, the semantic properties that could be captured by a model? Because, as we show in the next section, there are simple extensions of the type systems we met so far for which a model does not exist.

4.4 Recursive Types and Models

As we said at the end of the introduction, to complete the 5 steps of the semantic subtyping approach is far from being trivial. One of the main obstacles, if not the main one, may reside in the definition of a model. Not only that the model may be hard to find, but also that sometimes it does not exist. A simple illustration of this can be given in $\mathbb{C}\pi$. In the first step of the definition of $\mathbb{C}\pi$ in Section 3 we carefully specified that the types were not recursive: as pointed out in [CNV05], if we allow recursion inside channel types, then there does not exist any model. To see why, consider the following recursive type:

$$t = \texttt{int} \vee (ch(t) \wedge ch(\texttt{int}))\ .$$

If we had a model, then either $t = \texttt{int}$ or $t \neq \texttt{int}$ hold. Does $t = \texttt{int}$? Suppose it does, then $ch(t) \wedge ch(\texttt{int}) = ch(\texttt{int})$ and $\texttt{int} = t = \texttt{int} \vee ch(\texttt{int})$, which is not true since $ch(\texttt{int})$ is not contained in $\texttt{int}$. Therefore it must be $t \neq \texttt{int}$. According to our semantics this implies $ch(t) \wedge ch(\texttt{int}) = \mathbb{0}$, because they are interpreted as two distinct singletons (whence the invariance of ch types). Thus $t = \texttt{int} \vee \mathbb{0} = \texttt{int}$, contradiction. The solution is to avoid recursion inside channel types, for instance by requiring that on every infinite branch of a regular type there are only finitely many occurrences of the channel type constructors. Nevertheless, this is puzzling since the natural extension with recursion is inconsistent.

It is important to notice that this problem is not a singularity of $\mathbb{C}\pi$: it also appears in $\mathbb{C}$Duce as soon as we extend its type system by reference types, as explained in [CF05]. This raises the problem to understand what the non-existence of a model means. Does it correspond to a limit of the semantic subtyping approach, a limit that some different approach could overcome, or does it instead characterise some mathematical properties of the semantics of the types, by reaching the limits that every semantic interpretation of these types cannot overcome?

Quite remarkably the restriction on recursive types in $\mathbb{C}\pi$ can be removed, by moving to a *local* version of the calculus [Mer00], where only the output capability of a channel can be communicated. This can be straightforwardly obtained by restricting the syntax of input processes so that they only use channel constants (that is, $\sum_{i\in I} c^t(x:t_i)P_i$ instead of $\sum_{i\in I} \alpha(x:t_i)P_i$), which makes the type $ch^+(t)$ useless. Without this type, the example at the beginning of this section cannot be constructed (by removing $ch^+(t)$ we also remove $ch(t)$ which is just syntactic sugar) and indeed it is possible build a model of the types with full recursion. The absence of input channel types makes also the decision algorithm considerably simpler since equation (13) becomes:

$$ch^{-}(s) \leq \bigvee_{k\in K} ch^{-}(t_k) \tag{17}$$

and it is quite easy to check (e.g. graphically) that (17) holds if and only if there exists $k \in K$ such that $t_k \leq s$. Last but not least, the types of the local version of of $\mathbb{C}\pi$ are enough to encode the type system of $\mathbb{C}$Duce (this is shown in Section 4.7). However, as we discuss in Section 4.6, new problems appear (rather, old problems reappear). So the approach looks like too a short blanket, that if you pull it on one side uncovers other parts and seems to reach the limits of the type system.

4.5 Type-Case and Type Annotations

Both $\mathbb{C}$Duce and $\mathbb{C}\pi$ make use of type-case constructions. In both cases the presence of a type-case does not look strictly necessary to the development, but it is strongly supported, if not induced, by some semantic properties of the models. We already discussed these points while presenting the two calculi.

For $\mathbb{C}$Duce we argued that equation (8) and in particular the fact that the subtyping inequality it induces

$$(t_1 \vee t_2) \rightarrow (s_1 \wedge s_2) \lneq (t_1 \rightarrow s_1) \wedge (t_2 \rightarrow s_2) \tag{18}$$

is in general strict, suggests the inclusion of overloaded function in the language to distinguish the two types: an overloaded function can have different behaviour for t_1 and t_2, so it can belong to the right hand side, and not to the left hand side where all the functions uniformly return results in the intersection of s_1 and s_2. Of course, from a strict mathematical point of view it is not necessary for a function to be able to distinguish on the type of their argument in order to be in the right hand side but not in the left one: it suffices that it takes, say, a single point of t_1 into s_1/s_2 to be in the difference of the two types. If from a mathematical viewpoint this is the simplest solution from a programming language point of view, this is not the easy way. Indeed we want to be able to program this function (as long as we want that the model based on values induces the same subtyping relation as the bootstrap model). Now imagine that t_1 and t_2 are function types. Then a function which would have a different behaviour on just one point could not be programmed by a simple equality check on the input (such as "if the input is the point at issue then return a point in s_1/s_2 otherwise return something in s_2") as we cannot check equality on functions: the only thing that we can do is to apply them. This would imply a non trivial construction of classes of functions which have a distinguished behaviour on some specific points. It may be the case that such construction does not result technically very difficult (even if the presence of recursive types suggests the contrary). However constructing it would not be enough since we should also type-check it and, in particular, to prove that the function is typed by the difference of the two types in (18): this looks as an harder task. From a programming language perspective the easy mathematical solution is the difficult one, while the easy solution, that is the introduction of type-cases and of overloaded functions, has a hard mathematical sense (actually some researchers consider it as a mathematical non-sense).

For $\mathbb{C}\pi$ we raised a similar argument about the strictness of

$$ch^{+}(s) \vee ch^{+}(t) \lneq ch^{+}(s \vee t)$$

The presence of a type-case in the processes is not strictly necessary to the existence of the model (values do not involve processes but just messages) but it makes the two types observationally distinguishable. One could exclude the type-case from the language, but then we would have a too restrictive subtyping relation because it would not let values in the right type to be used where values of the left type are expected, even if the two types would not be operationally distinguishable: it would be better in that case to have the equality hold (as in the system defined by Hennessy and Riely [HR02] where no type-case primitive is present).

These observations make us wonder how much the semantic subtyping approach is bound to the presence of a type-case. We also see that if for instance in ℂDuce we try to provide a language without overloading, the formal treatment becomes far more difficult (see Section 5.6 of [Fri04]). Therefore one may also wonder whether the semantic subtyping approach is unfit to deal with languages that do not include a type case. Also, since we have a type-case, then we annotated explicitly by their type some values: λ-abstractions in ℂDuce and channel constants in $\mathbb{C}\pi$. One may wonder if any form of partial type reconstruction is possible[8], and reformulate the previous question as whether the semantic subtyping approach is compatible with any form of type reconstruction.

The annotations on the λ-abstractions raise even deeper questions. Indeed all the machinery on λ-calculus works because we added these explicit annotations. The point is that annotations and abstractions constitute an indissociable whole, since in the presence of a type-case the annotations observably change the semantics of the abstractions: using two different annotations on the same λ-abstraction yields two different behaviours of the program they are used in. For instance $\lambda^{Odd\to Odd\wedge Even\to Even}x.x$ will match a type-case on $Odd \to Odd$ while $\lambda^{Int\to Int}x.x$ will not. We are thus deeply changing the semantics of the elements of a function space, or at least of the λ-terms as usually intended. This raises a question which is tighten to—but much deeper than—the one raised by Section 4.3, namely which is the mathematical or logical meaning of the system of ℂDuce, and actually, is there any? A first partial answer to this question has been answered by Dezani *et al.* [DCFGM02] who showed that the subtyping relation induced by the model of Section 2 restricted to its positive part (that is arrows, unions, intersections but no negations) coincides with the relevant entailment of the $\mathbf{B}_+$ logic (defined 30 years before). However, whether analogous characterisations of the system with negation exist is still an open question. This seems a much more difficult task since the presence of negation requires deep modifications in the semantics of functions and in their typing. Thus, it still seems unclear whether the semantic subtyping technique for λ-calculus is just a syntactic hack that makes all the machinery work, or it hides some underlying mathematical feature we still do not understand.

4.6 Language with Enough Points and Deduction of Negations

We have seen in step 4 of Section 2, that lambda abstractions are typed in an unorthodox way, since the rule (abstr) can subtract any finite number of arrow types as long as the type is non-empty. In step 5 of the same section we justified this rule by the fact that we wanted a language that provided enough values to inhabit all the non-empty types.

[8] Full type reconstruction for ℂDuce is undecidable, since it already is undecidable for the λ-calculus with intersection types where typability is equivalent to strong normalizability.

This property is important for two reasons: (i) if the bootstrap model and the model of values induce the same subtyping relation, then it is possible to consider types as set of values, which is an easier concept to reason on (at least for a programmer) than the bootstrap model, and (ii) if two different types cannot be distinguished by a value than we would have too a constraining type system, since it would forbid to interchange the values of the two types even though the types are operationally indistinguishable.

The reader may have noticed that we do not have this problem in $\mathbb{C}\pi$. Indeed given a value, that is a channel c^t, it is possible to type it with the negation of all channel types that cannot be deduced for it. In particular we can deduce for c^t the types $\neg ch^{+}(s_1)$ and $\neg ch^{-}(s_2)$ for all $s_1 \not\geq t$ and $s_2 \not\leq t$, and this is simply obtained by subsumption, since it is easy to verify that all these types are supertypes of the minimum type deduced for c^t that is $ch^{+}(t) \wedge ch^{-}(t)$. For instance if $s_1 \not\geq t$ then $ch^{-}(t) \leq \neg ch^{+}(s_1)$ and so is the intersection.

But subsumption does not work in the case of $\mathbb{C}$Duce: to deduce by subsumption that $\lambda^{Int \to Int} x.(x+3) : \neg(Bool \to Bool)$ one should have $Int \to Int \leq \neg(Bool \to Bool)$, which holds if and only if $Int \to Int \wedge Bool \to Bool$ is empty, which is not since it contains the overloaded functions of the corresponding type.

Interestingly, the same problem pops up again if we consider the local version of $\mathbb{C}\pi$. In the absence of covariant channels it is no longer possible to use the same rules to deduce that c^t has type $\neg ch^{-}(s)$ for $s \not\leq t$. Indeed we can only deduce that $c^t : ch^{-}(t)$ and this is *not* a subtype of $\neg ch^{-}(s)$ (since $ch^{-}(t) \wedge ch^{-}(s)$ is non-empty: it contains $c^{s \vee t}$), thus subsumption does not apply and we have to modify the rule for channels, so that is uses the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^{-}(t) \wedge \neg ch^{-}(t_1) \wedge \ldots \wedge \neg ch^{-}(t_n)} \text{ (chan)}$$

Having rules of this form is quite problematic. First because one loses the simplicity and intuitiveness of the approach, but more importantly because the system no longer satisfies the minimum typing property, which is crucial for the existence of a typing algorithm. The point is that the minimum "type" of an abstraction is the intersection of the type at its index with all the negated arrow types that can be deduced for it. But this is not a type since it is an infinite intersection, while in our type system only finite intersections are admitted.[9] In order to recover the minimum typing property and define a type algorithm, Alain Frisch [Fri04] introduces some syntactic objects, called schemas, that represent in a finite way the infinite intersection above, but this does not allow the system to recover simplicity and makes it lose its set-theoretic interpretation.

Here it is, thus, yet another problematic behaviour shared between $\mathbb{C}$Duce and $\mathbb{C}\pi$. So once more the question is whether this problem is a limitation of semantic subtyping or it is implicit in the usage of negation types. And as it can be "solved" in the case case of $\mathbb{C}\pi$ by considering the full system instead of just the local version, is it possible to find similar (and meaningful) solutions in other cases (notably for $\mathbb{C}$Duce)?

[9] Of course the problem could be solved by annotating values (channels or λ-abstractions) also with negated types and considering it as the minimum type of the value. But it seems to us an aberration to precisely state all the types a term does not have, especially from a programming point of view, since it would require the programmer to forecast all the possible usages in which a function must not have some type. In any case in the perspective of type reconstruction evocated in the previous section this is a problem that must be tackled.

Finally, it would be interesting to check whether the semantic subtyping type system could be used to define a denotational semantics of the language by relating the semantic of an expression with the set of its types and (since our type system is closed by finite intersections and subsumption) build a filter model [BCD83].

4.7 The Relation Between $\mathbb{C}\pi$ and $\mathbb{C}$Duce

There exist several encodings of the λ-calculus into the π-calculus (see part VI of [SW02] for several examples), so it seems interesting to study whether the encoding of $\mathbb{C}$Duce into $\mathbb{C}\pi$ is also possible. In particular we would like to use a continuation passing style encoding as proposed in Milner's seminal work [Mil92] according to which a λ-abstraction is encoded as a (process with a free) channel that expects two messages, the argument to which the function must be applied and a channel on which to return the result of the application. Of course, in the $\mathbb{C}$Duce/$\mathbb{C}\pi$ case a translation would be interesting only if it preserved the properties of the type system, in particular the subtyping relation. In other terms, we want that our translation $\{\!\!\{\,\}\!\!\} : \mathbf{Types}_{\mathbb{C}\mathrm{duce}} \to \mathbf{Types}_{\mathbb{C}\pi}$ satisfies the property $t = \mathbb{0}$ if and only if $\{\!\!\{t\}\!\!\} = \mathbb{0}$ (or equivalently $s \leq t$ iff $\{\!\!\{s\}\!\!\} \leq \{\!\!\{t\}\!\!\}$).

Such an encoding can be found in a working draft we have been writing with Mariangiola Dezani and Daniele Varacca [CDV05]. The work presented there starts from the observation that the natural candidate for such an encoding, namely the typed translation used in [SW02] for $\lambda V^{\rightarrow}$ (the simply-typed call-by-value λ-calculus) and defined as $\{\!\!\{s \to t\}\!\!\} = ch^{-}(\{\!\!\{s\}\!\!\} \times ch^{-}(\{\!\!\{t\}\!\!\}))$ does not work for $\mathbb{C}$Duce/$\mathbb{C}\pi$ (from now on we will omit the inner mapping parentheses and write $ch^{-}(s \times ch^{-}(t))$ instead). This can be seen by considering that the following equality holds in $\mathbb{C}$Duce

$$s \to (t \wedge u) \quad = \quad (s \to t) \wedge (s \to u) \tag{19}$$

while if we apply the encoding above, the translation of the left hand side is a subtype of the translation of the right hand side but not viceversa. Once more, this is due to the strictness of some inequality, since the translation of the codomain of the left hand side $ch^{-}(t \wedge u)$, contains the translation of the codomains of the right hand side $ch^{-}(t) \vee ch^{-}(u)$ (use equation (11) and distribute union over product) but not viceversa.

So the idea of [CDV05] is to translate $s \to t$ as $ch^{-}(s \times ch^{\lambda}(t))$ where $ch^{\lambda}(t)$ is a type that is (i) contravariant (since it must preserve the covariance of arrow codomains), (ii) satisfies $ch^{\lambda}(t \wedge u) = ch^{\lambda}(t) \vee ch^{\lambda}(u)$ (so that it preserves equation (19)) and (iii) is a supertype of $ch^{-}(t)$ (since we must be able to pass on it the channel on which the result of the function is to be returned).

Properties (i) and (ii) can be satisfied by adding a double negation as for $\neg ch^{-}(\neg t)$: the double negation preserves the contravariance of ch^{-} while the inner negation by De Morgan's laws yields the distributivity of the union. For (iii) notice that $\neg ch^{-}(\neg t) \setminus ch^{-}(t) = ch(\mathbb{1})$, so it suffices to add the missing point by defining $ch^{\lambda}(t) = \neg ch^{-}(\neg t) \vee ch(\mathbb{1})$. With such a definition the translation $ch^{-}(s \times ch^{\lambda}(t))$ has the wanted properties. Actually, in [CDV05] it is proved that the three conditions above are necessary and sufficient to make the translation work, which induces a class of possible translations parametric in the definition of ch^{λ} (see [CDV05] for a characterisation of the choices for ch^{λ}). $ch^{\lambda}(t)$ is a supertype of $ch^{-}(t)$ but the latter is also the greatest channel type contained in $ch^{\lambda}(t)$. So there is gap between $ch^{\lambda}(t)$ and $ch^{-}(t)$ which constitutes the definition space of $ch^{\lambda}(t)$. What is the meaning of this gap, that is of $ch^{\lambda}(t) \setminus ch^{-}(t)$? We do not know, but it is surely worth of studying, since it has important consequences also in the interpretation of terms. The translation of terms is still work in progress, but we want here hint at our working hypotheses since they outline the importance of $ch^{\lambda}(t) \setminus ch^{-}(t)$. We want to give a typed translation of terms, where the translation of a term e of type t is a process with only one unrestricted channel α of type $ch^{-}(\{\!\{t\}\!\})$ (intuitively, this is the channel on which the process writes the result of e). We note this translation as $\{\!\{e\}\!\}_{\alpha}$. Consider the rule (abstr) for functions and note that the body of an abstraction is typed several times under several assumptions. If we want to be able to prove that the translation preserves typing, then the translation must mimic this multiple typing. This can be done in $\mathbb{C}\pi$ by using a summation, and thus by translating $\lambda^{\wedge_{i\in I} s_i \to t_i} x.e$ into a process that uses the unrestricted channel $\alpha : ch^{-}(\{\!\{\wedge_{i\in I} s_i \to t_i\}\!\}) = ch^{-}(ch^{-}(\vee_{i\in I}(s_i \times ch^{\lambda}(t_i))))$ as follows:

$$\{\!\{\lambda^{\wedge_{i\in I} s_i \to t_i} x.e\}\!\}_{\alpha} = (\nu f^{\vee_{i\in I}(s_i \times ch^{\lambda}(t_i))})(\overline{\alpha}(f) \parallel !\textstyle\sum_{i\in I} f(x : s_i, r : ch^{-}(t_i))\{\!\{e\}\!\}_r)$$

Unfortunately, the translation above is not correct since it is not exhaustive. More precisely, it does not cover the cases in which the second argument is in $ch^{\lambda}(t_i) \setminus ch^{-}(t_i)$: to type $\{\!\{e\}\!\}_r$ the result channel r must have type $ch^{-}(t_i)$ (since the only types the $\mathbb{C}$Duce type system deduces for e are the t_i's), but the encoding of arrow types uses $ch^{\lambda}(t_i)$ in second position. Thus, it seems important to understand what the difference above means. Is it related to the negation of arrow types in the (abstr) rule? Note that in this section we worked on the local version of $\mathbb{C}\pi$, so we have recursive types (Section 4.4) but also negated channels in the (chan) typing rule (Section 4.6). If for local $\mathbb{C}\pi$ we use the rule without negations then $ch^{\lambda}(t) \setminus ch^{-}(t) = \varnothing$, so the encoding above works but we no longer have a consistent type system. We find again that our blanket is too short to cover all the cases. Is this yet another characterisation of some limits of the approach? Does it just mean that Milner's translation is unfit to model overloading? Or does it instead suggest that the encoding of $\mathbb{C}$Duce into $\mathbb{C}\pi$ has not a lot of sense and that we had better study how to integrate $\mathbb{C}$Duce and $\mathbb{C}\pi$ in order to define a concurrent version of $\mathbb{C}$Duce?

4.8 Dependent Types

As a final research direction for the semantic subtyping we want to hint at the research on dependent types. Dependent types raise quite naturally in type systems for the π-calculus (e.g. [YH00, Yos04]), so it seems a natural evolution of the study of $\mathbb{C}\pi$. Also, dependent types in the π-calculus are used to check the correctness of cryptographic protocols (see the research started by Gordon and Jeffrey [GJ01]) and unions, intersections, and negations of types look very promising to express properties of programs. Thus, it may be worth of study the definition of an extension of Gordon an Jeffrey systems with semantic subtyping, especially in the light of the connection of the latter with XML and the use of this in protocols for webservices.

Also quite interesting would be to study the extension of first order dependent type theory $\lambda\Pi$ [HHP93]. As far as we know, all the approaches to add subtyping to $\lambda\Pi$ are quite syntactic, since the subtyping relation is defined on β_2 normal forms (see for instance [AC96] or, for an earlier proposal, [Pfe93]). Even more advanced subtype systems, such as [CC01, Che99], still relay on syntactic properties such as the strong normalisation of the β_2-reduction, since the subtyping rules essentially mimic the β_2-reduction procedure. It would then be interesting to check whether the semantic subtyping approach yields a more semantic characterisation of the subtyping relation for dependent types.

5 Conclusion

The goal of this article was twofold: (i) to give an overview of the semantic subtyping approach and an aperçu of its generality by applying it both to sequential and concurrent systems and (ii) to show the new questions it raises. Indeed we were much more interested in asking questions than giving answers, and it is in this perspective that this paper was written. Some of the questions we raised are surely trivial or nonsensical, some others will probably soon result as such, but we do hope that at least one among them will have touched some interesting mathematical problem being worth of pursuing. In any case we hope to have interested the reader in this approach.

Acknowledgements. Most of this work was prepared while visiting Microsoft Research in Cambridge and it greatly benefited of the stimulating and friendly environment that I enjoyed during my stay, a clear break with respect to my current situation. In particular, I want to warmly thank Nick Benton (Section 4.5 results from a long discussion I had with him), Gavin Bierman, Luca Cardelli, and Andy Gordon. I want also to thank Mario Coppo, Mariangiola Dezani, and Daniele Varacca for their suggestions on various versions of this work. All my gratitude to Microsoft Research and all its personnel for their warm hospitality.

References

[AC96] D. Aspinall and A. Compagnoni. Subtyping dependent types. In *11th Ann. Symp. on Logic in Computer Science*, pages 86–97, 1996.

[ACC93] M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 21:9–58, 1993. Special issue in honour of Corrado Böhm.

[APP91] Martín Abadi, Benjamin Pierce, and Gordon Plotkin. Faithful ideal models for recursive polymorphic types. *International Journal of Foundations of Computer Science*, 2(1):1–21, March 1991. Summary in Fourth Annual Symposium on Logic in Computer Science, June, 1989.

[AW93] Alexander Aiken and Edward L. Wimmers. Type inclusion constraints and type inference. In *Proceedings of the Seventh ACM Conference on Functional Programming and Computer Architecture*, pages 31–41, Copenhagen, Denmark, June 93.

[BCD83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[CC01] G. Castagna and G. Chen. Dependent types with subtyping and late-bound overloading. *Information and Computation*, 168(1):1–67, 2001.

[CDV05] G. Castagna, M. Dezani, and D. Varacca. Encoding $\mathbb{C}$Duce into $\mathbb{C}\pi$. Working draft, February 2005.

[CF05] G. Castagna and A. Frisch. A gentle introduction to semantic subtyping. In Proceedings of *PPDP '05, the 7th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming*, Lisboa, Portugal, 2005. ACM Press. Joint ICALP-PPDP keynote talk.

[Che99] G. Chen. Dependent type system with subtyping. *Journal of Computer Science and Technology*, 14(1), 1999.

[CNV05] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the π-calculus. In *LICS '05, 20th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2005.

[Dam94] F. Damm. Subtyping with union types, intersection types and recursive types II. Research Report 816, IRISA, 1994.

[DCFGM02] M. Dezani-Ciancaglini, A. Frisch, E. Giovannetti, and Y. Motohama. The relevance of semantic subtyping. In *Intersection Types and Related Systems*. Electronic Notes in Theoretical Computer Science 70(1), 2002.

[FCB02] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic Subtyping. In *Proceedings, Seventeenth Annual IEEE Symposium on Logic in Computer Science*, pages 137–146. IEEE Computer Society Press, 2002.

[Fri04] Alain Frisch. *Théorie, conception et réalisation d'un langage de programmation fonctionnel adapté à XML.* PhD thesis, Université Paris 7, December 2004.

[GJ01] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *CSFW 2001: 14th IEEE Computer Security Foundations Workshop*, pages 145–159, 2001.

[HFC05] H. Hosoya, A. Frisch, and G. Castagna. Parametric polymorphism for XML. In *POPL '05, 32nd ACM Symposium on Principles of Programming Languages*. ACM Press, 2005.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[Hos01] Haruo Hosoya. *Regular Expression Types for XML.* PhD thesis, The University of Tokyo, 2001.

[HP01] Haruo Hosoya and Benjamin C. Pierce. Regular expression pattern matching for XML. In *The 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2001.

[HP03] H. Hosoya and B. Pierce. XDuce: A typed XML processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[HR02] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[LMS93] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. The genericity theorem and parametricity in the polymorphic λ-calculus. *Theor. Comput. Sci.*, 121(1-2):323–349, 1993.

[Mer00] Massimo Merro. *Locality in the pi-calculus and applications to distributed objects.* PhD thesis, Ecole des Mines de Paris, Nice, France, 2000.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Pfe93] F. Pfenning. Refinement types for logical frameworks. In *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*, May 1993.

[Rey83] J.C. Reynolds. Types, abstractions and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, 1983.

[SW02] D. Sangiorgi and D. Walker. *The π-calculus.* Cambridge University Press, 2002.

[YH00] N. Yoshida and M. Hennessy. Assigning types to processes. In *Proc. of the 15th IEEE Symposium on Logic in Computer Science*, pages 334–348, 2000.

[Yos04] Nobuko Yoshida. Channel dependent types for higher-order mobile processes. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 147–160. ACM Press, 2004.

Biological Systems as Reactive Systems

Luca Cardelli

Microsoft Research

Systems Biology is a new discipline aiming to understand the behavior of biological systems as it results from the (non-trivial, ”emergent”) interaction of biological components. We discuss some biological networks that are characterized by simple components, but by complex interactions. The components are separately described in stochastic pi-calculus, which is a ”programming language” that should scale up to description of large systems. The components are then wired together, and their interactions are studied by stochastic simulation. Subtle and unexpected behavior emerges even from simple circuits, and yet stable behavior emerges too, giving some hints about what may be critical and what may be irrelevant in the organization of biological networks

Mobile Agents Computing: Security Issues and Algorithmic Solutions

Nicola Santoro

School of Computer Science, Carleton University, Canada
santoro@scs.carleton.ca

Abstract. The use of *mobile agents* is becoming increasingly popular when computing in networked environments, ranging from Internet to the Data Grid, both as a theoretical computational paradigm and as a system-supported programming platform.

In networked systems that support autonomous mobile agents, the main theoretical concern is how to develop efficient agent-based *system protocols*; that is, to design protocols that will allow a team of identical simple agents to cooperatively perform (possibly complex) system tasks.

The computational problems related to these operations are definitely non trivial, and a great deal of theoretical research is devoted to the study of conditions for the solvability of these problems and to the discovery of efficient algorithmic solutions.

At a practical level, in these environments, *security* is the most pressing concern, and possibly the most difficult to address. Actually, even the most basic security issues, in spite of their practical urgency and of the amount of effort, must still be effectively addressed.

Among the severe security threats faced in distributed mobile computing environments, two are particularly troublesome: *harmful agent* (that is, the presence of malicious mobile processes), and *harmful host* (that is, the presence at a network site of harmful stationary processes).

The former problem is particularly acute in unregulated non-cooperative settings such as Internet (e.g., e-mail transmitted viruses). The latter not only exists in those settings, but also in environments with regulated access and where agents cooperate towards common goals (e.g., sharing of resources or distribution of a computation on the Grid). In fact, a local (hardware or software) failure might render a host harmful.

In this talk I will concentrate on two security problems, one for each type: *locating a black hole*, and *capturing an intruder*; I will describe the recent algorithmic solutions and remaining open questions for both problems.

Efficient Algorithms for Detecting Regular Point Configurations

Luzi Anderegg[1], Mark Cieliebak[1], and Giuseppe Prencipe[2]

[1] ETH Zurich
{anderegg, cieliebak}@inf.ethz.ch
[2] Università di Pisa
prencipe@di.unipi.it

Abstract. A set of n points in the plane is in *equiangular configuration* if there exist a center and an ordering of the points such that the angle of each two adjacent points w.r.t. the center is $\frac{360}{n}$, i.e., if all angles between adjacent points are equal. We show that there is at most one center of equiangularity, and we give a linear time algorithm that decides whether a given point set is in equiangular configuration, and if so, the algorithm outputs the center. A generalization of equiangularity is *σ-angularity*, where we are given a string σ of n angles and we ask for a center such that the sequence of angles between adjacent points is σ. We show that σ-angular configurations can be detected in time $O(n^4 \log n)$.

Keywords: Weber point, equiangularity, σ-angularity, design of algorithms, computational geometry.

1 Introduction

We study how to identify geometric configurations that are in a sense generalizations of stars: a set of n distinct points P in the plane is in *equiangular configuration* if there exists a point $c \notin P$ — the *center of equiangularity* — and an ordering of the points such that each two adjacent points form an angle of $\frac{360}{n}$ w.r.t. c (see Figure 1(a)).

Obviously, if all points have the same distance from the center, then they form a regular star. Note that we exclude the special case that any of the given points is at center c. Furthermore, observe that the number of points in equiangular configurations can be odd or even, and that any set of two points is always in equiangular configuration. In the remainder of this paper, we will consider only point sets with at least three points.

There is a strong connection between equiangular configurations and *Weber points*, which are defined as follows: a point w is a Weber point of point set P if it minimizes $\sum_{p \in P} |p - x|$ over all points x in the plane, where $|p - x|$ denotes the Euclidean distance between p and x [8]. Hence, a Weber point minimizes the sum of all distances between itself and all points in P. The Weber point for an arbitrary point set is unique except for the case of an even number of points which are all on a line [3]. We will show that the center of equiangularity, if it

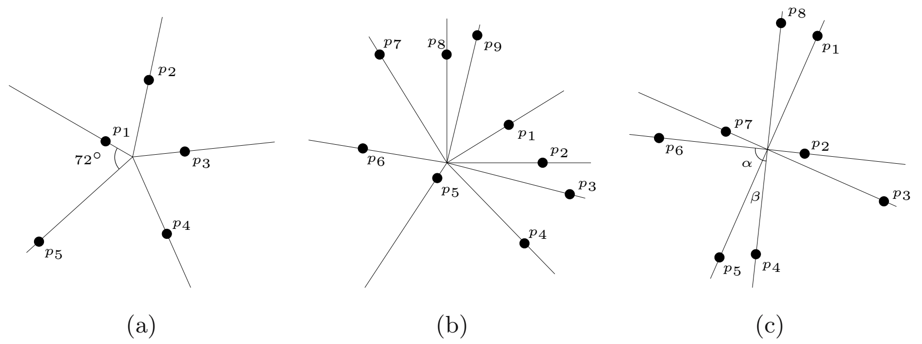


Fig. 1. Example of (a) equiangular configuration with $n = 5$; (b) σ-angular configuration with $n = 9$, where $\sigma = (31°, 45°, 14°, 31°, 49.5°, 68.5°, 76°, 31°, 14°)$; and (c) biangular configuration with $n = 8$.

exists, is a Weber point; thus, there is at most one center of equiangularity for $n \geq 3$ points that are not on a line. Obviously, we could check easily whether a given set of points is in equiangular configuration if we could find their Weber point. Unfortunately, no efficient algorithms are known to find the Weber point in general; even worse, it can be shown that the Weber point cannot even be computed using radicals [2]. Hence, other algorithms are necessary, which we will develop throughout this paper. The algorithm we will present for identifying equiangularity has an interesting implication on Weber points: If n points are in equiangular configuration, then we can use our algorithm to compute their Weber point. This is rather surprising, since such results are known for only few other patterns (e.g. all points are on a line), whereas this does not hold in general for many easy–looking geometric pattern: for instance, it has been shown that it is hard to find the Weber point even if all points are on a circle [5].

A generalization of equiangularity is *σ-angularity*, where we are given a string $\sigma = (\sigma_1, \ldots, \sigma_n)$ of n angles and we ask whether there exists a center c and an ordering of the points such that the sequence of angles between each two adjacent points w.r.t. the center is σ (see Figure 1(c)).Observe that the center of σ-angularity is not necessary unique. Obviously, equiangularity is equivalent to σ-angularity with $\sigma = (\frac{360}{n}, \frac{360}{n}, \ldots, \frac{360}{n})$. The case of two alternating angles α and β, i.e., $\sigma = (\alpha, \beta, \alpha, \ldots, \beta)$, is referred to as *biangular* (see Figure 1(c)).

σ-angular configurations have been applied successfully in robotics, namely in solving the Gathering Problem, which – informally – can be defined as follows: given is a set of autonomous mobile robots that cannot communicate at all and that can only observe the positions of all other robots in the plane. The task is to gather the robots at an arbitrary point in the plane that is not fixed in advance. One of the main difficulties of this problem is to deal with configurations that are totally symmetric, for instance where the robots' positions form a regular[1] n-gon. Recently, an algorithm solving the Gathering Problem has been proposed which uses – among other techniques – the center

[1] Note that a regular n-gon is a special case of equiangular configuration.

of equiangularity or biangularity to gather the robots there [4]. Hence, efficient algorithms are needed to find the centers of such configurations.

In this paper, we present algorithms that decide whether a point set is in σ-angular configuration, for a given string σ, and if so, the algorithms output a corresponding center. To our knowledge, there is no treatment of equiangularity in the vast amount of literature on computational geometry. Only very distantly related, if at all, are for instance star-shaped polygons and star–graphs [7].

For the general case of σ-angularity, we will present in Section 2 an algorithm with running time $O(n^4 \log n)$. For the special cases of biangular and equiangular configurations, this algorithm runs in cubic time, if the two angles are given. In Section 3, we will give another algorithm that allows to detect equiangular configurations even in linear time. All algorithms are straightforward to implement.

2 σ-Angular Configurations

In this section, we present an algorithm that detects σ-angularity in running time $O(n^4 \log n)$, and we show how to simplify this algorithm for biangular and equiangular configurations, yielding running time $O(n^3)$. Our algorithms rely on the notion of *Thales circles*, which we introduce below.

2.1 Thales Circles

Given two points p and q and an angle $0° < \alpha < 180°$, a circle $\mathcal{C}$ is a *Thales circle of angle α for p and q* if p and q are on $\mathcal{C}$, and there is a point x on $\mathcal{C}$ such that $\sphericalangle(p, x, q) = \alpha$, where $\sphericalangle(p, x, q)$ denotes the angle between p and q w.r.t. x. In the following we will denote such a circle also by $\mathcal{C}_{pq}$. An example of a Thales circle[2]can be found in Figure 2(a). It is well-known from basic geometry that all angles on a circle arc are identical, i.e., given a Thales circle $\mathcal{C}$ of angle α for points p and q such that $\sphericalangle(p, x, q) = \alpha$ for some point x on $\mathcal{C}$, then $\sphericalangle(p, x', q) = \alpha$ for every point x' on the same circle arc of $\mathcal{C}$ where x is.

Lemma 1. *Given two points p and q and an angle $0° < \alpha < 180°$, the Thales circles of angle α for p and q can be constructed in constant time.*

The lemma can be proven using basic geometry.

Observe that for two points p and q, there is exactly one Thales circle of angle $90°$, and there are two Thales circles of angle $\alpha \neq 90°$. However, in the remainder of this paper we will often speak of "*the* Thales circle", since it will be clear from the context which of the two Thales circles we refer to.

The connection between Thales circles and σ-angular configurations is the following (see Figure 2(b)): Assume that a point set P is in σ-angular configuration with center c, for some string of angles $\sigma = (\sigma_1, \ldots, \sigma_n)$. For two points $p, q \in P$, let α be the angle between p and q w.r.t. c, i.e., $\alpha = \sphericalangle(p, c, q)$. Then

[2] The name "Thales circle" refers to Thales of Miletus, who was one of the first to show that all angles in a semi-circle have $90°$.

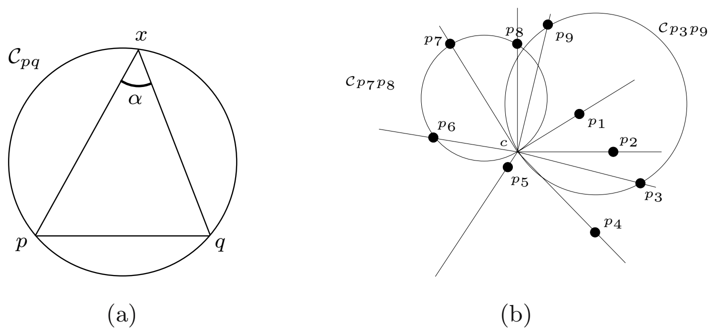


Fig. 2. (a) A Thales circle of angle $\alpha < 90°$. (b) $\mathcal{C}_{p_3p_9}$ is Thales circle for p_3 and p_9 with $\alpha = \sigma_9 + \sigma_1 + \sigma_2$, and $\mathcal{C}_{p_7p_8}$ is Thales circle for p_7 and p_8 with $\alpha = \sigma_7$.

$\alpha = \sum_{k=i}^{j} \sigma_k$ for two appropriate indices i, j (taken modulo n). Let $\mathcal{C}$ be the circle through p, q and c. Then $\mathcal{C}$ is a Thales circle of angle α for p and q. Since this holds for any pair of points from P - with appropriate angles - this yields the following:

Observation 1. *The center of σ-angularity must lie in the intersection of Thales circles for any two points from P of appropriate angle.*

We will use this observation in our algorithms to find candidates for the center of σ-angularity.

2.2 Algorithm for σ-Angular Configurations

We now present an algorithm that detects σ-angular configurations in time $O(n^4 \log n)$; before, we observe basic properties of σ-angular configurations that we apply in our algorithm.

Let σ be a string of angles and P be a point set that is in σ-angular configuration with center c. First, observe that the angles in σ must sum up to 360°; thus, there can be at most one angle in σ that is larger than 180°. Let α_{max} be the maximum angle in σ. Then the following holds (cf. Figure 3): If $\alpha_{max} > 180°$, then center c is outside the convex hull of P; if $\alpha_{max} = 180°$ and there is no other angle of 180° in σ, then center c is on the convex hull; if there are two angles of 180° in σ, then all other angles in σ are of 0°, and the points in P are on a line; and, finally, if $\alpha_{max} < 180°$, then center c is strictly inside the convex hull of P. Furthermore, observe for the last case – where c is inside the convex hull – that the ordering of the points in P that yields σ, if restricted to the points on the convex hull of P, corresponds to the ordering of these points along the convex hull.

Our algorithm to detect σ-angularity will distinguish these four cases. For each case, the algorithm will generate few candidate points that might be a center of σ-angularity. Then we test for each candidate c whether it is indeed a center as follows: We compute the angle between a fixed point on the convex hull, say x, and every other point $p \in P$ w.r.t. c. We sort these angles and compute

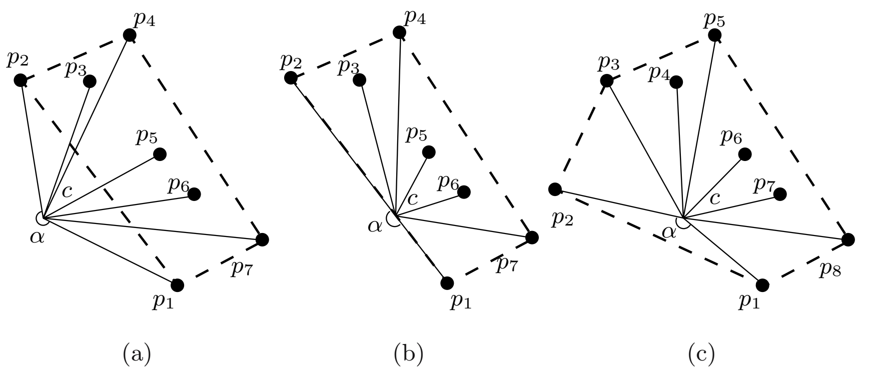


Fig. 3. σ-angular configurations with (a) $\alpha_{max} > 180°$, (b) α_{max} unique angle of $180°$, and (c) $\alpha_{max} < 180°$ (dashed lines show the convex hull of $\{p_1, \ldots, p_n\}$).

sequence τ by subtracting each angle from its successor. Then candidate c is a center of σ-angularity if sequence τ or its reverse is a cyclic shift of σ. This test requires time $O(n \log n)$.

Theorem 1. *Given a point set P of $n \geq 3$ distinct points in the plane and a string $\sigma = (\sigma_1, \ldots, \sigma_n)$ of n angles with $\sum_i \sigma_i = 360°$, there is an algorithm with running time $O(n^4 \log n)$ that decides whether the points are in σ-angular configuration, and if so, the algorithm outputs a center of σ-angularity.*

Proof. The algorithm consists of different routines, depending on whether the largest angle in σ is greater than, equal to, or less than $180°$. We present the algorithm for the case that all angles are less than $180°$. The other cases are solved similarly.

Case All angles less than $180°$*.* In this case, a center of σ-angularity, if it exists, is strictly inside the convex hull of P (see Figure 3(c)). Moreover, if we fix three points from P and compute the Thales circles for these points with appropriate angles, then the center of equiangularity lies in the intersection of these circles (cf. Observation 1). This idea is implemented in Algorithm 1.

To see the correctness of the algorithm, note that α and β are two consecutive range sums of σ' that sum over at most n angles. Thus, the algorithm has a loop for each angle α that might occur between x and y according to σ.

If for some angle α all points from P are on the Thales circle $\mathcal{C}_{xy}$, then α cannot be the angle between x and y, since the center of σ-angularity would have to be both on circle $\mathcal{C}_{xy}$ and inside the convex hull of the points, which is not possible (recall that the center cannot be a point from P by definition).

The running time of Algorithm 1 is $O(n^4 \log n)$. □

We now show how to simplify the previous algorithm to test in only cubic time whether a point set is in biangular configuration, presumed we know the two corresponding angles:

Algorithm 1 Algorithm for all angles less than 180°.

```
1: If the points in $P$ are on a line Then
2:     Return ≪not $\sigma$-angular≫
3: $\sigma' = (\sigma_1, \ldots, \sigma_n, \sigma_1, \ldots, \sigma_n)$
4: $x, y$ = two arbitrary neighbors on convex hull of $P$
5: For $i = 1$ To $n$ Do
6:     For $j = i$ To $i + n - 1$ Do
7:         $\alpha = \sum_{l=i}^{j} \sigma'_l$
8:         $\mathcal{C}_{xy}$ = Thales circle of angle $\alpha$ for $x$ and $y$
9:         If at least one point from $P$ is not on $\mathcal{C}_{xy}$ Then
10:            $z$ = arbitrary point from $P$ that is not on $\mathcal{C}_{xy}$
11:            For $k = j + 1$ TO $j + n$ Do
12:                $\beta = \sum_{l=j+1}^{k} \sigma'_l$
13:                $\mathcal{C}_{xz}$ = Thales circle of angle $\beta$ for $x$ and $z$
14:                If $\mathcal{C}_{xy}$ and $\mathcal{C}_{xz}$ intersect in two points Then
15:                    $c$ = point in intersection that is not $x$
16:                    If $c$ is a center of $\sigma$-angularity Then
17:                        Return ≪$\sigma$-angular with center $c$≫
18:                $\mathcal{C}_{yz}$ = Thales circle of angle $\beta$ for $y$ and $z$
19:                If $\mathcal{C}_{xy}$ and $\mathcal{C}_{yz}$ intersect in two points Then
20:                    $c$ = point in intersection that is not $y$
21:                    If $c$ is a center of $\sigma$-angularity Then
22:                        Return ≪$\sigma$-angular with center $c$≫
23: Return ≪not $\sigma$-angular≫
```

Corollary 1. *For two given angles α and β with $0° \leq \alpha, \beta \leq 180°$, biangular configurations can be detected in time $O(n^3)$.*

Proof. First observe that the special case where either α or β is of 180° is easy to solve: In this case, the other angle has to be 0°, the point set P consists of exactly 4 points, all points are on a line, and the center of biangularity is every point between the two median points in P.

In the following, we assume that $\alpha, \beta < 180°$, and adapt Algorithm 1 from above for this special case: First, we pick two neighbor points x and y on the convex hull of P. If the points in P are in biangular configuration, then the angle between x and y is $\gamma = k \cdot (\alpha + \beta) + \delta$, for some value $k \in \{0, \ldots, \frac{n}{2}\}$ and some angle $\delta \in \{0°, \alpha, \beta\}$. For each of these possibilities, we compute the corresponding Thales circle $\mathcal{C}_{xy}$ of angle γ. Like in Algorithm 1, if for some k and δ all other points are on $\mathcal{C}_{xy}$, then these values cannot be the right choice. Otherwise, we pick a point z that is not on $\mathcal{C}_{xy}$. The angle between y and z is $\gamma' = k' \cdot (\alpha + \beta) + \delta'$, for appropriate values $k' \in \{0, \ldots, \frac{n}{2}\}$ and $\delta' \in \{0°, \alpha, \beta\}$. We then compute all corresponding Thales circles $\mathcal{C}_{yz}$ of angle γ' and check for each of the (at most) two points in the intersection between C_{xy} and C_{yz} whether it is a center of biangularity. The correctness follows from Observation 1.

The running time of this algorithm is $O(n^3)$, instead of $O(n^4 \log n)$ for Algorithm 1. This improvement results from two facts: First, instead of considering all combinations of two consecutive range sums of σ in Algorithm 1, we consider only all combinations of $k(\alpha+\beta)+\delta$ and $k'(\alpha+\beta)+\delta'$, where $k, k' \in \{1, \dots, \frac{n}{2}\}$ and $\delta, \delta' \in \{0°, \alpha, \beta\}$. This reduces the number of executions of the inner loop from $O(n^3)$ to $O(n^2)$.

Second, within the loop, we can test a candidate c in linear time (instead of $O(n \log n)$) as follows: we compute all angles between x and p w.r.t. c for all points $p \in P$. If any of these angles is not of the form $k(\alpha + \beta) + \delta$ with $k \in \{0, \dots, \frac{n}{2}\}$ and $\delta \in \{0°, \alpha, \beta\}$, then c cannot be a center of biangularity. Otherwise, we use three arrays of length $\frac{n}{2}$ to store the points from p, each corresponding to one possible value of δ. More precisely, if the angle between x and p is $k \cdot (\alpha + \beta) + \delta$, then we store point p in position k of the corresponding array. This process resembles a kind of bucket counting sort (see e.g. [6]). The points are in biangular configuration with center c if and only if the following three conditions hold: 1. in the array corresponding to $\delta = 0°$ there is exactly one entry in each position; 2. one of the other two arrays is empty; and 3. in the remaining array there is exactly one entry in each position. □

Observe that if the configuration is biangular, then we can immediately determine the corresponding ordering of the points from the values in the three arrays from the algorithm above. Furthermore, note that the previous corollary yields immediately an algorithm for equiangularity with running time $O(n^3)$, since in this case $\alpha = \beta = \frac{360}{n}$. In the next section we will give an algorithm that identifies equiangular configurations even in linear time.

3 Equiangular Configurations

In this section, we first show that there is at most one center of equiangularity. Then we give a linear time algorithm that detects equiangular configurations.

Lemma 2. *Given a point set P of $n \geq 3$ distinct points in the plane that are in equiangular configuration with center c, the following holds:*

1. *Center c is invariant under straight movement of any of the points towards to or away from c, i.e., the points remain in equiangular configuration with center c.*

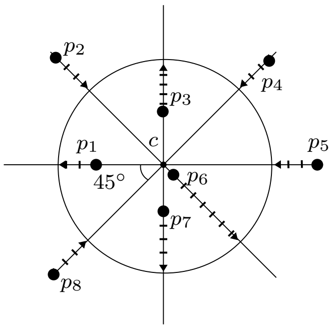


Fig. 4. Equiangular configuration and its “shifted” points

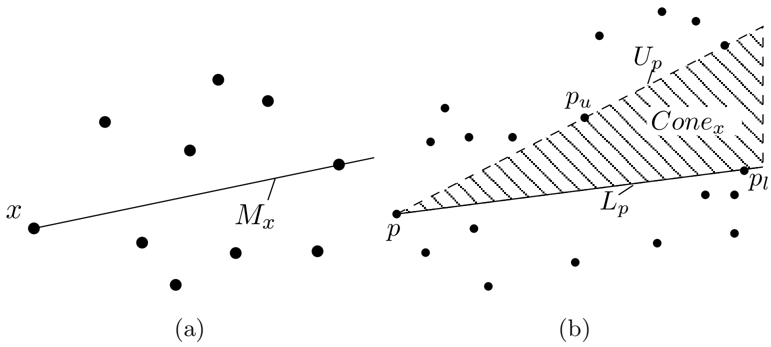


Fig. 5. (a) Median line M_x. (b) Median cone $Cone_x$.

2. Center c is the Weber point of P.
3. The center of equiangularity is unique.

Proof. The first claim is trivial. For the second claim, assume that we move the points in P straight in the direction of c until they occupy the vertices of regular n-gon centered in c (cf. Figure 4). These "shifted" points are rotational symmetric, with symmetry center c. It is straightforward to see that the Weber point for these shifted points is c, since it must be at the center of rotational symmetry. Since the movements were in the direction of c, and the Weber point is invariant under straight movement of any of the points in its direction [1], c must be the Weber point of P. The third claim of the lemma follows immediately from the previous one and from the fact that the Weber point is unique if the points are not collinear [3] (observe that more than two points in equiangular configuration cannot be collinear). □

A similar argument can be used to show that the center of biangularity, if it exists, is unique. We now present an algorithm that identifies equiangular configurations in linear time.

Theorem 2. *Given a point set P of $n \geq 3$ distinct points in the plane, there is an algorithm with running time $O(n)$ that decides whether the points are in equiangular configuration, and if so, the algorithm outputs the center of equiangularity.*

Proof. We handle separately the cases of n even and n odd, where the first case turns out to be much easier. Both algorithms can be divided into two steps: First, they compute (at most) one candidate point for the center of equiangularity; then they check whether this candidate is indeed the center of equiangularity.

1. Case: n is even. The main idea for n even is as follows: assume for a moment that the points in P are in equiangular configuration with center c. Since n is even, for every point $p \in P$ there is exactly one corresponding point $p' \in P$ on the line from p through c (there cannot be more than one point on this line, since this would imply an angle of 0° between these points and p w.r.t. c). Hence, the line through p and p' divides the set of points into two subsets of equal cardinality $\frac{n}{2} - 1$. We will refer to such a line as *median line* (cf. Figure 4(a)). Obviously, center of equiangularity c must lie in the intersection of the median lines of all points in P. We will use this observation in the following algorithm by first computing two median lines and then checking whether their intersection is the center of equiangularity.

Algorithm 2 Algorithm for n even.

```
 1: $x$ = arbitrary point on the convex hull of $P$
 2: $M_x$ = median line of $x$
 3: **If** $|M_x \cap P| > 2$ **Then Return** $\ll$not equiangular$\gg$
 4: $y$ = clockwise neighbor of $x$ on the convex hull of $P$
 5: $M_y$ = median line of $y$
 6: **If** $|M_y \cap P| > 2$ **Then Return** $\ll$not equiangular$\gg$
 7: **If** $M_x \cap M_y = \emptyset$ **Then Return** $\ll$not equiangular$\gg$
 8: $c$ = unique point in $M_x \cap M_y$
 9: **If** $c$ is the center of equiangularity **Then Return** $\ll\sigma$-angular with center $c\gg$
10: **Else Return** $\ll$not equiangular$\gg$
```

Correctness. To see the correctness of Algorithm 2, observe the following:

For a point on the convex hull of P, the median line is unique. (This does not necessarily hold for inner points.) If there are more than two points from P on M_x, then the configuration cannot be equiangular. To see this, recall that the center of equiangularity, if it exists, will lie on M_x. If there are at least three points from P on M_x, then two of them will have angle $0°$ w.r.t. the center, which is impossible for equiangular configurations.

If the two median lines M_x and M_y are equal, then the configuration cannot be equiangular. To see this, first observe that in this case x and y are the only points from P on M_x. Since x and y are adjacent points on the convex hull of P, all points from P are on one side of M_x. On the other hand, since M_x is a median line, by definition the number of points from P on both sides of M_x is equal. Hence, $n = 2$, in contradiction to the assumption of the theorem.

On the other hand, if the two median lines M_x and M_y do not intersect, then the points are not in equiangular configuration. This is obvious, since the center of equiangularity will lie in the intersection of the median lines of all points from P.

Finally, if the points are in equiangular configuration with center c, then c must lie in the intersection of M_x and M_y.

Time Complexity. We now show how to implement each step of Algorithm 2 in linear time.

In order to find point x on the convex hull of P, we could just compute the entire convex hull, but this would take time $\Theta(n \log n)$. Instead, we take as x an arbitrary point from P with a minimal x-coordinate. This is always a point on the convex hull of P and can be found in linear time.

We can find M_x in linear time as follows. First, we compute the slopes of all lines from x to any other point $p \in P - \{x\}$ and store these slopes in an (unsorted) array. Then we pick the line with the median slope. This requires only linear time, since selecting the k-th element of an unsorted array - and consequently the median as well - can be done in linear time [6].

Using the unsorted array from the previous paragraph, we can choose $y \in P$ such that the line through x and y has maximal slope amongst all lines through x and a point in P, using the unsorted array from the previous step. If there is more than one candidate for y, then we take the one closest to x. Then y is a point on the convex hull of P.

Finally, the test whether c is the center of equiangularity can be done in linear time as follows, analogous to the test in the proof of Corollary 1: Let $\alpha = \frac{360}{n}$. We compute all angles between x and p w.r.t. c for all points $p \in P, p \neq x$. If any of these angles is not a multiple of α, then the points are not in equiangular configuration. Otherwise, we store the points from P in an array of length $n-1$, where we store point p in the array at position k if the angle between x and p is $k \cdot \alpha$. This process resembles again a kind of bucket counting sort (see e.g. [6]). If there is exactly one point in each position of the array, then c is the center of equiangularity; otherwise, the points in P are not in equiangular configuration.

2. Case: n is odd. For the odd case, we need to relax the concept of median line, and introduce that of *cone*.

The basic idea of the algorithm for this case is similar to the case "n is even", but slightly more sophisticated, since we have to relax the concept of median lines: assume for a moment that the points are in equiangular configuration with center c. For every point $p \in P$, there is no other point on the line from c to p, since n is odd and no angle of $0°$ can occur. Hence, such a line divides the set of points into two subsets of equal cardinality $\frac{n-1}{2}$. If we pick two points $p_l, p_u \in P$ that are "closest" to this line, in the sense that the slope of the line from p to c is between the slopes of lines L_p and U_p from p to p_l and p_u, respectively, then these two points define a cone with tip p (cf. Figure 4(b)). We will refer to this cone as *median cone*, since the number of points from P on each side of the cone (including lines L_p and U_p, respectively) equals $\frac{n-1}{2}$. Observe that the median cone is unique for points on the convex hull of P, and that the center of equiangularity, if it exists, lies in the intersection of all median cones of points on the convex hull of P. Moreover, for two points x and y on the convex hull of P, every point that is between x and y in the ordering of the points that yields equiangularity is a point in the area "between" $Cone_x$ and $Cone_y$ (bold points in the upper left area in Figure 4(b)). We denote this number by k_{xy}. Notice that the angle between x and y w.r.t. c would be $(k_{xy} + 1) \cdot \alpha$.

The complete algorithm is shown in Algorithm 3, and illustrated in Figure 5. Its main idea is to elect three points x, y, z on the convex hull of P; to use the median cones of these points to determine the angels between the points w.r.t. the center of equiangularity (if it exists); to find one candidate center in the intersection of Thales circles for these points of appropriate angles; and, finally, to check whether this candidate is indeed the center of equiangularity.

Algorithm 3 Algorithm for n odd.

```
1: $x$ = arbitrary point on the convex hull of $P$
2: $Cone_x$ = median cone with tip $x$
3: $y$ = clockwise neighbor of $x$ on the convex hull of $P$
4: $Cone_y$ = median cone of $y$
5: $k_{xy}$ = number of points from $P - \{x, y\}$ that lie between the cones $Cone_x$ and
   $Cone_y$, including the boundaries
6: $\alpha = \frac{360}{n}$
7: $\beta_{xy} = (k_{xy} + 1) \cdot \alpha$
8: $\mathcal{C}_{xy}$ = Thales circle for $x$ and $y$ of angle $\beta_{xy}$
9: $Arc$ = circle arc $\mathcal{C}_{xy} \cap Cone_x \cap Cone_y$
10: $g$ = line through the starting and ending point of $Arc$
11: $H$ = halfplane defined by $g$ that does not contain $x$ and $y$
12: $S = H \cap P$
13: **If** $S = \emptyset$ **Then Return** ≪not equiangular≫
14: $z$ = point from $S$ with maximal perpendicular distance from $g$ over all points in $P$
15: **If** $z \in \mathcal{C}_{xy}$ **Then Return** ≪not equiangular≫
16: $k_{yz}$ = number of points from $P - \{x, y, z\}$ that lie between the cones $Cone_y$ and
    $Cone_z$, including the boundaries
17: $\beta_{yz} = (k_{yz} + 1) \cdot \alpha$
18: $\mathcal{C}_{yz}$ = Thales circle for $y$ and $z$ of angle $\beta_{yz}$
19: **If** $|\mathcal{C}_{xy} \cap \mathcal{C}_{yz}| = 1$ **Then Return** ≪not equiangular≫
20: $c$ = unique point in $\mathcal{C}_{xy} \cap \mathcal{C}_{yz} - \{y\}$
21: **If** $c$ is the center of equiangularity **Then Return** ≪$\sigma$-angular with center $c$≫
22: **Else Return** ≪not equiangular≫
```

Correctness. To see the correctness of Algorithm 3, observe the following:

By definition, no point from P can be strictly inside cones $Cone_x$ or $Cone_y$.

The center of equiangularity, if it exists, is inside the convex hull of the points. Thus, it is straightforward which of the two Thales circles for x and y is appropriate, since x and y are points on the convex hull. The same holds for the Thales circle for y and z.

After picking x and y, we need to find another point z on the convex hull such that the intersection of the two corresponding Thales circles yields a candidate for the center of equiangularity. For the choice of z we have to be careful: if we simply took z as the next point on the convex hull (after x and y), it might happen that the corresponding Thales circle $\mathcal{C}_{yz}$ coincides with $\mathcal{C}_{xy}$, and, consequently, we do not obtain a single candidate for the center of equiangularity in their intersection. Therefore, we have to chose z such that it is on the convex hull and such that it does not lie on $\mathcal{C}_{xy}$. This is done in Lines 9–15, and discussed in the following (see Figure 5(b)).

If S is empty, then the points are not in equiangular configuration. To see this, observe that the center of equiangularity, if it exists, must be on Arc, with Arc as computed in Line 9. On the other hand, if $S = \emptyset$, then Arc is completely outside the convex hull of P, in contradiction to the fact that the center of equiangularity must be inside the convex hull.

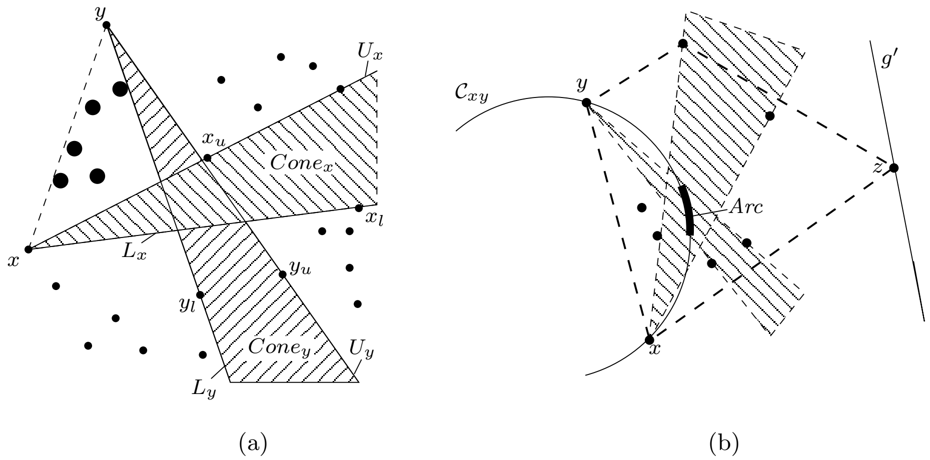


Fig. 6. (a) Determination of $k_{xy} = 5$ between U_x and L_y. (b) Determination of g'.

Point z computed in Line 15 is on the convex hull of P: since, by construction, all points from P are on one side of the line through z that is parallel to g (line g' in Figure 5(b)).

If point z is on $\mathcal{C}_{xy}$, then the points in P are not in equiangular configuration. To see this, assume that $z \in \mathcal{C}_{xy}$. Then $z \in Arc$ by construction, hence, $z \in Cone_x \cap Cone_y$. By definition, no point from P can be strictly inside any median cone; thus, z has to be at one of the two endpoints of Arc. Moreover, no point from P can be strictly inside H, since H is delimited by g, and z is the point from P furthest away from g. This implies that points on Arc are on or outside the convex hull of P. On the other hand, only points on Arc might be a center of equiangularity, since such a center must lie in the intersection of $Cone_x, Cone_y$ and $\mathcal{C}_{xy}$. Since the center of equiangularity, if it exists, must be strictly inside the convex hull of P (otherwise there would be an angle of 180°), the points cannot be in equiangular configuration.

The two circles $\mathcal{C}_{xy}$ and $\mathcal{C}_{yz}$ are different, since $z \notin \mathcal{C}_{xy}$. Thus, they either intersect in one or two points. If $|\mathcal{C}_{xy} \cap \mathcal{C}_{yz}| = 1$, then y is the unique point in the intersection, and the points in P cannot be in equiangular configuration. On the other hand, if the center of equiangularity exists, then it is in $\mathcal{C}_{xy} \cap \mathcal{C}_{yz}$, and thus found in Line 20.

Time Complexity. We now show how to implement each step of Algorithm 2 in linear time.

Points x and y can be found in linear time like in the algorithm for the case "n is even".

To find the median cone $Cone_x$, we proceed analogous to the search for the median line in algorithm for the case "n is even": First we compute the slopes of all lines from x to every other point $p \in P$, and store them in an (unsorted) array. Let L_x and U_x be the two lines such that their slope is the $\lfloor n/2 \rfloor$-th and the $\lceil n/2 \rceil$-th largest, respectively, among all stored slopes. These two lines define $Cone_x$, and can be found in linear time using like before an algorithm to select the k-th element of an unsorted list [6]. Analogous, we can find $Cone_y$.

To determine value k_{xy}, let p be the point in the intersection of L_y and U_x, the two lines that delimit cones $Cone_x$ and $Cone_y$ (cf. Figure **??**(a)). Then k_{xy} is the number of points from $P - \{x, y\}$ that lie inside or on the convex angle in p with edges L_y and U_x. This number can be obtained in linear time by comparing the position of every point in P against L_y and U_x. Value k_{yz} can be found analogous.

Finally, the test whether c is the center of equiangularity can be done in linear time like in the algorithm for the case "n is even". □

4 Conclusions

We have given an algorithm that decides in time $O(n^4 \log n)$ whether n points are in σ-angular configuration, and if so, the algorithm outputs a center of σ-angularity. The adaption of this algorithm for biangular configurations runs in cubic time, if the corresponding two angles are given. Finally, for the case of equiangularity, we have given an algorithm that runs even in time $O(n)$.

While the algorithm for equiangularity is already asymptotically optimal, we believe that the running time of the algorithm for σ-angularity allows to be significantly improved.

As already stated, our algorithm for equiangularity allows to find the Weber point for such configurations, and this is a rather surprising result, since algorithms to compute the Weber point (in finite time) are known for only few other patterns. We are not aware of any general characterization of patterns where the Weber point is easy to find; this might be an interesting line of future research.

References

1. L. Anderegg, M. Cieliebak, G. Prencipe, and P. Widmayer. When is Weber Point Invariant? *(Manuscript)*.
2. C. Bajaj. The Algebraic Degree of Geometric Optimization Problem. *Discrete and Computational Geometry*, 3:177–191, 1988.
3. R. Chandrasekaran and A. Tamir. Algebraic optimization: The Fermat-Weber location problem. *Mathematical Programming*, 46:219–224, 1990.
4. M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *Proceedings of ICALP 2003*, pages 1181–1196, 2003.
5. E. J. Cockayne and Z.A. Melzak. Euclidean constructibility in graph-minimization problems. *Math. Magazine*, 42:206 – 208, 1969.
6. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
7. J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry.* CRC Press LLC, Boca Raton, FL, 1997.
8. E. Weiszfeld. Sur le Point Pour Lequel la Somme Des Distances de n Points Donnés Est Minimum. *Tohoku Mathematical*, 43:355–386, 1936.

Pickup and Delivery for Moving Objects on Broken Lines

Yuichi Asahiro[1,⋆], Eiji Miyano[2,⋆⋆], and Shinichi Shimoirisa[3]

[1] Department of Social Information Systems, Kyushu Sangyo University, Fukuoka 813-8503, Japan
asahiro@is.kyusan-u.ac.jp

[2] Department of Systems Innovation and Informatics, Kyushu Institute of Technology, Fukuoka 820-8502, Japan
miyano@ces.kyutech.ac.jp

[3] Products Development Center, Toshiba Tec Corporation, Japan

Abstract. We consider the following variant of the Vehicle Routing Problem that we call the *Pickup and Delivery for Moving Objects (PDMO)* problem, motivated by robot navigation: The input to the problem consists of n products, each of which moves on a predefined path with a fixed constant speed, and a robot arm of capacity one. In each round, the robot arm grasps and delivers one product to its original position. The goal of the problem is to find a collection of tours such that the robot arm grasps and delivers as many products as possible. In this paper we prove the following results: (i) If the products move on broken lines with at least one bend, then the PDMO is MAXSNP-hard, and (ii) it can be approximated with ratio two. However, (iii) if we impose the "straight line without bend" restriction on the motion of every product, then the PDMO becomes tractable.

1 Introduction

The *Vehicle Routing Problem* (VRP for short) or *Vehicle Scheduling Problem* is one of the best known routing problems (see for a survey [1] and the references contained therein). In the VRP we have scarce capacitated vehicles, their home depot, and a large number of customers who request the vehicles to transport their products to the depot. A typical VRP can be described as the problem of designing as short routes from the depot to the customers as possible in such a way that each customer is visited only once by exactly one vehicle, all routes start and end at the depot, the total demand of all customers on one particular route cannot exceed the capacity of the vehicle, and all the products have to be collected to the depot. The VRP has received considerable attention especially in the operations research literature for its wide applicability to practical situations and thus a lot of its variants have been introduced, such as the VRP with Time

⋆ Supported in part by Grant-in-Aid for Young Scientists 15700021, and Research on Priority Areas 16092223.

⋆⋆ Supported in part by Grant-in-Aid for Research on Priority Areas 16092223.

Windows (VRPTW) and the VRP with Backhauls (VRPB) (see, e.g., [2,3,4]). In the VRPTW, each customer has a release time and a deadline, and the goal is to visit as many customers as possible within their "time-windows." The VRPB involves both a set of customers to whom products are to be delivered and a set of vendors whose products need to be transported back to the depot, and the critical assumption is that all deliveries must be made on each route before any pickups can be made.

In this paper, we consider a new extension of the classical VRP motivated by a real-world routing situation, called the *Pickup and Delivery for Moving Objects (PDMO)* problem: Given manufacture products initially located on numbers of convey belts in arbitrary positions, the robot arm must grasp and deliver the products to the fixed position (depot). That is, the input to the problem consists of n objects (products), each of which moves on a predefined broken-line path with a fixed constant speed, and a robot arm of capacity one. In each round, the robot arm grasps and delivers one object to the depot. If some object can move faster than the robot in the opposite direction from its current position, then the robot is not able to collect the object. Therefore, the goal of the problem is to find a collection of tours such that the robot arm grasps and delivers as many objects as possible.

1.1 Our Contributions

Since the original VRP and its variants are not only practically but also theoretically interesting, and not at all easy to solve, it is important to study the PDMO from both the complexity and the algorithmic viewpoints. Hence it follows that a fundamental question is the complexity status of the PDMO, especially with respect to approximation. In this paper, we prove that the PDMO is MAXSNP-hard in the general case, and moreover it is still true for the case where all objects move on broken lines with only one bend. That is, unfortunately, we cannot obtain a polynomial-time approximation scheme (PTAS) for the PDMO under the assumption $\mathcal{P} \neq \mathcal{NP}$ even if every motion of the object is highly restricted. On the other hand, we provide a simple greedy algorithm that obtains a solution with a value of at least $\frac{1}{2}$ times the value of an optimal solution. Finally, as a straightforward step towards tackling this intractable problem, we slightly impose one restriction on the motion of every object; we assume that every object moves on an individual straight line with no bend. Then, under this assumption, we show that there is an $O(n \log n)$-time algorithm which maximizes the number of objects collected by a robot arm.

1.2 Related Work

The VRP includes the *Traveling Salesperson Problem* (TSP) as a special case, i.e., only one vehicle is available at the depot and no additional operational constraints are imposed, which is probably the best important in the field of complexity theory and combinatorial optimization (see, e.g., the classical book edited by Lawler, Lenstra, Rinnooy Kan, and Shmoys [5]). Similarly to the problem of this paper, very recently, *kinetic* versions of the TSP, called the *Dynamic* *k-Collect TSP* problem [6], the *Moving-Target TSP* problem [7], or the *Kinetic TSP* problem [8], have been introduced: Given n objects, each j moving with a constant speed v_j, and a robot arm R with capacity k and with the maximum speed $v_R > v_j$ for all j's, we ask for the robot arm's tour that collects all the n moving objects and whose length is as short as possible. It is important to note that the goal of the problem is to minimize the *total completion time*, not to maximize the so-called *profit*. Chalasani, Motwani, and Rao [6] proved that the problem can be approximated in polynomial time within a constant factor if $k = \infty$, and can be solved in polynomial time if $k = 1$ and the robot and the objects move on a single track-line. Helvig, Robins, and Zelikovsky [7] showed that there is a $(1+\alpha)$-approximation algorithm if $k = \infty$ and at most $O(\frac{\log n}{\log \log n})$ objects are moving, where α denotes an approximation factor of the best heuristic for the classical TSP. In [8] Hammar and Nilsson showed that there is a PTAS if $k = \infty$ and all objects have the same speed. Although there hardly appears to be any prior work on the profit maximization, Asahiro, Horiyama, Makino, Ono, Sakuma, and Yamashita [9] provided a couple of first-step results for several special cases, and Asahiro, Miyano and Shimoirisa [10] for the time constrained version; only a few approximable results are known for a very restricted case, and non-approximable results have not been shown.

One can notice that there is a close connection between the PDMO and some class of time constrained job scheduling problems, such as the *sequencing problems* [11,12] and the *job interval selection problems* [13]. In the job scheduling model, the input is a set of jobs, each of which has its constant processing time. The objective is to schedule as many jobs as possible to a single machine (or, several machines). Note, however, that in the PDMO the processing time for each moving object, i.e., the period between the robot's departure time and its arrival time at the depot is variable, which makes the PDMO harder. The job scheduling model also contains a large number of intractable cases, and hence a great deal of effort has been devoted to the design of approximation algorithms with small constant performance ratios [14,15,16].

1.3 Organization

The rest of this paper is organized as follows: We begin with more precise definitions of our problems and then summarize our results in Sect. 2. Then we give a 2 factor approximation algorithm for the general problem, followed by a polynomial time algorithm for moving objects on straight lines in Sections 3 and 4, respectively. In Sect. 5, we prove the MAXSNP-hardness of the PDMO. Finally, we conclude in Sect. 6.

2 Problems and Results

Let $B = \{b_1, b_2, \cdots, b_n\}$ be a set of *moving objects* in the xy-plane with the Euclidean metric. The speed of each object b_i is $v(b_i)$. As illustrated in Fig. 1, a path of each object b_i is a broken line given as a sequence of line segments, or vertices labeled as $C_0(b_i) = (x_0(b_i), y_0(b_i)), C_1(b_i) = (x_1(b_i), y_1(b_i))$ through

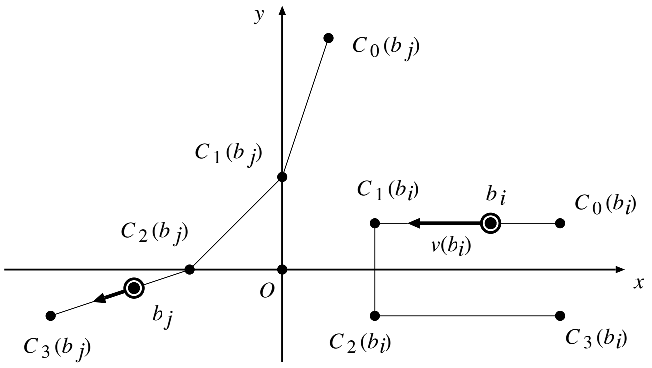


Fig. 1. Moving objects and their paths

$C_{m(b_i)+1} = (x_{m(b_i)+1}(b_i), y_{m(b_i)+1}(b_i))$. That is, the broken line has $m(b_i)$ bends. Also, let $m = \max_{b_i \in B} m(b_i)$. Every b_i starts from $C_0(b_i)$ at time 0 and moves towards the next point $C_1(b_i)$ with speed $v(b_i)$. Then, at time $|\overline{C_1(b_i)C_0(b_i)}|/v(b_i)$, b_i arrives at $C_1(b_i)$, changes the direction there, and moves towards $C_2(b_i)$, and so on.

We now can formulate the problem of this paper, *Pickup and Delivery for Moving Objects on Broken Lines (PDMO)*, as follows:

PDMO:
Instance: A set $B = \{b_1, b_2, \cdots, b_n\}$ of moving objects and their paths, where each b_i is initially placed on $C_0(b_i)$.
Question: Find a longest scheduling of a robot arm R with speed v_R, that is represented as a sequence of triples (s_1, b_{i_1}, g_1), (s_2, b_{i_2}, g_2), $\cdots$, for $i_1, i_2, \cdots \in \{1, \cdots, n\}$, such that for any j, k, $b_{i_j} \neq b_{i_k}$ and each of (s_j, b_{i_j}, g_j)'s satisfies the following three conditions:
(i) R is at the origin O at time s_j.
(ii) (Pickup) R can start from O at time s_j, move onto the path of b_{i_j}, and then grasp b_{i_j}. Namely, b_{i_j} arrives at the point where R is waiting for b_{i_j} to come.
(iii) (Delivery) R can move back to O at time g_j, and $g_j \leq s_{j+1}$ (except for the last one).

We simply call a turn of pickup and delivery of an object, *collect* the object. A scheduling is *valid* if the three conditions (i), (ii), and (iii) are satisfied. We assume that the robot arm can stop only at the origin and makes no detour to move to any position, because a general scheduling can be transformed easily to such a scheduling without decreasing the number of objects to be collected. This assumption yields that g_j is uniquely determined by s_j and b_{i_j}. If it is not ambiguous in the context, we often abbreviate, for example, $C_0(b_i)$ and $v(b_i)$ as C_0 and v, respectively. Without loss of generality we set $v_R = 1$. Since an object whose speed is slower than v_R can be collected at any time, we focus only on the case that all the objects move with faster or same speeds compared to the robot arm. We say an algorithm ALG is an r factor approximation algorithm for a maximization problem if for any instance I, $\mathrm{OPT}(I)/\mathrm{ALG}(I)$ is bounded above by r, where $\mathrm{OPT}(I)$ is the objective function value of an optimal solution for I and $\mathrm{ALG}(I)$ is that of a solution ALG obtains.

In this paper we mainly prove the following three theorems:

Theorem 1. *The PDMO is MAXSNP-hard even if $m(b_i) = 1$ for all i's.*

Theorem 2. *There exists a 2 factor approximation algorithm for the (general) PDMO that runs in $O(mn^2)$.*

Theorem 3. *If $\forall i, m(b_i) = 0$ (i.e., all the objects move on straight lines), then there exists an $O(n \log n)$-time algorithm that maximizes the total number of the moving objects collected by a robot arm.*

In the following sections, we shall give the proofs of Theorems 2, 3 and 1, in that order.

3 A 2-Approximation Algorithm

In this section, we present the 2 factor approximation algorithm of Theorem 2, but before describing it, we first show a lemma. Let the *completion time* $T_i(t)$ of an object b_i be the smallest t_d where the robot arm starts at time t or later from the origin O, grasps the object b_i, and delivers it at time t_d back to O.

Lemma 1. *For any object b_i and time t, $T_i(t)$ can be computed in $O(m)$ time.*

Proof. Consider an object b with speed $v > 1$ $(= v_R)$. First we argue only about the first line segment of its path. Let the first two vertices of its path be $C_0 = (x, y)$ and $C_1 = (x_1, y)$ ($x_1 < x$ and $y > 0$), i.e., b horizontally moves left. Note that when we focus on one object, we can adjust the axes to meet this condition by rotation and symmetry in $O(1)$ time. If b can be collected on this segment, $x > 0$ holds; otherwise, b cannot be collected since in the case where $x \leq 0$, it moves away from the origin and is faster than the robot arm.

Consider a point $M = (ex, y)$, where $x_1/x \leq e \leq 1$. If the robot arm R grasps b on M and delivers it to the origin O, then the time $t(e)$ that the robot arm delivers b is given as summation of time that the object reaches M and that the robot arm moves back to the origin from M:

$$t(e) = \frac{(1-e)x}{v} + \sqrt{y^2 + e^2x^2}.$$

This takes minimum by setting

$$e = \frac{y}{x\sqrt{v^2-1}},$$

and hence the point $M_{\min}$ on which the robot arm grasps b is

$$M_{\min} = (\frac{y}{\sqrt{v^2-1}}, y). \tag{1}$$

Then the minimum completion time $T_{\min}$ that the robot arm grasps b on $\overline{C_0C_1}$ is obtained as

$$T_{\min} = \frac{x}{v} - \frac{y}{v\sqrt{v^2-1}} + \frac{vy}{\sqrt{v^2-1}}.$$

Also, b arrives at $M_{\min}$ at time

$$t_{\min} = \frac{(1-e)x}{v} = \frac{x}{v} - \frac{y}{v\sqrt{v^2-1}}$$

and the distance $|\overline{OM_{\min}}|$ is equal to $vy/\sqrt{v^2-1}$. Therefore, to minimize the completion time, R starts at time

$$t_0 = t_{\min} - \frac{vy}{\sqrt{v^2-1}}$$

from O (if it can), moves to $M_{\min}$, grasps b, and finally returns back to O. However, since the above t_0 depends on the location (x,y) of C_0, it is possibly to be negative that implies that the robot arm cannot start from the origin at t_0. Therefore, in summary, if $x > 0$, $t_0 \geq 0$ and $0 \leq t \leq t_0$, the robot arm R can accomplish the motion and then $T_i(t) = T_{\min}$ holds since R can stay at O until time t_0, moves to the point $M_{\min}$, grasps and eventually delivers b.

For the other case $t_0 < 0$ or $t > t_0$, suppose that R starts at time t from O and grasps b on $(x - vt'(v,x,y,t), y)$ at time $t'(v,x,y,t) \leq |\overline{C_0C_1}|/v = (x-x_1)/v$. Let t_1 in the following satisfy $t_1 + |\overline{OC_1}| = (x-x_1)/v$. The time T that R delivers b is

$$t'(v,x,y,t) + \sqrt{(x - vt'(v,x,y,t))^2 + y^2}.$$

Since $t'(v,x,y,t) = t + \sqrt{(x - vt'(v,x,y,t))^2 + y^2}$,

$$T = t + 2\sqrt{(x - vt'(v,x,y,t))^2 + y^2}, \quad \text{and}$$
$$t'(v,x,y,t) = \frac{(vx-t)^2 + \sqrt{(vx-t)^2 - (v^2-1)(x^2+y^2-t^2)}}{v^2-1}.$$

This motion can be done when $(vx-t)^2 - (v^2-1)(x^2+y^2-t^2) \geq 0$, that derives

$$\frac{x - \sqrt{2x^2-(v^2-1)y^2}}{v} \ (\overset{\text{def}}{=} t_2) \ \leq t \leq \frac{x + \sqrt{2x^2-(v^2-1)y^2}}{v} \quad (\overset{\text{def}}{=} t_3).$$

As a result,

$$T(t) = \begin{cases} \dfrac{x}{v} - \dfrac{y}{v\sqrt{v^2-1}} + \dfrac{vy}{\sqrt{v^2-1}}, \ \text{if } x > 0, t_0 \geq 0, \ \text{and } 0 \leq t \leq t_0, \\ t + 2\sqrt{(x - vt'(v,x,y,t))^2 + y^2}, \\ \qquad\qquad \text{if } x > 0 \text{ and } \max\{t_0, t_2\} < t \leq \min\{t_1, t_3\}, \ \text{and} \\ \text{undefined, otherwise.} \end{cases}$$

All the above $t_0, t_1, t_2, t_3, t'(v, x, y, t)$ and so $T(t)$'s can be computed from x, x_1, y, v and t in $O(1)$ time. By "undefined" we mean that the object cannot be collected on the line segment in that situation.

Now we look at a line segment $\overline{C_j C_{j+1}}$ of the path ($j \leq m(b)$). Similarly we can compute the minimum completion time $T^{(j)}(t)$ with regard to $\overline{C_j C_{j+1}}$ in $O(1)$ time: Let the time that b arrives at C_j be

$$s_j = \sum_{k=0}^{j-1} \frac{|\overline{C_k C_{k+1}}|}{v}.$$

Only by replacing the above symbols, e.g., x with x_j, one can see that the following holds:

$$T^{(j)}(t) = \begin{cases} \frac{x_j}{v} - \frac{y_j}{v\sqrt{v^2-1}} + \frac{vy_j}{\sqrt{v^2-1}}, \text{ if } x_j > 0, t_0 \geq 0, \text{ and } 0 \leq t \leq s_j + t_0 \\ t + 2\sqrt{(x_j - vt'(v, x_j, y_j, t))^2 + y_j^2}, \\ \qquad \text{if } x_j > 0 \text{ and } s_j + \max\{t_0, t_2\} < t \leq s_j + \min\{t_1, t_3\}, \text{ and} \\ \text{undefined, otherwise.} \end{cases}$$

Since the domain of $T^{(j)}(t)$ for $\overline{C_j C_{j+1}}$ overlaps the others, to obtain the true completion time at t, we have to compute the above $T^{(j)}(t)$'s for all the line segments $\overline{C_j C_{j+1}}$'s, and then select the minimum value among them, which requires $O(m)$ time.

As for the case $v(b) = v_R$, similar but rather simpler discussion can be done that proves the statement of the lemma. We omit it here. □

The following is a description of the algorithm. The operator $\circ$ denotes concatenation of a sequence and a triple which represents a tour of collecting an object.

```
Algorithm Iterated_Greedy:
Stage 0. Set the current time $t = 0$, and initial scheduling $\mathcal{S}$ = NULL.
Stage 1. The following operations are executed while some of objects are possible to be collected, i.e., the completion time for an object is defined.
    1-1. Set $j = \text{argmin}_i\{T_i(t)\}$.
    1-2. Grasp and deliver $b_j$: $\mathcal{S} = \mathcal{S} \circ (t, b_j, T_j(t))$
    1-3. Update $t = T_j(t)$, and return to the beginning of Stage 1.
Stage 2. Output $\mathcal{S}$ and exit.
```

Proof of Theorem 2. First of all, we estimate the running time of the above Iterated_Greedy. From Lemma 1, **Stage 1-1** is executed in $O(mn)$ time since each completion time $T_i(t)$ for b_i is obtained in $O(m)$ time and the number of objects is n. **Stages 1-2** and **1-3** can be done in $O(1)$ time. Since the number of iterations of **Stage 1** is at most n and **Stage 2** is done in $O(n)$ time, Iterated_Greedy runs in $O(mn^2)$ time.

Next we prove that `Iterated_Greedy` is a 2 factor approximation algorithm. Suppose that given an instance, `Iterated_Greedy` collects p moving objects, $u_1, u_2, \cdots, u_p$, in this order. Also, suppose that for the same instance an optimal algorithm OPT collects $v_1, v_2, \cdots, v_q$ $(q > p)$ in this order. Let the scheduling $\mathcal{S}_A$ of `Iterated_Greedy` be $(s_1(A), u_1, g_1(A))$, $(s_2(A), u_2, g_2(A))$, $\cdots$, $(s_p(A), u_p, g_p(A))$ and also let $\mathcal{S}_{OPT}$ of OPT be $(s_1(OPT), v_1, g_1(OPT))$, $(s_2(OPT), v_2, g_2(OPT))$, $\cdots$, $(s_q(OPT), v_q, g_q(OPT))$. We divide the time interval $[0, T_m]$, where $T_m = \max\{g_p(A), g_q(OPT)\}$, into $[0, g_1(A))$, $[g_1(A), g_2(A))$, $\cdots$, $[g_p(A), T_m]$.

Consider the first interval $[0, g_1(A))$. Since `Iterated_Greedy` first collects an object whose completion time is minimum, in this interval, OPT delivers no objects and $g_1(OPT) \geq g_1(A)$ holds. In the second interval $[g_1(A), g_2(A))$, `Iterated_Greedy` delivers one object u_1. Assume that $g_2(OPT) < g_2(A)$, i.e., OPT delivers two objects v_1 and v_2 in this interval. If $v_2 \in B - \{u_1, v_1\}$, then this contradicts the fact that $g_2(A)$ is the minimum among the completion times of $B - \{u_1\}$, since $g_1(A) \leq g_1(OPT) \leq s_2(OPT) < g_2(OPT) < g_2(A)$ that implies the completion time of v_2 is less than that of u_2. Therefore, in this case, v_2 is only possible to be identical to u_1. Therefore within the interval $[g_1(A), g_2(A))$ in which A delivers one object, OPT can deliver at most two objects. Similarly for the other interval $[g_i(A), g_{i+1}(A))$, if the number of objects OPT collects in this interval is more than one, then the additional objects collected belong to $\{u_1, \cdots, u_i\}$, because u_{i+1}'s completion time is minimum among $B - \{u_1, \cdots, u_i\}$ at time $g_i(A)$. By induction, the number of objects OPT collects in $[0, g_{i+1}(A))$ is at most $2i$, while A collects i objects $u_1, \cdots, u_i$ in this interval, in which the increase i comes from $\{u_1, \cdots, u_i\}$. This concludes the proof. $\square$

Although the above analysis is rather simple, the factor 2 is best possible for `Iterated_Greedy`; there is a tight example that `Iterated_Greedy` can collect only half of objects that an optimal solution does.

Proposition 1. *There exists an instance for which the number of objects collected by* `Iterated_Greedy` *is 1/2 of an optimal solution.*

Proof. See Fig. 2. The initial points of objects b_1 and b_2 are respectively (l, l) and $(l + \epsilon, -(l + \epsilon))$, where l is some constant and ϵ is small positive. The angle $\angle C_1$ is set to, say, $60°$. Suppose that the speeds of the objects and the robot arm are all 1. Since the minimum completion time $2l$ is obtained by grasping b_1 on the point D at time l, `Iterated_Greedy` collects b_1 first. However, at time $2l$ or later, b_2 cannot be collected, so that the number of objects collected by `Iterated_Greedy` is one. On the other hand, the optimal solution is collecting b_2 and b_1 in this order. When b_2 is delivered at time $2l + 2\epsilon$, b_1 already passed through D but there is another chance to collect it on the point E at time $(1 + 2\sqrt{3})l$. Therefore, the optimal number of objects to be collected is two, and is twice of that by `Iterated_Greedy`. $\square$

This worst case example intuitively shows the intractability of the problem, and is based on the proof of MAXSNP-hardness in Sect. 5. On the other hand,

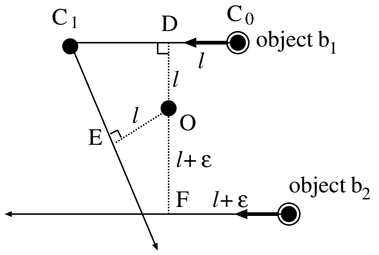


Fig. 2. A worst case example for `Iterated_Greedy`

we can obtain optimal solutions if all the paths of objects are straight lines, which is shown in the next section.

4 An $O(n \log n)$-Time Algorithm for No Bend

Assuming that all the paths of the objects are straight lines, we shall give an algorithm mentioned in Theorem 3. At first, we show the following lemma on `Iterated_Greedy` in this setting:

Lemma 2. `Iterated_Greedy` *obtains an optimal solution when all the paths of objects are lines.*

Proof. Look again at the discussion of the proof of Theorem 2 in the previous section. The main reason why the approximation ratio rose up to two was that, say, in the interval $[g_1(A), g_2(A))$, OPT could collect an object u_1 after the time $g_1(A)$ at which `Iterated_Greedy` delivered u_1. However, if u_1 moves on a straight line, this cannot happen: Recall that we assume the speed v of an object is faster than or equal to the speed $v_R(=1)$ of the robot arm. We explain the situation by using Fig. 2 again. Suppose that `Iterated_Greedy` grasps b_2 on a point Z (not shown in Fig. 2) at time t. Then, the time `Iterated_Greedy` delivered b_2 is $t + |\overline{OZ}|/v_R$. $|\overline{OZ}|/v_R > |\overline{ZF}|/v$ holds since $|\overline{OZ}| > |\overline{ZF}|$ and $v_R \leq v$. Let us consider any algorithm that has not collected b_2 until time $t + |\overline{OZ}|$. Such an algorithm cannot collect b_2 starting from O at time $t + |\overline{OZ}|$ or later, whatever $|\overline{OZ}|$ is, since at that time b_2 passed through F and the speed v_R of the robot arm is less than or equal to that of b_2 in this situation. This implies that any algorithm can collect at most one object in a divided interval described in the proof of Theorem 2, except the first interval $[0, g_1(A))$ that no algorithm can collect any object. Therefore the number of objects collected by `Iterated_Greedy` is optimal. □

`Iterated_Greedy` calculates all the completion times and selects the minimum one iteratively in Stage 1. However, we show that by obtaining once the *completion time functions* $T_i(t)$'s of all the objects for an arbitrary time t at the beginning and by using the *lower envelope* of those functions, the running time can be improved. Now, let $\mathcal{T} = \{T_1(t), T_2(t), \cdots, T_n(t)\}$ be a collection of completion time functions. The *lower envelope* of $\mathcal{T}$ is now defined as $E_{\mathcal{T}}(t) = \min_{1 \leq i \leq n} T_i(t)$.

```
Algorithm Envelope:
Stage 1. For each object $b_i$, compute its completion time function $T_i(t)$ for an
    arbitrary time $t$.
Stage 2. Compute the lower envelope $E_{\mathcal{T}}(t)$ of $\mathcal{T} = \{T_1(t), T_2(t), \cdots, T_n(t)\}$.
Stage 3. Set $t = 0$ and $\mathcal{S}$ =NULL initially.
Stage 4. Execute the following operations while the lower envelope is defined
    at time $t$:
    4-1. Select the lowest function $T_j(t)$ at time $t$ by using the lower envelope
        $E_{\mathcal{T}}(t)$. Then, grasp and deliver the object $b_j$: $\mathcal{S} = \mathcal{S} \circ (t, b_j, T_j(t))$.
    4-2. Update $t = T_j(t)$ and return to the beginning of Stage 4.
Stage 5. Output $\mathcal{S}$ and exit.
```

Proof of Theorem 3. First we shall note the correctness of the algorithm `Envelope`. This algorithm essentially executes the same greedy strategy used in `Iterated_Greedy`, and so runs correctly. Next, we show that `Envelope` runs in $O(n \log n)$ time. **Stage 1** needs $O(n)$ time based on Lemma 1, since only one segment exists for each object and so its completion time function can be obtained in $O(1)$ time. In **Stage 2**, the lower envelope is obtained by a divide-and-conquer method [17]. Its running time is dependent of the number of intersections between a pair of completion time functions. Since all the objects move on straight lines at constant speed, a completion time function intersects only once with the other completion time function, so that the required time to obtain the lower envelope is $O(n \log n)$. **Stages 3** and **5** are done in $O(1)$ time and $O(n)$ time, respectively.

The rest is analysis for **Stage 4** that takes $O(n)$ time in an amortized sense: The lower envelope is assumed to be given as pairs of an time interval and an index that indicates which object yields the minimum completion time within the interval. By definition of the paths of the objects, each completion time function for an object appears only once as a part of the lower envelope, hence the number of the pairs of time interval and an index is at most n. Suppose that at an iteration of **Stage 4**, the current time is t and b_k is the one whose completion time is minimum among all. Then as stated above in the proof of Lemma 2, for time $T_k(t)$ which is used as t for the next iteration, the completion time for the object b_k is undefined. Therefore, each time-interval of the lower envelope is under consideration at most once in **Stage 4**. Since **Stage 4-2** can be done in $O(1)$ time, **Stage 4** takes $O(n)$ time in total. In summary, the time complexity of **Stage 2** is the largest among all the stages, and then the running time of `Envelope` is $O(n \log n)$ in total. □

5 MAXSNP-Hardness

We shall show the proof of Theorem 1. In [13], the following restricted version of Job Interval Scheduling Problem, called *JISP2*, is shown to be MAXSNP-hard. We reduce the JISP2 to a restricted version of the PDMO in which all $m(b_i)$'s are equal to one, i.e., all the moving objects turn only once.

JISP2:

Instance: A set $J = \{1, 2, \cdots, n\}$ of n jobs, two intervals of length 2 associated with each job j: $I_{j,1} = [r_{j,1}, r_{j,1} + 2]$, and $I_{j,2} = [r_{j,2}, r_{j,2} + 2]$, where $r_{j,1}$ and $r_{j,2}$ are integers, $r_{j,1} \geq 0$ and $r_{j,1} + 2 < r_{j,2}$ holds.

Question: Find a longest sequence $\mathcal{S}$ of intervals $I_{j_1,k_1}, I_{j_2,k_2}, \cdots, I_{j_h,k_h}$ that satisfies the following two conditions:

– Either $I_{j,1}$ or $I_{j,2}$ can be included in $\mathcal{S}$ for any j. (Neither may be included.)

– For $1 \leq i \leq h-1$, $r_{j_i,k_i} + 2 \leq r_{j_{i+1},k_{i+1}}$ (i.e., no interval overlaps any others in $\mathcal{S}$.)

We show an *L-reduction* from the JISP2 to the PDMO. The definition of the L-reduction from problem A to problem B is as follows [18]:

> An L-reduction is a pair of functions R and S, both computable in logarithmic space, with the following two additional properties: First, if x is an instance of A with optimal cost $OPT(x)$, then $R(x)$ is an instance of B with optimal cost that satisfies $OPT(R(x)) \leq \alpha \cdot OPT(x)$, where α is a positive constant. Second, if s is any feasible solution of $R(x)$, then $S(s)$ is a feasible solution of x such that $|OPT(x) - c(S(s))| \leq \beta \cdot |OPT(R(x)) - c(s)|$, where β is another positive constant.

For a job j of the JISP2 which has parameters r_1 and r_2 as its starting times of intervals, we construct an object b_j with speed $v = 2$ and its path for the PDMO: As illustrated in Fig. 3, b_j is initially placed on a point A, and its path is bended at a point C. We determine (i) the radius l (and so F), (ii) the location of D, (iii) the location of C that determines E and G, (iv) the location of A, and finally (v) the location of B, in this order. The reason why the speed v is set to two is presented at the almost end of this proof.

We first set

$$l = \frac{\sqrt{v^2 - 1}}{v} = \frac{\sqrt{3}}{2},$$

and then $\overline{AC}$ and $\overline{CB}$ are tangential lines of a circle whose center is at O and radius is l. In addition to that, $\overline{AC}$ is parallel to the horizontal x-axis. We now determine the exact location of D. Let the points at which $T(t)$ gives the minimum on $\overline{AC}$ (and $\overline{CB}$) be D (and E) that can be obtained by (1) in the proof of Lemma 1,

$$D = \left(\frac{l}{\sqrt{v^2 - 1}}, l \right) = \left(\frac{1}{v}, \frac{\sqrt{v^2 - 1}}{v} \right) = \left(\frac{1}{2}, \frac{\sqrt{3}}{2} \right),$$

by which $|\overline{OD}| = 1$ holds.

Next we determine the location of C. Note that $|\overline{OC}|^2 = l^2 + |\overline{FC}|^2$ holds. Then, we choose the location of C such that $(|\overline{DC}| + |\overline{CE}|)/v = r_2 - (r_1 + 2)$ which is the gap between the first and the second intervals of the job j. Since

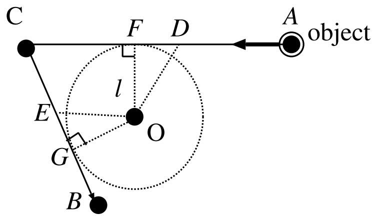


Fig. 3. Reduction from the JISP2 to the PDMO

$|\overline{DF}| = |\overline{GE}|$ and hence $|\overline{DC}|+|\overline{CE}| = |\overline{FC}|+|\overline{CG}| = 2|\overline{FC}|$, we set the location of C such that

$$|\overline{OC}|^2 = l^2 + \left(\frac{v(r_2-(r_1+2))}{2}\right)^2 = \frac{3}{4} + (r_2-(r_1+2))^2.$$

We can always determine the location of C that satisfies the above equation. Then the location of G is determined in straightforward, and also E is determined based on the location of C as D above.

The remainder of the points, A and B are to be determined. The location of A is the point such that $|\overline{AD}| = (r_1+1)v$, i.e.,

$$A = \left(\frac{1}{v} + (r_1+1)v, \frac{\sqrt{v^2-1}}{v}\right) = \left(\frac{1}{2} + 2(r_1+1), \frac{\sqrt{3}}{2}\right)$$

holds. Then we extend the line segment $\overline{CG}$ to some arbitrary position labeled as B. This ends the reduction that can be done in logarithmic space. Note that D and E can be always located on $\overline{AF}$ and $\overline{CG}$, respectively, because $|\overline{DF}| = |\overline{EG}| = 1/v = 1/2$ and the above configuration gives $|\overline{AF}| > v = 2$ and $|\overline{CG}| \geq v/2 = 1$. By this reduction, the minimum completion time for b_j is obtained by starting from O at time r_1 (or r_2), then grasps it on D (or E) at time r_1+1 (or r_2+1), and finally delivers it at time r_1+2 (or r_2+2).

For an instance X of the JISP2, we show the first condition of the L-reduction in a stronger form $OPT(X) = OPT(R(X))$, where $R(X)$ is an instance of the PDMO given as the above reduction from X. Let a feasible solution S of JISP2 be $I_{1,k_1}, I_{2,k_2}, \cdots$, where $k_i \in \{1,2\}$. For all i's, if $k_i = 1$, then we choose as the robot arm's motion $(r_{i,1}, b_i, r_{i,1}+2)$ in which the robot arm starts from O at time r_1, grasps b_i on D, and delivers it at time r_1+2. Otherwise ($k_i = 2$), $(r_{i,2}, b_i, r_{i,2}+2)$ that grasps b_i on E is chosen as well. Since $\forall i$, $r_{i,k_i}+2 \leq r_{i+1,k_{i+1}}$ in S, the scheduling of the robot arm chosen is valid, and then the number of objects collected is equal to the number of jobs done in S for the JISP2. At this point, we can only say that $OPT(R(X)) \geq OPT(X)$.

We call the robot arm's scheduling $\mathcal{S}$ is *canonical* if it includes only these two kinds of triples $(r_{i,1}, b_i, r_{i,1}+2)$ and $(r_{i,2}, b_i, r_{i,2}+2)$. Also we call a triple or motion (s, b, g) is *canonical* if it is either of them. In the above, we transformed a feasible solution of the JISP2 to a canonical scheduling of the PDMO with the same objective value. Next we will show that any scheduling of the robot arm can be transformed into a canonical one without changing the objective value (the number of collected objects). This implies that it is sufficient to consider canonical schedulings only, and then concludes that $OPT(R(X)) = OPT(X)$ since canonical schedulings can preserve the objective value.

Suppose that a scheduling $\mathcal{S}$ of the PDMO is not canonical. $\mathcal{S}$ includes (i) a motion starts before $r_{i,1}$(or $r_{i,2}$) and grasps an object b_i on $\overline{AD}$ (or $\overline{CE}$), and (ii) a motion starts after $r_{i,1}$(or $r_{i,2}$) and grasps an object b_i on $\overline{DC}$ (or $\overline{EB}$). See Fig. 3. As for (i), since the motion $(r_{i,1}, b_i, r_{i,1}+2)$ for object b_i has the minimum completion time, the motion (s, b_i, g) of (i) meets the conditions $s < r_{i,1}$ and $r_{i,1} + 2 < g$. Then, if $\mathcal{S}$ is valid, replacement of (s, b_i, g) to $(r_{i,1}, b_i, r_{i,1} + 2)$ in $\mathcal{S}$ also gives a valid scheduling and the number of collected objects does not change. As for the case considering the motion $(r_{i,2}, b_i, r_{i,2} + 2)$, it is similar.

Let $\mathcal{S}'$ be another scheduling which is obtained by replacing all the motions of (i) in $\mathcal{S}$ with canonical ones. That is, $\mathcal{S}'$ includes canonical motions and the motions of (ii). Consider the first motion (s, b_i, g) of (ii) in $\mathcal{S}'$. It satisfies $r_{i,1} < s$ and $r_{i,1} + 2 < g$ (or, $r_{i,2} < s$ and $r_{i,2} + 2 < g$). Suppose that $s - r_{i,1} \le t_{\max} < 1$ (which is shown later) where $t_{\max}$ denotes the maximum latency of time to start and grasp b_i compared to the canonical motion. Let us consider the case the robot arm collects b_j by a canonical motion $(r_j, b_j, r_j + 2)$, and next collects b_i by a motion (s, b_i, g) of (ii), i.e., $r_j + 2 \le s$. The intervals corresponding to the jobs of the instance X for the JISP2 are supposed to be $[r_j, r_j + 2]$ and $[r_i, r_i + 2]$. (We do not have to care about whether they are $I_{j,1}$ or $I_{j,2}$, and $I_{i,1}$ or $I_{i,2}$.) Then, $r_j + 2 \le s \le r_i + t_{\max}$ holds. There are the following two cases: (a) $r_i \ge r_j + 2$ and (b) $r_i < r_j + 2$. The case (a) implies that $[r_j, r_j + 2]$ and $[r_i, r_i + 2]$ do not overlap, and therefore, the canonical motion $(r_i, b_i, r_i + 2)$ can be adopted instead of the motion (s, b_i, g). In the case (b), since r_j and r_i are both integers, $r_i < r_j + 2$ implies that $r_i + 1 \le r_j + 2$. Then $s \le r_i + t_{\max} < r_i + 1 \le r_j + 2$, and this contradicts the assumption that the robot arm collects b_i starting at time s from the origin after it delivers b_j at time $r_j + 2$. From the cases (a) and (b), therefore, the first motion of (ii) in $\mathcal{S}'$ can be also replaced with a canonical motion. Since this replacement can be done iteratively, as a result all the motions of (ii) can be replaced with canonical ones without changing the number of collected objects.

Next we prove the assumption $t_{\max} < 1$ in the above discussion. Suppose that the robot has to start from O at time t to grasp an object b on the point D, which yields minimum completion time. Since $D = (1/2, \sqrt{3}/2)$, at time $t + 1$ the object b arrives at D. Consider the motion of the robot arm that starts at time $t + t_w(z)$ and grasps b at the point on $\overline{DC}$ whose distance from D is z, so that $\max_z t_w(z) = t_{\max}$. See Fig. 3 again. Since at that point, the object b arrives at time $t + 1 + z/v$, and so

$$t_w(z) = t + 1 + \frac{z}{v} - \left(t + \sqrt{l^2 + \left(\frac{1}{v} - z\right)^2} \right) = 1 + \frac{z}{2} - \sqrt{z^2 - z + 1}.$$

$$\frac{z}{2} - \sqrt{z^2 - z + 1} < 0$$

always holds, and then $t_w(z) < 1$ for any z that we would like to show. This observation is also true for the point E and the other objects. The rest of the proof for the L-reducibility is to show how to obtain a feasible solution $S(s_R)$ of X, the instance of the JISP2, from a feasible solution s_R of $R(X)$, the instance of the PDMO. Simply we need to transform s_R to a canonical one. The corresponding sequence of jobs is set as the feasible solution $S(s_R)$ for X. According to the above discussion, since the transformation of a scheduling of the robot arm to a canonical one does not change the number of collected objects, the number of collected objects in a canonical scheduling corresponds to the number of jobs executed in a solution for the JISP2, $c(S(s_R)) = c(s_R)$. In conjunction with $OPT(R(x)) = OPT(x)$, this implies the second condition of the L-reducibility in a stronger form, $|OPT(x) - c(S(s_R))| = |OPT(R(x)) - c(s_R)|$. □

6 Conclusion

In this paper we introduced a kinetic extension of the VRP, which is formulated as the robot grasp and delivery problem: Given moving objects, the objective is to find a collection of tours such that the robot arm grasps and delivers as many objects as possible. Then, we proved its MAXSNP-hardness, and gave a 2 factor approximation algorithm assuming that there is one robot arm with capacity one. An apparent remaining problem is to develop a better approximation algorithm with a performance ratio less than 2. Another direction for complexity and algorithmic research is to consider more general cases, for example, several robots move simultaneously, and/or each can collect the larger number of moving objects in one round. Especially, the case the capacity of the robot arm is a more general, for example, positive constant such as three, is an interesting variation to be considered. Two types of the problem in such a case are possible: The robot arm *has to* consume its capacity, i.e., collects exactly three objects, in each tour, or it can deliver only one or two objects to the origin if needed. For the former problem, the similar idea of `Iterated_Greedy` in Sect. 3 may work with additional polynomial time, though we have not yet obtained non-trivial results for the latter.

References

1. P. Toth and D. Vigo. An overview of vehicle routing problems, In *The Vehicle Routing Problem*, P. Tosh and D. Vigo (Eds.), SIAM, 2001.
2. N. Bansal, A. Blum, S. Chawla, A. Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *Proc. ACM Symposium on Theory of Computing*, pp.166–174, 2004.
3. A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In *Proc. IEEE Symposium on Foundations of Computer Science*, pp.46–55, 2003.
4. M. Desrochers, J. Desrosiers, M.M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40, pp.342–354, 1992.

5. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.). *The traveling salesman problem*, Wiley, Chichester, UK, 1985.
6. P. Chalasani, R. Motwani, A. Rao. Approximation algorithms for robot grasp and delivery. In *Proc. International Workshop on Algorithmic Foundations of Robotics*, pp.347–362, 1996.
7. C.S. Helvig, G. Robins, A. Zelikovsky. Moving target TSP and related problems. In *Proc. European Symposium on Algorithms*, pp.453–464, 1998.
8. M. Hammar and B.J. Nilsson. Approximation results for kinetic variants of TSP. In *Proc. International Colloquium on Automata, Languages and Programming*, pp.392–401, 1999.
9. Y. Asahiro, T. Horiyama, K. Makino, H. Ono, T. Sakuma, M. Yamashita. How to collect balls moving in the Euclidean plane. *Electronic Notes in Theoretical Computer Science*, 91, pp.229–245, 2004.
10. Y. Asahiro, E. Miyano, S. Shimoirisa. K-collect tours for moving objects with release times and deadlines. To appear in *Proc. Systemics, Cybernetics and Informatics*, 2005.
11. J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, pp.343–362, 1977.
12. J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15, pp.102–109, 1968.
13. F.C.R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2, pp.215–227, 1999.
14. D. Karger, C. Stein, J. Wein. Scheduling algorithms. In *Handbook of Algorithms and Theory of Computation*, M.J. Atallah (Eds), CRC Press, 1997.
15. A. Bar-Noy, S. Guha, J. Naor, B. Schieber. Approximating the throughput of multiple machines under real-time scheduling. *SIAM Journal on Computing*, 31 (2), pp.331-352, 2001.
16. J. Chuzhoy, R. Ostrovsky, Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problem. In *Proc. IEEE Symposium on Foundations of Computer Science*, pp.348–356, 2001
17. P.K. Agarwal, M. Sharir. Davenport-Schinzel sequences and their geometric applications. In *Handbook of Computational Geometry*, J. Sack and J. Urutia (Eds.), Elsevier Science, 1999.
18. C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1994.

A Static Analysis of PKI-Based Systems

Benjamin Aziz[1], David Gray[2], and Geoff Hamilton[2]

[1] Department of Computing, Imperial College,
180 Queen's Gate, London SW7 2AZ, U.K
baziz@doc.ic.ac.uk
[2] School of Computing, Dublin City University, Dublin 9, Ireland
{dgray, hamilton}@computing.dcu.ie

Abstract. This paper presents a non-uniform static analysis for *SPIKY*, an extension of the spi calculus with capabilities for PKI operations. The analysis, which follows a denotational framework, captures the property of term substitutions resulting from communications, cryptographic and PKI capabilities. The results of the analysis are used to formalise definitions of two security properties: the term secrecy and (un)certified peer-entity participation.

1 Introduction

In [7], an extension of the spi calculus [1] called the SPIKY language was introduced to clarify some of the issues related to key usage in spi calculus protocol specifications, like the validity of key-user bindings and users' ownership of processes, by making use of special primitives for performing Public-Key Infrastructure (PKI) key-retrieval operations. These included primitives for the retrieval of private and public parts of key pairs belonging to PKI users registered in the domain of the underlying PKI state. The primitives express both certified and uncertified public key-user bindings. Uncertified bindings are necessary for the modelling of protocols designed to run over devices with relatively limited computational power, e.g. handheld PDAs.

In this paper, we construct a non-uniform static analysis for the SPIKY language that captures the property of term substitutions occurring in PKI systems. In particular, the analysis captures PKI users sending and obtaining the substituted terms. Based on this information, it is possible to formalise security properties like for example, whether a user is capable of learning a term, and whether a user can (possibly) confirm that another user has participated in an instance of some protocol. To demonstrate this latter property, consider the following simple protocol between A and B, assuming they both know each other's public keys, K_A^+ and K_B^+, and N_A, N_B are fresh nonces:

Message 1	$A \rightarrow B : A, N_A$
Message 2	$B \rightarrow A : B, N_B, [\{N_A\}]_{K_B^-}$
Message 3	$A \rightarrow B : [\{N_B\}]_{K_A^-}$

A specification of this protocol in the spi calculus will necessarily make the assumption that the public key K_B^+ (respectively K_A^+) must be validly bound to B (respectively A), and that the partial correctness property of the protocol terminating will ensure that B (respectively A) participated in the protocol, even in the presence of middle-man attacks. Using the key retrieval capabilities of SPIKY, it is possible to assert this binding formally. Moreover, our static analysis will formally detect the fact that (un)certified public keys were used to arrive at the fact that B (respectively A) participated in the protocol, and that the private keys used in the signatures were not leaked to third parties.

The static analysis presented in this paper is directly related to previous analyses [3,4,5,6], which were designed to capture the term substitution property for systems specified in the π-calculus and the spi calculus. The main novelty about the current analysis is that it can deal with systems equipped with key retrieval capabilities over the underlying PKI state.

The rest of the paper is organised as follows. In Section 2 we review the syntax of the SPIKY language. In Section 3 we extend the domain model of [4] to SPIKY processes and we build a denotational semantics for the language. In Section 4, we give an extended semantics that captures the property of term substitutions in SPIKY language. This semantics is safely abstracted in Section 5. In Section 6, we formalise the security properties based on the results of the previous section and in Section 7, we review a simple example of a mobile authentication protocol. Finally, in Section 8, we conclude the paper.

2 The SPIKY Language

We review here the syntax of the SPIKY language introduced in [7]. The syntax consists of *terms*, *processes*, *systems* and *protocols*, as shown in Figure 1. The syntax of terms and processes is mostly similar to that of the spi calculus, except for the following differences, which occur in the syntax of processes:

- The *signature with recovery validation* process, *case* L *of* $[\{x\}]_N$ *in* P, is similar to its counterpart in the spi calculus, whereas the *signature with appendix validation* process, $[[\{L, M\}]_N]P$, behaves as P only if L is the signature $[\{M\}]_K$ and N is the public key K^+, otherwise, the process will block.
- *Abstraction instantiations*, $A(M)$, substitute x in the definition, $A(x) \triangleq P$, where x is bound to P, by the term M yielding a process $P[M/x]$. In order to simplify our static analysis, we consider only non-recursive definitions.
- The processes, *let* $x =$ $\mathbf{private}(M)$ *in* P, *let* $x =$ $\mathbf{public}(M)$ *in* P and *let* $x =$ $\mathbf{certified}(M)$ *in* P, attempt to retrieve the private, uncertified and certified public keys, respectively, of the PKI user, M. If successful, the retrieved key, k, substitutes x, and the process continues as $P[k/x]$, otherwise, the process blocks. The success or failure of these key-retrieval processes depends on the identity of the owner of that process.

In addition to terms and processes, *systems* formalise the idea of the ownership of a user, N, to a process, P, written as $\lceil P \rceil^N$, where N must reduce to an

L, M, N ::=		terms
	$a, b, c, k, m, n \in \mathcal{N}$	names
	$x, y, z, v, w \in \mathcal{V}$	variables
	$A, B, C, U \in \mathcal{AG}$	agents
	$\{M\}_N$	symmetric encryption
	$\{[M]\}_N$	public-key encryption
	$[\{M\}]_N$	digital signature
	(M, N)	pair
	M^+	public key component
	M	private key component
P, Q, R ::=		processes
	$\overline{M}\langle N\rangle.P$	output
	$M(x).P$	input
	$P \mid Q$	parallel composition
	$(\nu\, n)P$	restriction
	$!P$	replication
	$[M\ is\ N]P$	match
	$\mathbf{0}$	null
	$let\ (x, y) = M\ in\ P$	pair splitting
	$case\ L\ of\ \{x\}_N\ in\ P$	symmetric decryption
	$case\ L\ of\ \{[x]\}_N\ in\ P$	public-key decryption
	$case\ L\ of\ [\{x\}]_N\ in\ P$	signature with recovery validation
	$[[\{L, M\}]_N]P$	signature with appendix validation
	$A(M)$	abstraction instantiation
	$let\ x = \ \mathbf{private}(M)\ in\ P$	private key retrieval
	$let\ x = \ \mathbf{public}(M)\ in\ P$	public key retrieval
	$let\ x = \ \mathbf{certified}(M)\ in\ P$	certified public key retrieval
E, F, G ::=		systems
	$E \mid F$	parallel composition
	$(\nu\, n)E$	restriction
	$\lceil P \rceil^N$	process ownership
$Prot$ ::=		protocols
	(θ, E)	(PKI state, system) pair

Fig. 1. The syntax of the SPIKY language

agent's name. Systems can also be composed in parallel and new names can be introduced within the scope of a system. Finally, a *protocol* is a pair consisting of a PKI *state*, $\theta : \mathcal{AG} \rightarrow \mathcal{N}$, and a system, E. A PKI state maps distinct names of PKI users (agents) to their corresponding key pairs. The details of the actual binding of users to key pairs are abstracted, as well as the mechanisms for revocation. In general, θ is assumed to be up to date when a protocol is run.

In the following sections, we assume that the notions of α-conversion and term substitution as well as free and bound names and variables of terms and processes all apply as usual. We also assume that there are no occurrences of homonymous bound names and variables. Finally, in any protocol, we assume that there are only finitely many agents, possibly with replicated process behaviour.

3 A Domain-Theoretic Model

We introduce here a domain-theoretic model for closed processes in the SPIKY language that is an extension of the model originally given in [4] for the spi calculus. Our model is based on the following predomain equations:

$$\begin{aligned}
\mathit{Spiky} &\cong 1 + \mathbb{P}(\mathit{Spiky}_\perp + \mathit{In} + \mathit{Out}) && (1)\\
\mathit{In} &\cong N \times (T \rightarrow \mathit{Spiky}_\perp) && (2)\\
\mathit{Out} &\cong N \times (T \times \mathit{Spiky}_\perp + N \rightarrow \ldots N \rightarrow (T \times \mathit{Spiky}_\perp)) && (3)\\
T &\cong N + \mathit{Sec} + \mathit{Pub} + \mathit{Sig} + \mathit{Pair} && (4)\\
\mathit{Sec} &\cong T \times N && (5)\\
\mathit{Pub} &\cong T \times N && (6)\\
\mathit{Sig} &\cong T \times N && (7)\\
\mathit{Pair} &\cong T \times T && (8)
\end{aligned}$$

Where $\mathit{Spiky}_\perp$ is the domain of processes, *In* and *Out* are the predomains of input and output actions, respectively. Input actions are modelled as pairs; a name, N (the channel), and a function, $T \rightarrow \mathit{Spiky}_\perp$, that can be instantiated with a term, T, yielding a process in $\mathit{Spiky}_\perp$. Output actions are divided into free and bound output actions. These are pairs consisting of the channel, N, and either another pair, $T \times \mathit{Spiky}_\perp$, denoting the message, T, and the residue $\mathit{Spiky}_\perp$ (free outputs), or composed functions, $N \rightarrow \ldots N \rightarrow (T \times \mathit{Spiky}_\perp)$, that introduce new names to the message, T, and the residue, $\mathit{Spiky}_\perp$ (bound outputs). $\mathbb{P}(-)$ is Plotkin's powerdomain [8] applied to the disjoint union of input, output and silent actions (the latter represented by $\mathit{Spiky}_\perp$) to construct *Spiky*. The one-element predomain, 1, representing terminated (deadlocked) processes is adjoined as in [2, Def. 3.4]. The flat predomain of closed terms, T, is defined as the disjoint union of the predomains of names, N, secret-key ciphers, *Sec*, public-key ciphers, *Pub*, digital signatures, *Sig*, and pairs, *Pair*. The predomains *Sec*, *Pub* and *Sig* are represented as pairs, $T \times N$, where the term, T, is encrypted/signed using the key, N. There is no predomain of variables as part of the definition of T since we only deal with closed semantic terms.

The following functions are also defined, leading into $\mathit{Spiky}_\perp$ [2, Def. 3.3]:

$$\begin{aligned}
\emptyset &: 1 \rightarrow \mathit{Spiky}_\perp && (9)\\
\{\!|\, - \,|\!\} &: (\mathit{Spiky}_\perp + \mathit{In} + \mathit{Out})_\perp \rightarrow \mathit{Spiky}_\perp && (10)\\
\uplus &: (\mathit{Spiky}_\perp \times \mathit{Spiky}_\perp) \rightarrow \mathit{Spiky}_\perp && (11)\\
\mathit{new} &: (N \rightarrow \mathit{Spiky}_\perp) \rightarrow \mathit{Spiky}_\perp && (12)
\end{aligned}$$

The empty set, $\emptyset$, is required to represent inactive processes. The singleton map, $\{\!|\, - \,|\!\}$, creates elements of $\mathit{Spiky}_\perp$ from elements of input, output and silent actions, and $\uplus$, is the standard multiset union operator representing nondeterminism. Finally, *new* is used to interpret the effects of restriction.

Concrete elements of $t \in T$ include names, a, b, c, secret-key ciphers, $\mathit{sec}(t, k)$, public-key ciphers, $\mathit{pub}(t, k)$, digital signatures, $\mathit{sig}(t, k)$, and pairs, (t, t'). Elements $p \in \mathit{Spiky}_\perp$ include the bottom element, $\{\!|\perp|\!\}$, the empty set, $\emptyset$ (where $\{\!|\perp|\!\} \sqsubseteq \emptyset$), input actions, $\{\!|\mathit{in}(a, \lambda y.p)|\!\}$, free output actions, $\{\!|\mathit{out}(a, t, p)|\!\}$, bound

$$
\begin{array}{|lcl|}
\hline
new(\lambda n.\emptyset) & = & \emptyset \\
new(\lambda n.\{\!\mid \perp \mid\!\}) & = & \{\!\mid \perp \mid\!\} \\
new(\lambda n.\{\!\mid in(a,\lambda x.p) \mid\!\}) & = & \begin{cases} \emptyset, & \text{if } a = n \\ \{\!\mid in(a,\lambda x.new(\lambda n.p)) \mid\!\}, & \text{otherwise} \end{cases} \\
new(\lambda n.\{\!\mid out(a,t,p) \mid\!\}) & = & \begin{cases} \emptyset, & \text{if } a = n \\ \{\!\mid out(a,\lambda n.(t,p)) \mid\!\}, & \text{if } n \in n(t) \\ & \text{and } n \neq a \\ \{\!\mid out(a,t,new(\lambda n.p)) \mid\!\}, & \text{otherwise} \end{cases} \\
new(\lambda n.\{\!\mid out(a,\lambda m_1 \ldots \lambda m_k.(t,p)) \mid\!\}) & = & \\
\multicolumn{3}{|l|}{\begin{cases} \emptyset, & \text{if } a = n \\ \{\!\mid out(a,\lambda n.\lambda m_1 \ldots \lambda m_k.(t,p)) \mid\!\}, & \text{if } n \in n(t) \text{ and } n \neq a \\ \{\!\mid out(a,\lambda m_1 \ldots \lambda m_k.(t,new(\lambda n.p))) \mid\!\}, & \text{otherwise} \end{cases}} \\
new(\lambda n.\{\!\mid tau(p) \mid\!\}) & = & \{\!\mid tau(new(\lambda n.p)) \mid\!\} \\
new(\lambda n.(p_1 \uplus p_2)) & = & new(\lambda n.p_1) \uplus new(\lambda n.p_2) \\
\hline
\end{array}
$$

Fig. 2. The concrete definition of *new* over elements $p \in Spiky_\perp$

output actions, $\{\!\mid out(a,\lambda n_1 \ldots \lambda n_m.(t,p)) \mid\!\}$ silent actions, $\{\!\mid tau(p) \mid\!\}$ and sums, $p \uplus q$. The effects of restriction are interpreted by defining *new* as in Figure 2. These effects lead to the blocking of processes attempting to communicate over fresh, non-extruded channels and the transformation of free outputs to bound outputs whenever the message of communication is a fresh name. Otherwise, *new* has no effect and it is simply distributed over $\uplus$.

The denotational semantics for the SPIKY language is given as a semantic function, $\mathcal{S}([E])\ \rho\ \phi_{\mathcal{S}}\ \theta \in Spiky_\perp$, defined by the set of rules of Figure 3. The θ environment is defined as the PKI state of some protocol, such that $\theta(U)$ is a name representing the key pair associated with the user, U, where $\theta(U)^+$ is the public part of that key pair and $\theta(U)^-$ is the private part. The multiset, ρ, is used to hold systems composed in parallel with the analysed system and $\{\!\mid - \mid\!\}$ and $\uplus$ are overloaded from their definitions in (10),(11) to deal with ρ. Furthermore, rule ($\mathcal{R}0$) is used to interpret the contents of ρ. The environment, $\phi_{\mathcal{S}} : V \to T$, where V is the flat predomain of variables, captures any term substitutions that occur in the semantics[1]. The special function, $\varphi_{\mathcal{S}}$, returns the semantic value of a term:

$$
\forall \phi_{\mathcal{S}}, M : \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M) = \begin{cases} \phi_{\mathcal{S}}(M), & \text{if } M \in N \wedge M \in V \\ sec(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = \{[M']\}_N \\ sig(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = [\{M'\}]_N \\ (\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L)), & \text{if } M = (N, L) \end{cases}
$$

Rules ($\mathcal{S}0A$) and ($\mathcal{S}0B$) interpret parallelism and restriction between two systems by joining the parallel systems to ρ and using the *new* operator, respectively. Rules ($\mathcal{S}1$)–($\mathcal{S}16$) deal with process ownership by cases. Rule ($\mathcal{S}1$), deals with output actions taking into consideration any communications that may occur between the output channel and appropriate input channels guarding processes in ρ. The $\phi_{\mathcal{S}}$ is updated appropriately with the substituted semantic elements. Rules ($\mathcal{S}2$) deals with input functions leaving out communications since these are considered in ($\mathcal{S}1$). Rule ($\mathcal{S}3$) interprets directly parallel

[1] Note that initially, $\forall u \in V + N : \phi_{\mathcal{S}0}(u) = u$.

$$
\begin{array}{lll}
(\mathcal{S}0A) & \mathcal{S}(\![E \mid F]\!)\ \rho\ \phi\ \ \theta & = \mathcal{R}(\![\{\!|E|\!\} \uplus \{\!|F|\!\} \uplus \rho]\!)\ \phi\ \ \theta \\
(\mathcal{S}0B) & \mathcal{S}(\![(\nu\, n)E]\!)\ \rho\ \phi\ \ \theta & = new(\lambda n.\mathcal{R}(\![\{\!|E|\!\} \uplus \rho]\!)\ \phi\ \ \theta) \\
(\mathcal{S}1) & \mathcal{S}(\![\lceil \overline{M}\langle L\rangle.P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{(\biguplus_{M\ (z).P\ \ U\ \ \rho} \{\!|tau(\mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho[\lceil P\ \rceil^{U}\ /\lceil M\ (z).P\ \rceil^{U}\ ]]\!)\ \phi\ \ \theta|\!\})} \\
& \multicolumn{2}{l}{\quad \uplus \{\!|out(\varphi\ \ (\phi\ \ , M), \varphi\ \ (\phi\ \ , L), \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ )|\!\}} \\
& & \text{where, } \phi\ \ = \phi\ \ [z \mapsto \varphi\ \ (\phi\ \ , L)] \\
& & \text{and, } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , M\ \ ) \in N \\
(\mathcal{S}2) & \mathcal{S}(\![\lceil M(y).P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta & = \{\!|in(\varphi\ \ (\phi\ \ , M), \lambda y.\mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta)|\!\} \\
& & \text{where } \varphi\ \ (\phi\ \ , M) \in N \\
(\mathcal{S}3) & \mathcal{S}(\![\lceil P \mid Q\rceil^{U}]\!)\ \rho\ \phi\ \ \theta & = \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \{\!|\lceil Q\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta \\
(\mathcal{S}4) & \mathcal{S}(\![\lceil (\nu\, n)P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta & = new(\lambda n.\mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta) \\
(\mathcal{S}5) & \mathcal{S}(\![\lceil !P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta & = \bigsqcup \mathcal{F} \\
& \multicolumn{2}{l}{\text{where, } \mathcal{F} = \{\{\!|\bot|\!\}, \mathcal{S}(\![\lceil \prod_{i} P[bnv_i(P)/bnv(P)]\rceil^{U}\ ]\!)\ \rho\ \phi\ \ \theta \mid i = 0 \ldots \infty\}} \\
& \multicolumn{2}{l}{\text{and, } bnv_i(P) = \{x_i \mid x \in bnv(P)\}} \\
(\mathcal{S}6) & \mathcal{S}(\![\lceil [M\ is\ N]P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta, \text{ if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , N) \\ \emptyset, \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}7) & \mathcal{S}(\![\lceil \mathbf{0}\rceil^{U}]\!)\ \rho\ \phi\ \ \theta & = \emptyset \\
(\mathcal{S}8) & \mathcal{S}(\![\lceil let\ (x, y) = M\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto t, y \mapsto t\ ]\ \theta, \text{ if } \varphi\ \ (\phi\ \ , M) = (t, t\ ) \\ \emptyset, \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}9) & \mathcal{S}(\![\lceil case\ L\ of\ \{x\}_N\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto t]\ \theta, \text{ if } \varphi\ \ (\phi\ \ , L) = sec(t, k) \text{ and } \varphi\ \ (\phi\ \ , N) = k \\ \emptyset, \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}10) & \mathcal{S}(\![\lceil case\ L\ of\ \{[x]\}_N\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto t]\ \theta, \text{ if } \varphi\ \ (\phi\ \ , L) = pub(t, k^{+}) \text{ and } \varphi\ \ (\phi\ \ , N) = k \\ \emptyset, \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}11) & \mathcal{S}(\![\lceil case\ L\ of\ [\{x\}]_N\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta, \text{ if } \varphi\ \ (\phi\ \ , L) = sig(t, k\ \ ) \text{ and } \varphi\ \ (\phi\ \ , N) = k^{+} \\ \text{where, } \phi\ \ = \phi\ \ [x \mapsto t] \\ \emptyset, \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}12) & \mathcal{S}(\![\lceil [\ [\{L, M\}]_N\ ]P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta, \text{ if } \varphi\ \ (\phi\ \ , L) = sig(t, k\ \ ) \text{ and } \varphi\ \ (\phi\ \ , N) = k^{+} \\ \qquad\qquad\qquad\qquad\quad \text{and } \varphi\ \ (\phi\ \ , M) = t \\ \emptyset, \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}13) & \mathcal{S}(\![\lceil A(M)\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ \theta, \text{ where } A(x) \triangleq P \text{ and } \phi\ \ = \phi\ \ [x \mapsto \varphi\ \ (\phi\ \ , M)] \\ \emptyset, \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}14) & \mathcal{S}(\![\lceil let\ x = \ \mathbf{private}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \theta(U)\ \ ]\ \theta, \text{ if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , U) \\ \emptyset, \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}} \\
(\mathcal{S}15) & \mathcal{S}(\![\lceil let\ x = \ \mathbf{public}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = & \\
& \multicolumn{2}{l}{\begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \theta(U)^{+}]\ \theta, \text{ if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , U) \\ \biguplus_{U\ \ dom(\theta)} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \theta(U\ \ )^{+}]\ \theta, \text{ otherwise} \end{cases}} \\
(\mathcal{S}16) & \multicolumn{2}{l}{\mathcal{S}(\![\lceil let\ x = \ \mathbf{certified}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\ \ \theta = \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \theta(M)^{+}]\ \theta} \\
(\mathcal{R}0) & \mathcal{R}(\![\rho]\!)\ \phi\ \ \theta & = \biguplus_{E\ \ \rho} \mathcal{S}(\![E]\!)\ (\rho \backslash \{\!|E|\!\})\ \phi\ \ \theta
\end{array}
$$

Fig. 3. The standard denotational semantics of the SPIKY language

composition by the addition of the parallel subprocesses to ρ. Rule ($\mathcal{S}4$) uses *new* to interpret the meaning of a restriction. Rule ($\mathcal{S}5$) interprets a replication, $\lceil !P\rceil^{U}$, as the least upper bound of the infinite poset $\mathcal{F}$. This least upper bound represents the least fixed point meaning of $!P$. Due to the fact that the semantic domain, $Spiky_{\bot}$, is infinite, the calculation of this least fixed point may not terminate within finite limits. The rule also uses a labelling mechanism to

$$
\begin{array}{ll}
(\mathcal{E}0A) & \mathcal{E}(\![E \mid F]\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\![\{\!|E|\!\} \uplus \{\!|F|\!\} \uplus \rho]\!)\ \phi\quad \theta \\
(\mathcal{E}0B) & \mathcal{E}(\![(\nu\, n)E]\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\![\{\!|E|\!\} \uplus \rho]\!)\ \phi\quad \theta \\
(\mathcal{E}1) & \mathcal{E}(\![\lceil \overline{M}\langle L\rangle.P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \bigcup_{\phi \; M\ (z).P\quad U\quad \rho} \phi\quad \cup_{\phi}\ \phi \\
 & \qquad \text{if, } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , M\ \ ) \in N \\
 & \text{where, } \phi\ \ = \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho[\lceil P\ \ \rceil^{U}\ \ /\lceil M\ \ (z).P\ \ \rceil^{U}\ \ ]]\!)\ \phi \\
 & \text{and, } \phi\ \ = \phi\ \ [z \mapsto \{(\varphi\ \ (\phi\ \ , L), U, U\ \ )\}]\ \theta \\
(\mathcal{E}2) & \mathcal{E}(\![\lceil M(y).P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \phi \\
(\mathcal{E}3) & \mathcal{E}(\![\lceil P \mid Q\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \{\!|\lceil Q\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\quad \theta \\
(\mathcal{E}4) & \mathcal{E}(\![\lceil (\nu\, n)P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\quad \theta \\
(\mathcal{E}5) & \mathcal{E}(\![\lceil !P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \bigsqcup \mathcal{F} \\
 & \text{where, } \mathcal{F} = \{\bot_{D}\quad , \mathcal{E}(\![\ \lceil \prod_{i} P[bnv_i(P)/bnv(P)]\rceil^{U}\ ]\!)\ \rho\ \phi\quad \theta \mid\ i = 0\ldots\infty\} \\
 & \text{and, } bnv_i(P) = \{x_i \mid x \in bnv(P)\} \\
(\mathcal{E}6) & \mathcal{E}(\![\lceil [M\ is\ N]P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\quad \theta, & \text{if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , N) \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}7) & \mathcal{E}(\![\lceil \mathbf{0}\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \phi \\
(\mathcal{E}8) & \mathcal{E}(\![\lceil let\ (x,y) = M\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(t,U,U)\}, y \mapsto \{(t\ \ , U, U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , M) = (t, t\ \ ) \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}9) & \mathcal{E}(\![\lceil case\ L\ of\ \{x\}_{N}\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , L) = sec(t,k) \\ & \text{and } \varphi\ \ (\phi\ \ , N) = k \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}10) & \mathcal{E}(\![\lceil case\ L\ of\ \{\![x]\!\}_{N}\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , L) = pub(t,k^{+}) \\ & \text{and } \varphi\ \ (\phi\ \ , N) = k \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}11) & \mathcal{E}(\![\lceil case\ L\ of\ [\{x\}]_{N}\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , L) = sig(t,k\ \ \ ) \\ & \text{and } \varphi\ \ (\phi\ \ , N) = k^{+} \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}12) & \mathcal{E}(\![\lceil [\ [\{L, M\}]_{N}\ \ ]P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\quad \theta, & \text{if } \varphi\ \ (\phi\ \ , L) = sig(t,k\ \ \ ) \text{ and } \varphi\ \ (\phi\ \ , N) = k^{+} \\ & \text{and } \varphi\ \ (\phi\ \ , M) = t \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}13) & \mathcal{E}(\![\lceil A(M)\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(\varphi\ \ (\phi\ \ , M), U, U)\}]\ \theta, & \text{where } A(x) \triangleq P \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}14) & \mathcal{E}(\![\lceil let\ x =\ \mathbf{private}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(\theta(U)\ \ \ , U, U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , U) \\ \phi\ \ , & \text{otherwise} \end{cases} \\
(\mathcal{E}15) & \mathcal{E}(\![\lceil let\ x =\ \mathbf{public}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \begin{cases} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(\theta(U)^{+}, U, U)\}]\ \theta, & \text{if } \varphi\ \ (\phi\ \ , M) = \varphi\ \ (\phi\ \ , U) \\ \quad \bigcup_{\phi\ \ U\quad dom(\theta)} \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(\theta(U\ \ )^{+}, U, U)\}]\ \theta, & \text{otherwise} \end{cases} \\
(\mathcal{E}16) & \mathcal{E}(\![\lceil let\ x =\ \mathbf{certified}(M)\ in\ P\rceil^{U}]\!)\ \rho\ \phi\quad \theta = \\
 & \mathcal{R}(\![\{\!|\lceil P\rceil^{U}|\!\} \uplus \rho]\!)\ \phi\ \ [x \mapsto \{(\theta(M)^{+}, U, U)\}]\ \theta \\
(\mathcal{R}0) & \mathcal{R}(\![\rho]\!)\ \phi\quad \theta = \bigcup_{\phi\ E\quad \rho} \mathcal{E}(\![E]\!)\ (\rho \backslash \{\!|E|\!\})\ \phi\quad \theta
\end{array}
$$

Fig. 4. The non-standard semantics of the SPIKY language

rename all the bound variables and names, $bnv(P)$, of the spawned processes by subscripting those variables and names with a number signifying process copy.

Rule ($\mathcal{S}6$) resolves the meaning of two terms using $\varphi_{\mathcal{S}}$. Rule ($\mathcal{S}7$) interprets the meaning of a null system as the empty set mapping, $\emptyset$. Rule ($\mathcal{S}8$) splits the elements of a pair term. Rules ($\mathcal{S}9$)–($\mathcal{S}12$) deal with cryptographic systems for the decryption of symmetric and public-key ciphertexts and signatures with recovery and appendix validations. A residual system, $\lceil P \rceil^U$, signifying the success of the operation is added to ρ, else, if the operation fails, $\emptyset$ is returned instead. Rule ($\mathcal{S}$13) interprets the meaning of abstraction instantiations directly by adding the definition to ρ and updating $\phi_{\mathcal{S}}$ with the substituted term. Rules ($\mathcal{S}$14)–($\mathcal{S}$16) deal with PKI operations for retrieving private, uncertified and certified public keys. This is done using the PKI state, θ, and the user owning the system, U. The uncertified public key operation offers less guarantees (if the owner of the process requires other users' keys), therefore, it may return the public key of any PKI user, U', in $dom(\theta)$. On the other hand, the certified version is always guaranteed to return a valid public key, regardless of the owner's identity.

4 Non-Standard Semantics

We extend here the standard semantics of the previous section to a non-standard semantics that captures the property of term substitutions. Hence, the meaning of a system is now given as a special environment, $\phi_{\mathcal{E}} : V \rightarrow \wp(T \times AG \times AG)$, which maps each variable of a closed system to a set of triples representing semantic terms that may substitute the variable, and names of PKI users that instantiate and own that variable. Note that AG here represents the flat predomain of agent names corresponding to the set $\mathcal{AG}$. Since the non-standard semantics is precise (copies of bound names and variables are always distinct), each variable will be mapped to a singleton set per choice of control flow.

A non-standard semantic domain, $D_\perp = V \rightarrow \wp(T \times AG \times AG)$, can be constructed, ordered by subset inclusion as follows:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp : \phi_{\mathcal{E}1} \sqsubseteq_D \quad \phi_{\mathcal{E}2} \Leftrightarrow \forall x \in V : \phi_{\mathcal{E}1}(x) \subseteq \phi_{\mathcal{E}2}(x)$$

with the bottom element, $\perp_D$, being the null environment, $\phi_{\mathcal{E}0}$, that maps each variable to the empty set. The union of environments operation, $\cup_\phi$, can also be defined as follows:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp, x \in V : (\phi_{\mathcal{E}1} \cup_\phi \phi_{\mathcal{E}2})(x) = \phi_{\mathcal{E}1}(x) \cup \phi_{\mathcal{E}2}(x)$$

The non-standard semantics of the SPIKY language is defined using the semantic function, $\mathcal{E}([E])\ \rho\ \phi_{\mathcal{E}}\ \theta \in D_\perp$, as illustrated in Figure 4. The definitions of ρ and θ are as in Section 3. The definition of the special function, $\varphi_{\mathcal{E}}$ allows for the meaning of a closed term to be computed under some $\phi_{\mathcal{E}}$:

$$\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M)^2 = \begin{cases} t, & \text{if } M \in V \wedge \\ & \quad \phi_{\mathcal{E}}(M) = \{(t, U, U')\} \\ M, & \text{if } M \in N \\ sec(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{[M']\}_N \\ sig(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = [\{M'\}]_N \\ (\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'')), & \text{if } M = (M', M'') \end{cases}$$

[2] Note that the case where $M \in V \wedge \phi_{\mathcal{E}}(M) = \{\}$ will never occur for closed terms.

The main difference in this semantics as compared to the standard semantics of the previous section is the fact that the meaning of a process is a $\phi_{\mathcal{E}}$ environment rather than an element of $\mathit{Spiky}_{\perp}$. Note again the difference in performing uncertified versus certified public key retrieval in rules ($\mathcal{E}15$) and ($\mathcal{E}16$), respectively. In the former case, the owner of a process may obtain any public key stored in θ when asking for some other user's public key without any guarantees as to the validity of the key-user binding (unless the owner asks for its own public key). In the latter case, this requirement is always guaranteed to return a public key that is validly bound to its user.

5 Abstract Semantics

One problem with the non-standard semantics of the previous section is that it is not guaranteed to terminate due to the possibility of infinite behaviour resulting from the presence of replication. Therefore, it is necessary to introduce a safe abstraction that limits the size of the semantic domain. This abstraction is a variation of the abstraction used in [3,4,6].

We begin first by assuming a predomain of tags, *Tag*, ranged over by $t, \dot{t}, \ddot{t}$, where t is the tag of a generic term, $\dot{t}$ is the tag of a name or a variable, and $\ddot{t}$ is the tag of a complex term (ciphertext, signature, pair). Next, we appropriately tag M, M' in $\overline{N}\langle M\rangle.P$, $\mathit{let}\ (x, y) = (M, M')\ \mathit{in}\ P$, $\mathit{case}\ \{M\}_N\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P$, $\mathit{case}\ \{[M]\}_N\ \mathit{of}\ \{[x]\}_N\ \mathit{in}\ P$, $\mathit{case}\ [\{M\}]_N\ \mathit{of}\ [\{x\}]_N\ \mathit{in}\ P$, $A(M)$, and tag each of **private**(M), **public**(M) and **certified**(M) in the syntax. Furthermore, the following functions are defined over tags and systems:

– $\mathit{value\_of}(\{t_1, \ldots, t_n\}) = \{M_1, \ldots, M_n\}$, which when applied to a set of tags, $\{t_1, \ldots, t_n\}$, returns the corresponding set of terms, $\{M_1, \ldots, M_n\}$.
– $\mathit{tags\_of}(E) = \{t_1, \ldots, t_n\}$, which when applied to a system, E, returns its set of tags, $\{t_1, \ldots, t_n\}$.

We now introduce the $\alpha_{k,k}$ abstraction function, which keeps to a finite level, the number of copies of bound variables, bound names and tags.

Definition 1. *Define* $\alpha_{k,k} : \mathbb{N} \times \mathbb{N} \times (V + N + \mathit{Tag}) \to (V^{\sharp} + N^{\sharp} + \mathit{Tag}^{\sharp})$*:*

$$\forall M \in (V+N+\mathit{Tag}), i, k, k' \in \mathbb{N} : \alpha_{k,k}(M) = \begin{cases} \dot{t}_k, & \mathit{if}\ M = \dot{t}_i \in \mathit{Tag}\ \mathit{and}\ i > k \\ \ddot{t}_k, & \mathit{if}\ M = \ddot{t}_i \in \mathit{Tag}\ \mathit{and}\ i > k' \\ x_k, & \mathit{if}\ M = x_i \in V\ \mathit{and}\ i > k \\ a_k, & \mathit{if}\ M = a_i \in N\ \mathit{and}\ i > k \\ M, & \mathit{otherwise} \end{cases}$$

The resulting abstract predomains, $V^{\sharp}$, $N^{\sharp}$ and $\mathit{Tag}^{\sharp}$, can be defined as $V^{\sharp} = V \backslash \{x_j \mid j > k\}$, $N^{\sharp} = N \backslash \{a_j \mid j > k\}$ and $\mathit{Tag}^{\sharp} = \mathit{Tag} \backslash (\{\dot{t}_j \mid j > k\} \cup \{\ddot{t}_i \mid i > k'\})$. Informally, k constrains the number of bound variables and names, and tags of primitive terms, whereas k' constrains the number of tags of complex terms. In effect, constraining the tags of primitive terms implies limiting the copies of bound names and variables carrying the tags, whereas constraining the number of tags of complex terms means limiting the depth of data structures.

For example, in the process $!(\nu n)\overline{a}\langle n^{\dot{t}}\rangle \mid !a(x)$, it is possible to spawn infinite copies of each replication, $(\nu\, n_1)\overline{a}\langle n_1^{\dot{t}_1}\rangle \mid a(x_1) \mid (\nu\, n_2)\overline{a}\langle n_2^{\dot{t}_2}\rangle \mid a(x_2) \mid \ldots$. It is clear that $\dot{t}$ is an indicator to the number of copies n has after spawning each process. On the other hand, the process $!a(x).\overline{a}\langle\{x\}_k^{\ddot{t}}\rangle \mid \overline{a}\langle b\rangle$, which also spawns $a(x_1).\overline{a}\langle\{x_1\}_k^{\ddot{t}_1}\rangle \mid a(x_2).\overline{a}\langle\{x_2\}_k^{\ddot{t}_2}\rangle \mid \overline{a}\langle b\rangle \mid \ldots$ demonstrates the role of $\ddot{t}$ as an indicator to the number of times the ciphertext, $\{x\}_k$, is applied to b.

Using $\alpha_{k,k}$, we construct $\phi_{\mathcal{A}} : V^\sharp \to \wp(\mathit{Tag}^\sharp \times AG \times AG)$, with a meaning similar to $\phi_{\mathcal{E}}$ in the previous section. Furthermore, a domain, $D^\sharp_\perp = V^\sharp \to \wp(\mathit{Tag}^\sharp \times AG \times AG)$ is formed as follows:

$$\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D^\sharp_\perp, x \in V^\sharp : \phi_{\mathcal{A}1} \sqsubseteq_{D^\sharp} \phi_{\mathcal{A}2} \Leftrightarrow \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)$$

with a bottom element, $\perp_{D^\sharp}$, representing the null environment, $\phi_{\mathcal{A}0}$. Taking $D^\sharp_\perp$ as the abstract semantic domain, we can define the abstract semantics of the SPIKY language using the function, $\mathcal{A}([E])\ \rho\ \phi_{\mathcal{A}}\ \theta \in D^\sharp_\perp$, as shown in Figure 5. The definitions of ρ and θ are as in the previous sections. The special function, $\varphi_{\mathcal{A}}$, returns a set of terms corresponding to a term, M, given substitutions captured by $\phi_{\mathcal{A}}$, as follows:

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) = \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M')_{\{\}}$,
where, $M' = M[\alpha_{k,k}\ (t)/t][\alpha_{k,k}\ (x)/x][\alpha_{k,k}\ (n)/n]$, for all tags, t, names, n, and variables, x, of M,
and $\varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M)_s = \mathit{if}\ M \in s\ \mathit{then}\ \{\}\ \mathit{else}$

$$\begin{cases} \displaystyle\bigcup_{L \in \mathit{value\_of}(\mathit{fst}(\phi\ \ (M)))} \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}} & \text{if } M \in \mathcal{V} \\ \{M\}, & \text{if, } M \in \mathcal{N} \\ \{\{N'\}^t_L \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if, } M = \{N\}^t_L \\ \{\{[N']\}^t_L \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if, } M = \{[N]\}^t_L \\ \{[\{N'\}]^t_L \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if, } M = [\{N\}]^t_L \\ \{(L'_1, L'_2)^t \mid L'_1 \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L_1)_{s\cup\{M\}}, L'_2 \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L_2)_{s\cup\{M\}}\}, & \\ \text{if, } M = (L_1, L_2)^t & \end{cases}$$

We describe a few rules here. Rule ($\mathcal{A}1$) deals with the case of output actions, dealing with possible communications with appropriate input actions in ρ. The tag of the output message is registered in $\phi_{\mathcal{A}}$ as a value for the input variable. The semantics is imprecise, since $\phi_{\mathcal{A}}$ only captures an abstract tag as a value for an abstract variable. Rule ($\mathcal{A}5$) introduces the functions:

$\mathit{ren}(x, i) = \mathit{fold}\ \mathit{sub}_i\ (\mathit{fold}\ \mathit{sub}_i\ x\ \mathit{bnv}(x))\ \mathit{tags\_of}(x)$
$\mathit{fold}\ f\ e\ \{x_1, \ldots, x_n\} = f(x_n, \ldots, f(x_1, e)\ldots)$
$\mathit{sub}_i\ x\ y = y[x_i/x]$

which are used in the definition of the least fixed point meaning of a replicated process. This meaning is defined as the least upper bound of the set $\mathcal{F}$, which

$$(\mathcal{A}0A)\quad \mathcal{A}(\!\llbracket E \mid F \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| E |\!\} \uplus \{\!| F |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta$$

$$(\mathcal{A}0B)\quad \mathcal{A}(\!\llbracket (\nu\, n)E \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| E |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta$$

$$(\mathcal{A}1)\quad \mathcal{A}(\!\llbracket \lceil \overline{M}\langle L^t\rangle.P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \bigcup_{\substack{\phi \\ M\ (z).P\quad U\quad \rho}} \phi\quad \cup_{\phi}\ \phi$$
$$\text{if, } \varphi\ (\phi\ , M) \cap \varphi\ (\phi\ , M\ ) \cap \mathcal{N} \neq \{\}$$
$$\text{where, } \phi\ = \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho[\lceil P\ \rceil^U\ / \lceil M\ (z).P\ \rceil^U\ ] \rrbracket\!)\ \phi\quad \theta$$
$$\text{and } \phi\ = \phi\ [\alpha_{k,k}\ (z) \mapsto \phi\ (\alpha_{k,k}\ (z)) \cup \{(\alpha_{k,k}\ (t), U, U\ )\}]$$

$$(\mathcal{A}2)\quad \mathcal{A}(\!\llbracket \lceil M(y).P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \phi$$

$$(\mathcal{A}3)\quad \mathcal{A}(\!\llbracket \lceil P \mid Q\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \{\!| \lceil Q\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta$$

$$(\mathcal{A}4)\quad \mathcal{A}(\!\llbracket \lceil (\nu\, n)P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta$$

$$(\mathcal{A}5)\quad \mathcal{A}(\!\llbracket \lceil !P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \bigsqcup \mathcal{F}$$
$$\text{where, } \mathcal{F} = \{\bot_{D^\sharp}\ , \mathcal{A}(\!\llbracket\ \prod_i ren(P, i)\ \rrbracket\!)\ \rho\ \phi\quad \theta \mid\ i = 0 \ldots \infty\}$$

$$(\mathcal{A}6)\quad \mathcal{A}(\!\llbracket \lceil [M\ is\ N]P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \text{ if } \varphi\ (\phi\ , M) \cap \varphi\ (\phi\ , N) \neq \{\} \\ \phi\ , \text{ otherwise} \end{cases}$$

$$(\mathcal{A}7)\quad \mathcal{A}(\!\llbracket \lceil \mathbf{0}\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \phi$$

$$(\mathcal{A}8)\quad \mathcal{A}(\!\llbracket \lceil let\ (x, y) = M\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \bigcup_{\substack{\phi \\ (L^t, N^t\ )\ \ \varphi\ (\phi\ , M)}} \mathcal{R}(\!\llbracket \{\!| P |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \text{ if } \exists (L^t, N^t\ ) \in \varphi\ (\phi\ , M) \\ \text{where, } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{\alpha_{k,k}\ (t), U, U\}, \\ \alpha_{k,k}\ (y) \mapsto \phi\ (\alpha_{k,k}\ (y)) \cup \{\alpha_{k,k}\ (t\ ), U, U\}] \\ \phi\ , \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}9)\quad \mathcal{A}(\!\llbracket \lceil case\ L\ of\ \{x\}_N\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \bigcup_{\substack{\phi \\ M^t\ \ n\ \ \varphi\ (\phi\ , L)}} \mathcal{R}(\!\llbracket \{\!| P |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \text{ if, } n \in \varphi\ (\phi\ , N) \\ \text{where, } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}] \\ \phi\ , \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}10)\quad \mathcal{A}(\!\llbracket \lceil case\ L\ of\ \{[x]\}_N\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \bigcup_{\substack{\phi \\ [M^t]\ _{n^+}\ \ \varphi\ (\phi\ , L)}} \mathcal{R}(\!\llbracket \{\!| P |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \text{ if, } n\quad \in \varphi\ (\phi\ , N) \\ \text{where, } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}] \\ \phi\ , \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}11)\quad \mathcal{A}(\!\llbracket \lceil case\ L\ of\ [\{x\}]_N\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \bigcup_{\substack{\phi \\ [\ M^t\ ]_n\ \ \varphi\ (\phi\ , L)}} \mathcal{R}(\!\llbracket \{\!| P |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \text{ if, } n^+ \in \varphi\ (\phi\ , N) \\ \text{where, } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}] \\ \phi\ , \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}12)\quad \mathcal{A}(\!\llbracket \lceil [\ [\{L, M\}]_N\ ]P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta, \\ \text{if } \exists n^+ \in \varphi\ (\phi\ , N), [\{B^t\}]_n\quad \in \varphi\ (\phi\ , L) \wedge B^t \in \varphi\ (\phi\ , M) \\ \phi\ , \text{ otherwise} \end{cases}$$

$$(\mathcal{A}13)\quad \mathcal{A}(\!\llbracket \lceil A(M^t)\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta \text{ where, } A(x) \triangleq P$$
$$\text{and } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}]$$

$$(\mathcal{A}14)\quad \mathcal{A}(\!\llbracket \lceil let\ x = \ \mathbf{private}(M)^t\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\begin{cases} \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}]\ \theta, \\ \text{ if } U \in \varphi\ (\phi\ , M), \text{ where, } \mathbf{private}(U) = \theta(U) \\ \phi\ , \text{ otherwise} \end{cases}$$

$$(\mathcal{A}15)\quad \mathcal{A}(\!\llbracket \lceil let\ x = \ \mathbf{public}(M)^t\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta = \mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\quad \theta$$
$$\text{where, } \phi\ = \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}]$$
$$\text{and, } value\_of(\{t\}) = \begin{cases} \{\theta(U)^+\}, \text{ if } U \in \varphi\ (\phi\ , M) \\ \{\theta(U\ )^+ \mid U\ \in dom(\theta)\}, \text{ otherwise} \end{cases}$$

$$(\mathcal{A}16)\quad \mathcal{A}(\!\llbracket \lceil let\ x = \ \mathbf{certified}(M)^t\ in\ P\rceil^U \rrbracket\!)\ \rho\ \phi\quad \theta =$$
$$\mathcal{R}(\!\llbracket \{\!| \lceil P\rceil^U |\!\} \uplus \rho \rrbracket\!)\ \phi\ [\alpha_{k,k}\ (x) \mapsto \phi\ (\alpha_{k,k}\ (x)) \cup \{(\alpha_{k,k}\ (t), U, U)\}]\ \theta,$$
$$\text{where, } \mathbf{certified}(M) = \theta(M)^+$$

$$(\mathcal{R}0)\quad \mathcal{R}(\!\llbracket \rho \rrbracket\!)\ \phi\quad \theta = \bigcup_{\substack{\phi \\ P\ U\quad \rho}} \mathcal{A}(\!\llbracket \lceil P\rceil^U \rrbracket\!)\ (\rho \backslash \{\!| \lceil P\rceil^U |\!\})\ \phi\quad \theta$$

Fig. 5. The abstract semantics of the SPIKY language

can only be finite in this semantic. As a result, the termination of the least fixed point is formalised as follows.

Theorem 1 (Termination of the least fixed point calculation)
The calculation of rule ($\mathcal{A}5$) *terminates.*

Proof Sketch. To prove the termination property, it is necessary to satisfy two requirements. First, the semantic domain must be finite. This is satisfied by the definition of $D^{\sharp}_{\perp}$, where $Tag^{\sharp}$ and AG are both finite. The second requirement is that $\mathcal{A}(\![\lceil \prod_i P\rceil^U]\!)\ \rho\ \phi_{\mathcal{A}}\ \theta$ must be monotonic over P, i.e. $\mathcal{A}(\![\lceil \prod_i P\rceil^U]\!)\ \rho\ \phi_{\mathcal{A}}\ \theta \sqsubseteq_{D^\sharp}$ $\mathcal{A}(\![\lceil \prod_{i+1} P\rceil^U]\!)\ \rho\ \phi_{\mathcal{A}}\ \theta$. □

We can state the safety of the abstract semantics by the following theorem.

Theorem 2 (Safety of the abstract semantics)
$\forall P, \rho, \phi_{\mathcal{E}}, \phi_{\mathcal{A}}, k, k', \mathcal{E}(\![\lceil P\rceil^U]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = (p, \phi'_{\mathcal{E}}), \mathcal{A}(\![\lceil P\rceil^U]\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = \phi'_{\mathcal{A}}$:
$(\exists M, x : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow$
$\exists t \in \phi_{\mathcal{A}}(\alpha_{k,k}\ (x)) : M^{\sharp} \in \mathit{value\_of}(\{t\})\ \wedge\ M^{\sharp} = \mathit{fold}\ \mathit{sub}_{k,k}\ \ M\ \mathit{nv}(M))$
$\Rightarrow$
$(\exists M, x : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow$
$\exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k}\ (x)) : M^{\sharp} \in \mathit{value\_of}(\{t\})\ \wedge\ M^{\sharp} = \mathit{fold}\ \mathit{sub}_{k,k}\ \ M\ \mathit{nv}(M))$
where, $\mathit{sub}_{k,k}\ \ x\ y = y[\alpha_{k,k}\ (x)/x]$
and, $\mathit{nv}(M)$ *is the set of names and variables of* M

Proof Sketch. The proof is by induction over the structure of the abstract semantics and relies on first proving the safety of the $\cup_{\phi}$ operation. □

The theorem states that for any term, M, captured in the non-standard semantics by including its $\varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M)$ value in the value of a variable, $\phi'_{\mathcal{E}}(x)$, then that corresponds to capturing a tag, t, in the abstract semantics, by $\phi'_{\mathcal{A}}(\alpha_{k,k}\ (x))$. The appropriateness of t is expressed by the ability to obtain (by folding) an abstract form, $M^{\sharp} = \mathit{fold}\ \mathit{sub}_{k,k}\ \ M\ \mathit{nv}(M)$, of the concrete term, M, by evaluating t using $\mathit{value\_of}$. More concisely, every concrete term, M, captured in the non-standard semantics is also captured in the form of the corresponding abstract tag, t, in the abstract semantics. From now on, we shall use the following predicate, to denote the property that a term, M, is captured by an agent, B, sent by another agent, A, given the results of an abstract semantics, $\phi_{\mathcal{A}}$, and the constraints, k and k':

$$\mathit{captured}(M, A, B, \phi_{\mathcal{A}}, k, k') \overset{\mathtt{def}}{=}$$
$$\exists t \in Tag^{\sharp}, x \in \mathit{dom}(\phi_{\mathcal{A}}) : (t, A, B) \in \phi_{\mathcal{A}}(x)\ \wedge\ M^{\sharp} \in \mathit{value\_of}(t)$$

6 Security Properties

In this section, we formalise two security properties that can be checked by the static analysis of the previous section. These properties are the *term secrecy* and the *peer-entity participation*.

6.1 Term Secrecy

We formalise the property that a term, M, is never leaked to some agent, U, with respect to the results of the abstract semantics, $\phi_{\mathcal{A}}$, and using the *captured* predicate defined in the previous section, as follows.

Property 1 (Secrecy of the term M w.r.t. U)
$\nexists U' \in AG : \ \mathit{captured}(M, U', U, \phi_{\mathcal{A}}, k, k')$

From now on, we write the predicate, $\mathit{secret}(M, U)$, to indicate that M remains secret with respect to U.

6.2 Peer-Entity Participation

Peer-entity participation means that an agent, A, knows to a certain degree of certainty that another agent, B, has participated in a session of some protocol in which A is also a participant. In reality, there are many scenarios that this property could be established, both in its one-way and two-way forms. In this section, we discuss one such scenario, where A creates a nonce, N, and N is signed by B, then, provided that only B has the knowledge of its own private key, A knows that B has just participated in the protocol if it receives back the signature and is able to verify it. The degree of certainty to which A may establish the property depends on whether A performs a certified or an uncertified retrieval of the public key of B. In the uncertified case, A may still be able to raise its degree of certainty by relying on trusted third parties to perform the certified public key retrieval.

Based on this, we define the *certified peer-entity participation* and the *uncertified peer-entity participation*, as follows.

Property 2 (Certified peer-entity participation of B by A)
$\mathit{captured}(\theta(B)^{-}, B, B, \phi_{\mathcal{A}}, k, k') \ \wedge$
$(\exists t \in \mathit{Tag}^{\sharp}, x \in \mathit{dom}(\phi_{\mathcal{A}}) : \ \phi_{\mathcal{A}}(x) = \{(t, A, A)\} \ \wedge \ \theta(B)^{+^{\sharp}} \in \mathit{value\_of}(t)) \ \wedge$
$(\exists U, L : \mathit{captured}([\![L]\!]_{\theta(B)} \ , U, A, \phi_{\mathcal{A}}, k, k')) \ \wedge$
$(\forall U' : U' \neq B \Rightarrow \mathit{secret}(\theta(B)^{-}, U'))$

Property 3 (Uncertified peer-entity participation of B by A)
$\mathit{captured}(\theta(B)^{-}, B, B, \phi_{\mathcal{A}}, k, k') \ \wedge$
$(\exists t \in \mathit{Tag}^{\sharp}, x \in \mathit{dom}(\phi_{\mathcal{A}}) : \ (t, A, A) \in \phi_{\mathcal{A}}(x) \ \wedge \ \theta(B)^{+^{\sharp}} \in \mathit{value\_of}(t)) \ \wedge$
$(\exists U, L : \mathit{captured}([\![L]\!]_{\theta(B)} \ , U, A, \phi_{\mathcal{A}}, k, k')) \ \wedge$
$(\forall U' : U' \neq B \Rightarrow \mathit{secret}(\theta(B)^{-}, U'))$

7 Example

We consider here an example of mobile protocol authentication taken from [7]. In this example, we have that agent A is running on a small device with computational power insufficient for performing the full certification of the public key of B, therefore it relies on a trusted server with much more computational power to complete the verification process and forward its signed nonce to B:

$$\begin{array}{rl}
INIT(a,b,s,ch,ch') \triangleq & (\nu\ n_a)\overline{ch}\langle a,n_a\rangle.ch(b',n_b,sig). \\
& [b \textbf{ is } b'] \textbf{ let } k_b = \textbf{public}(b) \textbf{ in} \\
& \quad [\ \{\!|sig,n_a|\!\}_{k_b}] \textbf{ let } k_a = \textbf{private}(a) \textbf{ in} \\
& \quad\quad \textbf{let } k_s = \textbf{certified}(s) \textbf{ in } \overline{ch'}\langle\{[\{\!|n_b|\!\}_{k_a},b,k_b]\}_{k_s}\rangle.\mathbf{0} \\
\\
RESP(b,a,ch) \triangleq & ch(a',n_a). \\
& [a \textbf{ is } a'] \textbf{ let } k_b = \textbf{private}(b) \textbf{ in} \\
& \quad (\nu\ n_b)\overline{ch}\langle b,n_b,\{\!|n_a|\!\}_{k_b}\rangle.ch(sig). \\
& \quad\quad \textbf{let } k_a = \textbf{certified}(a) \textbf{ in }\ [\ \{\!|sig,n_b|\!\}_{k_a}].\mathbf{0} \\
\\
SERV(a,s,ch1,ch2) \triangleq & \textbf{let } k_s = \textbf{private}(s) \textbf{ in} \\
& ch'(c).\textbf{case } c \textbf{ of } \{[p]\}_{k_s} \textbf{ in} \\
& \quad \textbf{let } (sig,b,key) = p \textbf{ in} \\
& \quad\quad \textbf{let } k_b = \textbf{certified}(b) \textbf{ in } [key \textbf{ is } k_b]\overline{ch}\langle sig\rangle.\mathbf{0} \\
\\
SYST \triangleq & (\nu\ ch)(\nu\ ch')(\lceil INIT(A,B,S,ch,ch')\rceil^{A}\ | \\
& \quad \lceil RESP(B,A,ch)\rceil^{B}\ |\ \lceil SERV(A,S,ch,ch')\rceil^{S})
\end{array}$$

Fig. 6. SPIKY definition of the mobile authentication protocol [7]

$$\begin{array}{ll}
(1.) & A \rightarrow B : A, N_A \\
(2.) & B \rightarrow A : B, N_B, \{\!|N_A|\!\}_{K_B} \\
(3.) & A \rightarrow S : \{[\{\!|N_B|\!\}_{K_A}\ , B, {K_B}^{+}]\}_{K_S{}^{+}} \\
(4.) & S \rightarrow B : \{\!|N_B|\!\}_{K_A}
\end{array}$$

The specification of this protocol in the SPIKY language is given in Figure 6.

7.1 Analysis Results

Applying $\mathcal{A}([\![SYST]\!])\ \rho\ \phi_{\mathcal{A}}\ \theta$, for the uniform case, where $k = k' = 1$, we find that Property 3 is satisfied for agent A, since *captured*$(\{\!|n_a|\!\}_{k_b}, B, A, \phi_{\mathcal{A}}, k, k')$ and A can only verify the signature with uncertified public key for B. Obviously, A later relies on the server, S, to compare the uncertified key of B with a certified version that S is capable of retrieving. On the other hand, we find that Property 2 is satisfied for agent B, since *captured*$(\{\!|n_b|\!\}_{k_a}, S, B, \phi_{\mathcal{A}}, k, k')$ is true and B can verify the signature with a certified public key of A. Finally, we also note that Property 1 is satisfied for the term, $\{[\{\!|n_b|\!\}_{k_a}, B, k_b]\}_{k_s}$ with respect to all the agents in the second protocol except the server, $SERV(A,S,ch,ch')$.

8 Conclusion

In this paper, we have presented a novel static analysis of PKI-based systems that captures the property of term substitutions in SPIKY-based specifications. The analysis was used to formally define the security properties of term secrecy and peer-entity participation. For the future, we hope to implement the analysis in some functional language, like SML or OCAML, and we plan then to use the implementation to verify a number of protocols that use PKIs.

References

1. Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the* 4th *ACM Conference on Computer and Communications Security*, pages 36–47, Zurich, Switzerland, April 1997. ACM Press.
2. Samson Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, June 1991.
3. B. Aziz. *A Static Analysis Framework for Security Properties in Mobile and Cryptographic Systems.* PhD thesis, School of Computing, Dublin City University, Dublin, Ireland, 2003.
4. B. Aziz, G.W. Hamilton, and D. Gray. A denotational approach to the static analysis of cryptographic processes. In *Proceedings of International Workshop on Software Verification and Validation*, volume 118, pages 19–36, Mumbai, India, December 2003. Electronic Notes in Theoretical Computer Science.
5. Benjamin Aziz and Geoff Hamilton. A privacy analysis for the π-calculus: The denotational approach. In *Proceedings of the* 2nd *Workshop on the Specification, Analysis and Validation for Emerging Technologies*, number 94 in Datalogiske Skrifter, Copenhagen, Denmark, July 2002. Roskilde University.
6. Benjamin Aziz, Geoff Hamilton, and David Gray. A static analysis of cryptographic processes: The denotational approach. *Journal of Logic and Algebraic Programming*, 64(2):285–320, August 2005.
7. David Gray, Benjamin Aziz, and Geoff Hamilton. Spiky: A nominal calculus for modelling protocols that use pkis. In *Proceedings of the International Workshop on Security Analysis of Systems: Formalism and Tools*, Orléans, France, June 2004.
8. Gordon Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, September 1976.

Subtyping Object and Recursive Types Logically
(Extended Abstract)

Steffen van Bakel[1,*] and Ugo de'Liguoro[2,**]

[1] Department of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ, UK
svb@doc.ic.ac.uk

[2] Dipartimento di Informatica, Università di Torino,
Corso Svizzera 185, 10149 Torino, Italy
deliguoro@di.unito.it

Abstract. Subtyping in first order object calculi is studied with respect to the logical semantics obtained by identifying terms that satisfy the same set of predicates, as formalized through an assignment system. It is shown that equality in the full first order ς-calculus is modelled by this notion, which on turn is included in a Morris style contextual equivalence.

1 Introduction

Subtyping is a prominent feature of type-theoretic foundation of object oriented programming languages. The basic idea is expressed by subsumption: any piece of code of type A can masquerade as a code of type B whenever A is a subtype of B, written $A{<:}B$.

In typed calculi equations are among terms of the same type; since terms may have several types because of subsumption, it is commonly postulated that if $a = b : A$ and $A{<:}B$ then $a = b : B$ (but not viceversa): call this *equational subsumption*. In the realm of object calculi, object types are essentially interfaces, and subtyping interface extension; therefore subsumption is justified by the intuition that any object which is able to react to messages mentioned in A a fortiori will answer correctly to messages in the smaller interfaces represented by its supertypes. Similarly equational subsumption is understood on the ground of contexts separability: a and b are contextually equivalent at type A if both typeable by A and no context with a hole of type A can take them apart. This provides an interpretation of subtyping: $A{<:}B$ should hold if any pair of terms contextually equivalent at type A cannot be separated at B.

Semantically this is understood in two ways according to the existing literature (see [10] ch. 10 for a gentle introduction to these approaches): either by means of coercions [4], or by inclusion of partial equivalence relations as in [5,6] and [1] ch. 14. But coercion semantics does not reflects the actual implementation practice of object-oriented

* Partially supported by the MIKADO project of the IST-FET Global Computing Initiative, no IST-2001-32222.

** Partially supported by EU within the FET - Global Computing initiative, project DART ST-2001-33477.

languages; on the other hand PER semantics is quite complex to use for reasoning about programs, and suffers of technical problems which are still open.

We propose a third approach which, in our view, can lead to a simpler logical framework for reasoning about object oriented programs. It is based on the ideas of logical semantics and domain logic. In the latter perspective the meaning of a term is determined by the set of the predicates it satisfies, so that two terms are equivalent if they are indiscernible. To account for equivalence "at" a certain type A we relativize this form of absolute indiscernibility to sets of predicates indexed over types, calling them *languages*. Hence a and b are logically equivalent at type A if they satisfy the same set of predicates form the language $\mathcal{L}_A$ associated to A.

For equational subsumption to be sound in our framework, it is needed that some relation between $\mathcal{L}_A$ and $\mathcal{L}_B$ exists whenever $A{<:}B$. Were we about a calculus of pure objects, such a relation would be simply $\mathcal{L}_A \supseteq \mathcal{L}_B$, and this is clearly enough. However, since in the present paper we consider a richer calculus with functions and recursive types, called $\mathsf{FOb}_{1<:\mu}$ in [1], this is no longer true in general, and is replaced by a more complex inclusion relation.

The logical equivalence is indeed the theory of a model. Such a model can be obtained by the filter model construction as in [3], with a more complex structure due to the presence of types (see [8] and [12]). Here we live aside the investigation of the model and concentrate on the theory itself, establishing two results: first if $\vdash a \leftrightarrow b : A$ is derivable in the equational theory of system $\mathsf{FOb}_{1<:\mu}$, then a and b are logically equivalent at type A; second two terms logically equivalent at type A are contextually equivalent at the same type. The latter result is a consequence of the characterization of convergence in terms of derivability of non trivial predicates in $\mathcal{L}_A$ much as in the case of λ-calculus and intersection types (see e.g. [11]). A similar result was proved for the type-free ς-calculus in [7].

Because of the limited space available, proofs are omitted: see [13].

1.1 Related Work

The present paper follows some previous works by the authors in [7,8,12]. The novelty is subtyping of object and recursive types, while subtyping polymorphism was considered in [8] for a λ-calculus with function and record types only. The idea of using languages to model types in a filter model originates from [2]: however in Abramsky's work the modeling of polymorphism was left out. In this case the predicate languages cannot be disjoint; moreover they need to have a structure reflecting the subtyping relation, as stressed above, a topic which has not been addressed in the literature. The theory of objects in [1] is a natural environment of investigation for the themes we are about here; Morris style contextual equivalence for first order object calculi is introduced and studied in [9], where system $\mathsf{FOb}_{1<:\mu}$ is considered: this is the reason for the choice of the same calculus in the present paper.

2 The System $\mathsf{FOb}_{1<:\mu}$

To keep the present exposition self-contained, we recall the definition of the system $\mathsf{FOb}_{1<:\mu}$ of [1]. As usual for polymorphic calculi, we will introduce type and term syntax in two steps: first by defining type expressions (pre-types) and pre-terms, namely terms decorated by pre-types; types and terms are then defined together with the the type derivation system as well-formed pre-types and well-typed pre-terms respectively.

Definition 1 (Pre-types and Pre-terms). Let $\mathcal{K}$ be a set of type constants, ranged over by K, and $\mathcal{V}$ a set of type variables, ranged over by X, $\{\ell_i \mid i \in N\}$ a denumerable set of *labels*, I and J finite subsets of N. The set of *types* $\mathcal{T}$, ranged over by $A, B, C, \ldots$ is defined by the following grammar:

$$A, B ::= K \mid X \mid [\ell_i{:}B_i \ ^{(i \in I)}] \mid A \rightarrow B \mid \mu X.A$$

The pre-terms of $\mathsf{FOb}_{1<:\mu}$ are defined through the following grammar, where c ranges over constants:

$$\begin{array}{rl} a, b ::= & x \mid c \mid \lambda x^A.a \mid a(b) \mid [\ell_i = \varsigma(x_i^A)b_i \ ^{(i \in I)}] \mid a.\ell \\ \mid & a.\ell \Leftarrow \varsigma(x)b \mid \mathsf{fold}(A, a) \mid \mathsf{unfold}(a) \end{array}$$

A type expression of the shape $[\ell_i{:}B_i \ ^{(i \in I)}]$ is used for an object type; $A \rightarrow B$ is the usual functional type and $\mu X.A$ is a recursive type: in the latter the type variable X is bound in A. In the expressions $\varsigma(x^A)b$ and $\lambda x^A.b$, x is bound in b; free and bound variables are defined as usual. Types and pre-terms are considered equal modulo α-conversion, i.e. up to renaming of bound variables.

In [1] the system is defined as the union of several fragments, which we subdivide into two parts; the first one concerns contexts, types and terms formation[1]:

Definition 2. A *context* for a type judgement is just a finite set E of type-decorated variables, of the shape $x{:}A$.

The system $\Delta_K \cup \Delta_x \cup \Delta_{\mathsf{Ob}} \cup \Delta_{\rightarrow} \cup \Delta_X \cup \Delta_\mu$ is given in Figure 1.

E is a *well-formed context* if $E \vdash \diamond$ is derivable, and A *is a type for* a if there exists E with $E \vdash a{:}A$.

The second one is about sybtyping:

Definition 3. The system $\Delta_{<:} \cup \Delta_{<:\mathsf{Ob}} \cup \Delta_{<:\rightarrow} \cup \Delta_{<:X} \cup \Delta_{<:\mu}$ can be found in Figure 2.

It is understood that such unions produce a set of inductive clauses generating a unique system where contexts and types in the rules from the first part can be formed according to the rules of the second part and vice-versa. There is also a certain redundancy: the context E, X is the same as $E, X <: \mathsf{Top}$. In what follows we will use the generic notation $\cdot \{\cdot \hookleftarrow \cdot\}$ for substitution both of type variables by type expressions and of term variables by terms, implicitly replacing all occurrences of the first parameter of $\{\cdot \hookleftarrow \cdot\}$ by the second in the preceding expression; as usual the replacements occur up to α-congruence to avoid variable clashes.

[1] As in [1], we will use a short-hand for rules, and write for example

$$\frac{E, x_i{:}A \vdash_\Sigma b_i{:}B_i \quad (\forall i \in I)}{E \vdash_\Sigma [\ell_i = \varsigma(x_i^A)b_i \ ^{(i \in I)}]{:}A} \text{ for } \frac{E, x_1{:}A \vdash_\Sigma b_1{:}B_1 \quad \ldots \quad E, x_n{:}A \vdash_\Sigma b_n{:}B_n}{E \vdash_\Sigma [\ell_i = \varsigma(x_i^A)b_i \ ^{(i \in I)}]{:}A}$$

assuming here that $I = \{1, \ldots, n\}$.

$$(\textsf{Env } \emptyset):\ \frac{}{\emptyset \vdash \diamond} \qquad (\textsf{Env } x):\ \frac{E \vdash A}{E, x{:}A \vdash \diamond}\ (x \notin dom(E)) \qquad (\textsf{Val } x):\ \frac{E', x{:}A, E'' \vdash \diamond}{E', x{:}A, E'' \vdash x{:}A} \qquad (\textsf{Type Const}):\ \frac{E \vdash \diamond}{E \vdash K}$$

$$(\textsf{Type Object}):\ \frac{E \vdash B_i \quad (\forall i \in I)}{E \vdash [\ell_i{:}B_i\ ^{(i\in I)}]} \qquad (\textsf{Val Object}):\ \frac{E, x_i{:}A \vdash b_i{:}B_i \quad (\forall i \in I)}{E \vdash [\ell_i = \varsigma(x_i^A)b_i\ ^{(i\in I)}]{:}A} \qquad (\textsf{Val Select}):\ \frac{E \vdash a{:}[\ell_i{:}B_i\ ^{(i\in I)}]}{E \vdash a.\ell_j{:}B_j}\ (j \in I)$$

$$(\textsf{Val Update}):\ \frac{E \vdash a{:}A \quad E, x{:}A \vdash_\Sigma b{:}B_j}{E \vdash (a.\ell_j \Leftarrow \varsigma(x^A)b){:}A}\ (A \equiv [\ell_i{:}B_i\ ^{i\in I}], j \in I)$$

$$(\textsf{Type Arrow}):\ \frac{E \vdash A \quad E \vdash B}{E \vdash A \to B} \qquad (\textsf{Val Fun}):\ \frac{E, x{:}A \vdash a{:}B}{E \vdash \lambda x^A.a{:}A \to B} \qquad (\textsf{Val Appl}):\ \frac{E \vdash a{:}A \to B \quad E \vdash b{:}A}{E \vdash a(b){:}B}$$

$$(\textsf{Env } X):\ \frac{E \vdash \diamond}{E, X \vdash \diamond}\ (X \notin dom(E)) \qquad (\textsf{Type } X):\ \frac{E', X, E'' \vdash \diamond}{E', X, E'' \vdash X}$$

$$(\textsf{Type Rec}):\ \frac{E, X \vdash A}{E \vdash \mu X.A} \qquad (\textsf{Val Fold}):\ \frac{E \vdash a{:}A\{X \hookleftarrow \mu X.A\}}{E \vdash \textsf{fold}(\mu X.A, a){:}\mu X.A} \qquad (\textsf{Val Unfold}):\ \frac{E \vdash a{:}\mu X.A}{E \vdash \textsf{unfold}(a){:}A\{X \hookleftarrow \mu X.A\}}$$

Fig. 1. Fragments $\Delta_K \cup \Delta_x \cup \Delta_{\textsf{Ob}} \cup \Delta_\to \cup \Delta_X \cup \Delta_\mu$

Definition 4 (Reduction). *Evaluating contexts* are term expressions with a hole [_], and are generated by the grammar:

$$\mathcal{E}[\_] ::= \_ \mid \mathcal{E}[\_].\ell \mid \mathcal{E}[\_].\ell \Leftarrow \varsigma(x^A)b \mid \mathcal{E}[\_](a) \mid \textsf{unfold}(\mathcal{E}[\_]) \mid \textsf{fold}(A, \mathcal{E}[\_]).$$

We will wrire $\mathcal{E}[a]$ for the replacement of _ by a in $\mathcal{E}$.

The *one-step reduction relation* on terms is the binary relation defined by the following rules:

$$\begin{array}{rcl} [\ell_i = \varsigma(x_i^{A_i})b_i\ ^{(i\in I)}].\ell_j & \to & b_j\{x_j \hookleftarrow [\ell_i = \varsigma(x_i^{A_i})b_i\ ^{(i\in I)}]\} \\ {[\ell_i = \varsigma(x_i^{A_i})b_i\ ^{(i\in I)}]}.\ell_j \Leftarrow \varsigma(x^A)b & \to & [\ell_i = \varsigma(x_i^{A_i})b_i{}^{i\in I\setminus j}, \ell_j = \varsigma(x^{A_j})b] \\ (\lambda x^A.a)(b) & \to & a\{x \hookleftarrow b\} \\ \textsf{unfold}(\textsf{fold}(X, a)) & \to & a \\ a \to b & \Rightarrow & \mathcal{E}[a] \to \mathcal{E}[b] \end{array}$$

The relation $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$.

The one-step reduction is from [9]. In [1], Ch. 6 the operational semantics of the object calculi is defined by means of a bigstep predicate $a \rightsquigarrow v$, where a is a closed term, v is a *value* as it is defined by the grammar:

$$v ::= c \mid \lambda x^A.a \mid [\ell_i : \varsigma(x_i^A)b_i\ ^{i\in I}] \mid \textsf{fold}(A, v).$$

$$(\mathsf{Sub\ Refl}):\ \frac{E \vdash A}{E \vdash A <: A} \qquad (\mathsf{Sub\ Trans}):\ \frac{E \vdash A <: B \quad E \vdash B <: C}{E \vdash A <: C} \qquad (\mathsf{Val\ Subsumption}):\ \frac{E \vdash a{:}A \quad E \vdash A <: B}{E \vdash a{:}B}$$

$$(\mathsf{Type\ Top}):\ \frac{E \vdash \diamond}{E \vdash \mathsf{Top}} \qquad (\mathsf{Sub\ Top}):\ \frac{E \vdash A}{E \vdash A <: \mathsf{Top}} \qquad (\mathsf{Sub\ Object}):\ \frac{E \vdash B_i \quad (\forall i \in I)}{E \vdash [\ell_i{:}B_i{}^{\,i \in I}] <: [\ell_i{:}B_i{}^{\,i \in J}]}\ (J \subseteq I)$$

$$(\mathsf{Sub\ Arrow}):\ \frac{E \vdash A' <: A \quad E \vdash B <: B'}{E \vdash A {\to} B <: A' {\to} B'} \qquad (\mathsf{Env}\ X{<:}):\ \frac{E \vdash A}{E, X <: A \vdash \diamond}\ (X \notin \mathit{dom}(E)) \qquad (\mathsf{Type}\ X{<:}):\ \frac{E', X <: A, E'' \vdash \diamond}{E', X <: A, E'' \vdash X}$$

$$(\mathsf{Sub}\ X):\ \frac{E', X <: A, E'' \vdash \diamond}{E', X <: A, E'' \vdash X <: A} \qquad (\mathsf{Type\ Rec}{<:}):\ \frac{E, X <: \mathsf{Top} \vdash A}{E \vdash \mu X.A}$$

$$(\mathsf{Sub\ Rec}):\ \frac{E \vdash \mu X.A \quad E \vdash \mu Y.B \quad E, Y <: \mathsf{Top}, X <: Y \vdash A <: B}{E \vdash \mu X.A <: \mu Y.B}$$

Fig. 2. Fragments $\Delta_{<:} \cup \Delta_{<:\mathsf{Ob}} \cup \Delta_{<:\to} \cup \Delta_{<:X} \cup \Delta_{\mu}$

It is easy to see that $a \rightsquigarrow v$ if and only if $a \xrightarrow{*} v$. The reduction relation is more general since it is defined for any term (possibly with free variable occurrences); it is even true that normal forms are not necessarily values. However it is easy to adapt the arguments in [1] to establish the following theorem:

Theorem 5 (Subject reduction property of $\mathsf{FOb}_{1<:\mu}$). *If $E \vdash a{:}A$ is derivable in the system $\mathsf{FOb}_{1<:\mu}$ and $a \to b$, then $E \vdash b{:}A$ is derivable as well.*

We just stress that, consistently with the definition of $\rightsquigarrow$ in [1], in the clause:

$$[\ell_i = \varsigma(x_i^{A_i})b_i\ ^{(i \in I)}].\ell_j \Leftarrow \varsigma(x^A)b \ \to\ [\ell_i = \varsigma(x_i^{A_i})b_i{}^{i \in I \backslash j}, \ell_j = \varsigma(x^{A_j})b]$$

a renaming of the self type of the bound variable x^A into x^{A_j} occurs. This is immaterial in the fragments of the ς-calculus without subtyping, but it is needed in the presence of rule ($\mathsf{Val\ Subsumption}$) since if $A = [\ell_i{:}B_i{}^{i \in I}]$, and $A <: C$, then we can give type C to any term of type A and therefore update a method in an object of type A with $\varsigma(x^C)b$; but the result of (naively) performing the update saving the self type C is no longer typeable, as the *selves* of the methods now have different types, so that rule ($\mathsf{Val\ Object}$) will not apply.

The reduction relation is trivially confluent. Even relaxing Definition 4 and taking the closure of $\to$ under arbitrary contexts would not destroy confluence, as can be shown e.g. by adapting the Martin-Löf technique for proving the Church-Rosser theorem for the λ-calculus. As for typed λ-calculi with recursion (e.g. PCF), typed terms do not necessarily have a normal form: $\Omega_B \equiv [\ell = \varsigma(x^A)x.\ell].\ell$ is typeable by B if A is any object type $[\ell{:}B, \ldots]$, and it is such that $\Omega_B \to \Omega_B$.

3 Predicates and Assignment

In this section we will introduce the syntax of the predicates and an assignment system to syntactically derive judgements associating predicates to typed terms under the assumption of similar judgements about a finite set of typed variables.

Predicates are transparently intersection types for a λ-calculus with records, and come from [7]. The essential difference is that the set of predicates is stratified into languages (see [8,12]), in such a way that whenever a predicate can be deduced for a term a, it belongs to the language $\mathcal{L}_A$ associated with A.

Much in the style of [11], in this section we will present a notion of *strict intersection types*, called *strict predicates* here; this is a technical choice and a departure from [12], making the proof theory of the system more manageable, without loss in the expressivity. Using these, we will define a notion of *predicate assignment*, which will consists basically of associating a predicate to a typed term.

Definition 6 (Predicates). $\mathcal{P}_\mathrm{S}$, the set of *strict predicates*, and the set $\mathcal{P}$ of *intersection predicates*, both ranged over by $\sigma, \tau, \ldots$, are defined through:

$$\begin{array}{rll} \mathcal{P}_\mathrm{S} & ::= \kappa \mid (\mathcal{P} \rightarrow \mathcal{P}_\mathrm{S}) \mid \langle \ell{:}\mathcal{P}_\mathrm{S} \rangle \mid \mu(\mathcal{P}_\mathrm{S}) & \\ \mathcal{P} & ::= (\mathcal{P}_{\mathrm{S}1} \wedge \ldots \wedge \mathcal{P}_{\mathrm{S}n}) & (n \geq 0) \end{array}$$

where κ ranges over a countable set of atoms. We will write ω for an intersection of zero strict types, and write $\wedge_{\underline{n}}\sigma_i$ for $\sigma_1 \wedge \ldots \wedge \sigma_n$, where we assume that each $\sigma_i \in \mathcal{P}_\mathrm{S}$. Also, rather than $\langle \ell{:}\sigma_1 \rangle \wedge \cdots \wedge \langle \ell{:}\sigma_n \rangle$ we will write $\langle \ell{:}\sigma_1 \wedge \cdots \wedge \sigma_n \rangle$ or $\langle \ell{:}\wedge_{\underline{n}}\sigma_i \rangle$, where $\underline{n} = \{1, \ldots, n\}$; also, rather than $\langle \ell_1{:}\sigma_1 \rangle \wedge \cdots \wedge \langle \ell_n{:}\sigma_n \rangle$ where the ℓ_i are distinct, we will write $\langle \ell_i{:}\sigma_i{}^{i \in \underline{n}} \rangle$ or $\langle \ell_i{:}\sigma_i{}^{\,(i \in I)} \rangle$.

Atomic predicates κ are intended to describe elements of atomic type in the domain of interpretation; $\sigma{\rightarrow}\tau$ is the property of functions sending element satisfying σ into elements satisfying τ; $\langle \ell{:}\sigma \rangle$ is the property of records having values that satisfy σ associated with the field ℓ. Predicates ω and $\sigma \wedge \tau$ mean '*truth*' and '*conjunction*' respectively. It should be noted that arbitrary intersection predicates like $(\sigma{\rightarrow}\tau) \wedge \langle \ell{:}\rho \rangle$ are allowed by the above definition.

To build a logic of predicates we need a notion of implication, written $\sigma \leq \tau$, which is a reflexive and transitive relation on predicates, defined below.

Definition 7 (Predicate pre-order). On predicates a pre-order $\leq$ is inductively defined by:

$$\frac{\sigma \leq \sigma_i}{\sigma \leq \wedge_{\underline{n}}\sigma_i}\ (\forall i \leq n \geq 0) \qquad \frac{}{\wedge_{\underline{n}}\sigma_i \leq \sigma_i}\ (\forall i \leq n \geq 1)$$

$$\frac{\rho \leq \sigma \quad \tau \leq \mu}{\sigma{\rightarrow}\tau \leq \rho{\rightarrow}\mu} \qquad \frac{\sigma \leq \tau \leq \rho}{\sigma \leq \rho} \qquad \frac{\sigma \leq \tau}{\langle \ell{:}\sigma \rangle \leq \langle \ell{:}\tau \rangle} \qquad \frac{\sigma \leq \tau}{\mu(\sigma) \leq \mu(\tau)}$$

Finally $\sigma = \tau \iff \sigma \leq \tau \leq \sigma$. A predicate is *trivial* if equivalent to ω.

Lemma 8. *The following rules are admissible*

$$\frac{}{\langle \ell_i{:}\sigma_i{}^{i \in I} \rangle \wedge \langle \ell_j{:}\tau_j{}^{j \in J} \rangle \leq \langle \ell_k{:}\rho_k{}^{\,(k \in I \cup J)} \rangle},\ \text{where} \begin{cases} \rho_k = \sigma_k \wedge \tau_k, & \text{if } k \in I \cap J, \\ \rho_k = \sigma_k, & \text{if } k \in I \backslash J, \\ \rho_k = \tau_k, & \text{if } k \in J \backslash I \end{cases}$$

$$\frac{}{\langle \ell_i{:}\sigma_i \ ^{(i\in I)}\rangle \leq \langle \ell_j{:}\sigma_j{}^{j\in J}\rangle}\ (J \subseteq I)$$

Lemma 9. *$\langle \ell_i{:}\sigma_i{}^{i\in I}\rangle \wedge \langle \ell_j{:}\tau_j{}^{j\in J}\rangle = \langle \ell_k{:}\rho_k \ ^{(k\in I\cup J)}\rangle$, provided $\sigma_i = \tau_i$ for $i \in I \cap J$.*

Although predicates are basically properties of untyped terms (resulting from typed terms essentially by erasing type decorations), types are quite relevant in the equational theory of the $\mathsf{FOb}_{1<:\mu}$ calculus; this was accounted for in [8,12] by means of the notion of *predicate languages*, whose definition easily extends to the present richer syntax.

Definition 10 (Languages). The set of all predicates $\mathcal{L}$ is stratified into a family $\{\mathcal{L}_A\}_A$ of sets of predicates called *languages*, indexed over closed types such that:

1. for every κ, there exists *exactly one* $K\in\mathcal{K}$ such that $\kappa \in \mathcal{L}_K$;
2. $\mathcal{L}_A$ is the least set (including atoms if $A \equiv K$) such that

$$\frac{\sigma_i \in \mathcal{L}_A \quad (\forall i \in \underline{n})}{\wedge_{\underline{n}}\sigma_i \in \mathcal{L}_A}\ (n \geq 0) \qquad \frac{\sigma \in \mathcal{L}_A \quad \tau \in \mathcal{L}_B}{\sigma \rightarrow \tau \in \mathcal{L}_{A\rightarrow B}} \qquad \frac{\sigma \in \mathcal{L}_{A\{X \leftarrow\!\dashv\, \mu X.A\}}}{\mu(\sigma) \in \mathcal{L}_{\mu X.A}}\ (\sigma \in \mathcal{P}_{\mathrm{s}})$$

$$\frac{\sigma \in \mathcal{L}_{A\rightarrow B_j}}{\langle \ell_j{:}\sigma\rangle \in \mathcal{L}_A}\ (A = [\ell_i{:}B_i \ ^{(i\in I)}], j \in I, \sigma \in \mathcal{P}_{\mathrm{s}})$$

The intuition behind languages is the following. Properties in $\mathcal{L}_A$ give some information about values of type A; to be a value of type A should then imply to enjoy at least a non-trivial property in $\mathcal{L}_A$. That two values are logically equivalent at type A means that they satisfy the same set of properties in that language; consistently $\mathcal{L}_{\mathsf{Top}}$ is the set of trivial types. A natural question is whether there exists a relation between languages and the subtyping relation, which is partly answered in the following proposition, for which we recall the definition of the Egli-Milner preorder over the powerset of any preordered set $(X, \leq)$: if $U, V \subseteq X$ then

$$U \sqsubseteq^{\natural} V \Longleftrightarrow \forall u \in U\ \exists v \in V.\ u \leq v\ \&\ \forall v \in V\ \exists u \in U.\ u \leq v.$$

Proposition 11. Let A and B be closed type expressions not including recursion and such that $\vdash A {<:} B$ then:

1. if A and B are object types then $\mathcal{L}_B \subseteq \mathcal{L}_A$;
2. if A and B are either object or arrow types then $\mathcal{L}_A \sqsubseteq^{\natural} \mathcal{L}_B$.

We say that $\mathcal{L}_B$ is a *restriction of* $\mathcal{L}_A$ when $\mathcal{L}_A \sqsubseteq^{\natural} \mathcal{L}_B$. Note that, since $\omega \in \mathcal{L}_B$ for any B (take $n = 0$ in the rule about intersection in Definition 10) and $\sigma \leq \omega$ for all σ, we have that $\mathcal{L}_B \subseteq \mathcal{L}_A$ implies $\mathcal{L}_A \sqsubseteq^{\natural} \mathcal{L}_B$. If $\mathcal{L}_A \sqsubseteq^{\natural} \mathcal{L}_B$ then $\mathcal{L}_B$ is weaker than $\mathcal{L}_A$: we speak of restriction, since its discriminating power is less than the power of $\mathcal{L}_A$, as it is immediately clear when $\mathcal{L}_B \subseteq \mathcal{L}_A$. This makes languages and restriction good candidates for modelling types and subtyping relation respectively.

The proof of Proposition 11 is by induction on the derivation of $\vdash A {<:} B$ and does not need to take the context into account at any step because of the assumptions; this is no longer true when recursive types are considered.

Definition 12. A map η from type variables to closed types is called a *type-environment*. For E a well-formed context, we say that η *respects the context* E if for any $X <: A \in E$ (if $X \in E$ then it is read as $X <: \mathsf{Top} \in E$) it is the case that $\mathcal{L}_{\eta(X)} \sqsubseteq^{\natural} \mathcal{L}_{\eta(A)}$, where $\eta(A)$ is the value of application to A of the obvious extension of η to the set of types.

Theorem 13. *If $E \vdash A <: B$, then for any type-environment η that respects E we have $\mathcal{L}_{\eta(A)} \sqsubseteq^{\natural} \mathcal{L}_{\eta(B)}$.*

We are now in place to introduce the main tool in the present work, namely the assignment system. It is a formal system to derive judgments of the form $a{:}A : \sigma$, whose intended meaning is: the denotation of a satisfies the property σ when seen as a value of type A (here a "value" could be the undefined object in the domain of interpretation: we shall see that in such a case σ has to be trivial).

Definition 14 (Statements, bases, compatibility).

1. A *statement* is an expression of the shape $a{:}A : \sigma$, where a is a term, A is a type for a, and σ is a predicate, and a is called the *subject* of this statement.
2. A *basis* Γ is a finite set of statements with only (distinct) term variables as subject.
3. For a basis Γ, we say that E *fits into* Γ, written $E \lhd \Gamma$, if $x{:}A{:}\sigma \in \Gamma$ implies $x{:}A \in E$. We write $\widehat{\Gamma}$ for the largest context that fits into Γ.
4. We say that two bases Γ_0, Γ_1 are *compatible* if there exists a context E including all variables occurring in both Γ_0 and Γ_1, fitting into both of them.
5. We say that Γ *preserves languages* if $\sigma \in \mathcal{L}_{\eta(A)}$ whenever $x{:}A{:}\sigma \in \Gamma$ and η is a type-environment respecting $\widehat{\Gamma}$.
6. We extend $\leq$ to bases by: $\Gamma' \leq \Gamma$ if and only if for every $x{:}A{:}\sigma \in \Gamma$ there exists $x{:}A{:}\sigma' \in \Gamma'$ such that $\sigma' \leq \sigma$.

Definition 15 (Predicate Assignment). The *predicate assignment system* to derive judgments of the form $\Gamma \vdash a{:}B : \sigma$ where Γ is a basis preserving languages, a a term, A a type and σ a predicate is defined in figure 3.

Lemma 16. *1. The rules*

$$\frac{\Gamma \vdash a{:}A : \sigma \quad \sigma \leq \tau}{\Gamma \vdash a{:}A : \tau} \quad \textit{and} \quad \frac{\Gamma \vdash a{:}A : \sigma \quad \sigma \leq \tau \quad \widehat{\Gamma} \vdash A <: B}{\Gamma \vdash a{:}B : \tau}\ (\tau \in \mathcal{L}_B)$$

are admissible.

2. If $\widehat{\Gamma} \vdash a{:}A$, $\widehat{\Gamma} \vdash A <: B$ and $\Gamma \vdash a{:}B : \tau$, then there exists $\sigma \in \mathcal{L}_A$ such that $\sigma \leq \tau$ and $\Gamma \vdash a{:}A : \sigma$.

4 Subject Reduction and Expansion

A minimal requirement for soundness of the assignment system is that predicates are invariant under reduction. This is established through the following result.

(Val x) :
$$\frac{}{\Gamma \vdash x{:}B : \sigma}\;(x{:}B{:}\tau \in \Gamma, \tau \leq \sigma)$$

(<:) :
$$\frac{\Gamma \vdash a{:}B : \sigma \quad \widehat{\Gamma} \vdash B <: C}{\Gamma \vdash a{:}C : \sigma}\;(\sigma \in \mathcal{L}_C)$$

(Val Fun) :
$$\frac{\Gamma, x{:}A{:}\tau \vdash a{:}B : \sigma}{\Gamma \vdash \lambda x^A.a{:}A{\to}B : \tau{\to}\sigma}$$

(Val Appl) :
$$\frac{\Gamma \vdash a{:}A{\to}B : \tau{\to}\sigma \quad \Gamma \vdash b{:}A : \tau}{\Gamma \vdash a(b){:}B : \sigma}$$

(Val Fold) :
$$\frac{\Gamma \vdash a{:}A\{X \hookleftarrow \mu X.A\} : \sigma}{\Gamma \vdash \mathsf{fold}(\mu X.A, a){:}\mu X.A : \mu(\sigma)}$$

(Val Unfold) :
$$\frac{\Gamma \vdash a{:}\mu X.A : \mu(\sigma)}{\Gamma \vdash \mathsf{unfold}(a){:}A\{X \hookleftarrow \mu X.A\} : \sigma}$$

in the next rules $A \equiv [\ell_i{:}B_i \ ^{(i \in I)}]$

(Val Select) :
$$\frac{\Gamma \vdash a{:}A : \langle \ell_j{:}\tau{\to}\sigma \rangle \quad \Gamma \vdash a{:}A : \tau}{\Gamma \vdash a.\ell_j{:}B_j : \sigma}$$

(Val Object) :
$$\frac{\Gamma, x_i{:}A{:}\tau_i \vdash b_i{:}B_i : \sigma_i \quad (\forall i \in I)}{\Gamma \vdash [\ell_i = \varsigma(x_i^A)b_i \ ^{(i \in I)}]{:}A : \langle \ell_j{:}\tau_j{\to}\sigma_j \rangle}\;(j \in I)$$

(Val Update$_1$) :
$$\frac{\Gamma \vdash a{:}A : \sigma \quad \Gamma, y{:}A{:}\rho \vdash b{:}B_j : \tau}{\Gamma \vdash (a.\ell_j \Leftarrow \varsigma(y^A)b){:}A : \langle \ell_j{:}\rho{\to}\tau \rangle}$$

(Val Update$_2$) :
$$\frac{\Gamma \vdash a{:}A : \langle \ell_j{:}\sigma \rangle \quad \Gamma, y{:}A{:}\rho \vdash b{:}B_i : \tau}{\Gamma \vdash (a.\ell_i \Leftarrow \varsigma(y^A)b){:}A : \langle \ell_j{:}\sigma \rangle}\;(i \neq j)$$

$$(\omega)\;\frac{E \vdash a{:}B}{\Gamma \vdash a{:}B : \omega}\;(E \lhd \Gamma) \qquad (\wedge I)\;\frac{\Gamma \vdash a{:}B : \sigma_i \quad (\forall i \in \underline{n})}{\Gamma \vdash a{:}B : \wedge_{\underline{n}}\sigma_i}\;(n \geq 1)$$

Fig. 3. Predicate Assignment

Theorem 17 (Subject Reduction). *If* $\Gamma \vdash a{:}A : \rho$, *and* $a \to a'$, *then* $\Gamma \vdash a'{:}A : \rho$.

Example 18. To better appreciate the importance of this standard result in the present setting, we review an example given in [12].

Suppose that $A \equiv [\ell_0{:}\mathit{Int}, \ell_1{:}\mathit{Int}]$ and $a \equiv [\ell_0 = \varsigma(x^A)1, \ell_1 = \varsigma(x^A)x.\ell_0]$ (using a constant 1 of type *Int*), so that in $\mathsf{FOb}_{1<:\mu}$ we have $\vdash a{:}A$. Then

$$\dfrac{\dfrac{}{x{:}A{:}\omega \vdash 1{:}\mathit{Int}{:}\mathsf{O}} \qquad \dfrac{\dfrac{}{x{:}A{:}\langle \ell_0{:}\omega{\to}\mathsf{O}\rangle \vdash x{:}A : \langle \ell_0{:}\omega{\to}\mathsf{O}\rangle}\,(\mathsf{Val}\ x) \quad \dfrac{}{x{:}A{:}\langle \ell_0{:}\omega{\to}\mathsf{O}\rangle \vdash x{:}A : \omega}\,(\omega)}{x{:}A{:}\langle \ell_0{:}\omega{\to}\mathsf{O}\rangle \vdash x.\ell_0{:}\mathit{Int}{:}\mathsf{O}}\,(\mathsf{Val\ Select})}{\vdash a{:}A : \langle \ell_0{:}\omega{\to}\mathsf{O}, \ell_1{:}\langle \ell_0{:}\omega{\to}\mathsf{O}\rangle{\to}\mathsf{O}\rangle}\,(\mathsf{Val\ Object}, \wedge I)$$

where ℓ_0 is a field, ℓ_1 is the method $\mathtt{get}\ell_0$, and $\mathsf{O} \in \mathcal{L}_{\mathit{Int}}$ is the predicate of being an odd integer. Using rules (Val Update$_1$), (Val Update$_2$) and $(\wedge I)$ one can derive (the seemingly incorrect):

$$\dfrac{\vdash a{:}A{:}\langle\ell_0{:}\omega\to\mathsf{O},\ell_1{:}\langle\ell_0{:}\omega\to\mathsf{O}\rangle\to\mathsf{O}\rangle \qquad \dfrac{}{y{:}A{:}\omega\vdash 2{:}\mathit{Int}{:}\mathsf{E}}}{\vdash(a.\ell_0\Leftarrow\varsigma(y^A)2){:}A{:}\langle\ell_0{:}\omega\to\mathsf{E},\ell_1{:}\langle\ell_0{:}\omega\to\mathsf{O}\rangle\to\mathsf{O}\rangle}$$

where $\mathsf{E}\in\mathcal{L}_{\mathit{Int}}$ is the predicate of being an even integer. This makes sense, however, since it simply states that if the value at ℓ_0 is an odd integer, then the method ℓ_1 will return an odd integer; it also states that this is vacuously true of the actual object $a.\ell_0\Leftarrow\varsigma(y^A)2$, since it has an even integer at ℓ_0. As a consequence of Theorem 17 we also know that this is harmless: indeed $(a.\ell_0\Leftarrow\varsigma(y^A)2).\ell_1\xrightarrow{*}2$ and we clearly assume that $\not\vdash 2{:}\mathit{Int}:\mathsf{O}$, so by contraposition $\not\vdash(a.\ell_0\Leftarrow\varsigma(y^A)2).\ell_1{:}\mathit{Int}:\mathsf{O}$. As a matter of fact, rule (Val Select) is not applicable, since $\not\vdash(a.\ell_0\Leftarrow\varsigma(y^A)2){:}A:\langle\ell_0{:}\omega\to\mathsf{O}\rangle$.

On the other hand, the following odd-looking assignment is legal as well, this time by rule (Val Object) and $(\wedge I)$:

$$\dfrac{\dfrac{}{x{:}A{:}\omega\vdash 1{:}\mathit{Int}{:}\mathsf{O}} \qquad \dfrac{\dfrac{}{x{:}A{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\vdash x{:}A{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle}\qquad\dfrac{}{x{:}A{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\vdash x{:}A{:}\omega}}{x{:}A{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\vdash(x.\ell_0){:}\mathit{Int}{:}\mathsf{E}}}{a\vdash A{:}\langle\ell_0{:}\omega\to\mathsf{O},\ell_1{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\to\mathsf{E}\rangle}$$

In the last case, however, the apparently odd predicate we deduce is of use to conclude as before:

$$\dfrac{\vdash a{:}A{:}\langle\ell_0{:}\omega\to\mathsf{O},\ell_1{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\to\mathsf{E}\rangle \qquad \dfrac{}{y{:}A{:}\omega\vdash 2{:}\mathit{Int}{:}\mathsf{E}}}{(a.\ell_0\Leftarrow\varsigma(y^A)2)\vdash A{:}\langle\ell_0{:}\omega\to\mathsf{E},\ell_1{:}\langle\ell_0{:}\omega\to\mathsf{E}\rangle\to\mathsf{E}\rangle}$$

which is what we expected.

The invariant property of predicates w.r.t. reduction is stronger as they are preserved even by expansion, as is the case for standard intersection type assignment systems (see e.g. [3,11]). However we have to be careful, since the simply typed λ-calculus is a subcalculus of $\mathsf{FOb}_{1<:\mu}$, for which it is known that subject expansion does not hold. In fact we can prove $\vdash(\lambda x^{A\to A}.x)(\lambda x^A.x){:}A\to A{:}\sigma\to\sigma$, but $\Gamma\not\vdash yy\{y\leftarrow(\lambda x^C.x)\}{:}A\to A:\sigma\to\sigma$, since there is no way to derive a type for yy for any choice of Γ and C.

As a matter of fact, subject expansion does hold for predicates whenever it is the case for types, and this suffices for giving semantics to typed terms consistently with the restriction of convertibility relation to terms of the same type.

Theorem 19 (Subject Expansion). *If $\Gamma\vdash a{:}A{:}\tau$, and a' is such that $\widehat{\Gamma}\vdash a'{:}A$ and $a'\to a$, then $\Gamma\vdash a'{:}A{:}\tau$.*

5 The Logical Equivalence

The assignment system of Definition 15 induces a logical notion of equivalence, according to which a and b are equal at A if they can be assigned the same set of predicates from $\mathcal{L}_A$. By extending this notion to open terms, we arrive at the following definition.

Definition 20 (Logical Equivalence).
Let a and b be terms such that $E \vdash a{:}A$ and $E \vdash b{:}A$; we define

$$[\![a{:}A]\!]_E = \{\sigma \in \mathcal{L}_A \mid \exists \Gamma . \widehat{\Gamma} = E \ \& \ \Gamma \vdash a{:}A{:}\sigma\}.$$

We then say that a and b are *logically equivalent* at A and environment E if

$$E \vdash a{:}A, E \vdash b{:}A \text{ and } [\![a{:}A]\!]_E = [\![b{:}A]\!]_E,$$

and write $a \simeq^{\mathcal{L}}_{E} b : A$.

Notice that, if the basis Γ respects languages, the requirement $\sigma \in \mathcal{L}_A$ in the above definition is clearly redundant. Logical equivalence is the theory of a model built out of predicates, where the denotation of a term is exactly the set of its properties: i.e. the *filter model* [3]. It can be constructed along the lines of [12], even if the type interpretation cannot be the same, because retractions do not model subtyping. We leave this investigation to further study, and concentrate here on the properties of logical equivalence.

A notion of equivalence among terms of the $\mathsf{FOb}_{1<:\mu}$ is defined via a system deriving statements of the shape $a \leftrightarrow b : A$, meaning that terms a and b are equal at type A; the system $\Delta_{=} \cup \Delta_{=x} \cup \Delta_{=<:} \cup \Delta_{=\rightarrow} \cup \Delta_{=\mathsf{Ob}} \cup \Delta_{=\mu}$ is shown in Figure 4, where with respect to the original system in [1], we have omitted the obvious rules, like (Eq Appl), and extensionality rules (called (Eval Eta) and (Eval Fold), respectively).

Such a notion includes (typed) convertibility but it does not coincide with it: in fact '$\leftrightarrow$' is a congruence whereas ' $\rightarrow$ ' is not closed under arbitrary contexts; more importantly, this is a consequence of subtyping and precisely of rule (Eq Sub Object)

(Eval Beta) :
$$\frac{E \vdash \lambda x^{A} b{:}A{\rightarrow}B \quad E \vdash a{:}A}{E \vdash (\lambda x^{A} b)(a) \leftrightarrow b\{x \leftarrow a\} : B}$$

(Eq Subsumption) :
$$\frac{E \vdash a \leftrightarrow a' : A \quad E \vdash A <: B}{E \vdash a \leftrightarrow a' : B}$$

(Eq Top) :
$$\frac{E \vdash a{:}A \quad E \vdash b{:}B}{E \vdash a \leftrightarrow b : \mathsf{Top}}$$

(Eq Select) :
$$\frac{E \vdash a \leftrightarrow a' : [\ell_i{:}B_i \ ^{i\in I}]}{E \vdash a.\ell_j \leftrightarrow a'.\ell_j : B_j}\ (j \in I)$$

(Eq Update) where $A \equiv [\ell_i{:}B_i \ ^{(i\in I)}]$:
$$\frac{E \vdash a \leftrightarrow a' : A \quad E, x{:}A \vdash b \leftrightarrow b' : B_j}{E \vdash a.\ell_j \Leftarrow \varsigma(x^{A})b \leftrightarrow a'.\ell_j \Leftarrow \varsigma(x^{A})b' : A}\ (j \in I)$$

(Eval Select) where $I \cap J = \emptyset,\ A \equiv [\ell_i{:}B_i \ ^{i\in I}],\ A' \equiv [\ell_i{:}B_i \ ^{i\in I\cup J}],\ a \equiv [\ell_i = \varsigma(x_i^{A}\ )b_i \ ^{i\in I}]$:
$$\frac{E \vdash a{:}A}{E \vdash a.\ell_j \leftrightarrow b_j\{x_j \leftarrow a\} : B_j}\ (j \in I)$$

(Eval Update) where $I \cap J = \emptyset,\ A \equiv [\ell_i{:}B_i \ ^{i\in I}],\ A' \equiv [\ell_i{:}B_i \ ^{i\in I\cup J}],\ a \equiv [\ell_i = \varsigma(x_i^{A}\ )b_i \ ^{i\in I}]$:
$$\frac{E \vdash a{:}A \qquad E, x{:}A \vdash b{:}B_j}{E \vdash a.\ell_j \Leftarrow \varsigma(x^{A})b \leftrightarrow [\ell_j = \varsigma(x^{A}\ )b, \ell_i = \varsigma(x^{A}\ )b_i \ ^{(i\in I\cup J\backslash\{j\})}] : A}\ (j \in J)$$

Fig. 4. The equation system $\Delta_{=} \cup \Delta_{=x} \cup \Delta_{=<:} \cup \Delta_{=\rightarrow} \cup \Delta_{=\mathsf{Ob}} \cup \Delta_{=\mu}$

(see the next example). Therefore, from the subject reduction and expansion theorems it does not follow that equality implies logical equivalence.

Example 21. Consider the terms (where $A \equiv [\ell_0{:}\mathit{Int}, \ell_1{:}\mathit{Int}]$)

$$a \equiv [\ell_0 = \varsigma(x_1^A)1, \ell_1 = \varsigma(x_1^A)1], b \equiv [\ell_0 = \varsigma(x_0^A)1, \ell_1 = \varsigma(x_1^A)x.\ell_0].$$

In [1], Section 7.6.2 it is argued that they cannot be equated at A. Indeed, they are not logically equivalent at A since, if we assume that 1 is the predicate expressing the property of "being the number 1", so $1 \in \mathcal{L}_{\mathit{Int}}$, and $\vdash 1{:}\mathit{Int}{:}1$, then $\vdash a{:}A{:}\langle \ell_1{:}\omega \to 1\rangle$ but $\not\vdash b{:}A{:}\langle \ell_1{:}\omega \to 1\rangle$. Indeed (omitting some parts of the derivation for readability):

$$\dfrac{\dfrac{}{x_1{:}A{:}\omega \vdash 1{:}\mathit{Int}{:}1}}{\vdash a{:}A{:}\langle \ell_1{:}\omega{\to}1\rangle}\ (\mathsf{Val\ Object})$$

Replacing a by b would not yield a valid derivation. The best we can do in the case of b is instead:

$$\dfrac{\dfrac{\dfrac{}{x_1{:}A{:}\langle \ell_0{:}\omega{\to}1\rangle \vdash x_1{:}A{:}\langle \ell_0{:}\omega{\to}1\rangle} \qquad \dfrac{}{x_1{:}A{:}\langle \ell_0{:}\omega{\to}1\rangle \vdash x_1{:}A{:}\omega}}{x_1{:}A{:}\langle \ell_0{:}\omega{\to}1\rangle \vdash x_1.\ell_0{:}\mathit{Int}{:}1}\ (\mathsf{Val\ Select})}{\vdash b{:}A{:}\langle \ell_1{:}\langle \ell_0{:}\omega{\to}1\rangle{\to}1\rangle}\ (\mathsf{Val\ Object})$$

To express this in natural language, what we have proved is that the value of a on calling method ℓ_1 is 1, and that this is a "field", in that it does not depend on other parts of a; on the other hand, for b the value returned by ℓ_1 depends on the actual value of ℓ_0 in b: the predicate $\langle \ell_1{:}\langle \ell_0{:}\omega{\to}1\rangle{\to}1\rangle$ expresses this.

However, in [1] paragraph 8.4.2 is observed that the equality $\vdash a \leftrightarrow b : [\ell_0{:}\mathit{Int}]$ is derivable since both

$$\vdash [\ell_0 = \varsigma(x_0^B)1] \leftrightarrow a : [\ell_0{:}\mathit{Int}] \text{ and } \vdash [\ell_0 = \varsigma(x_0^B)1] \leftrightarrow b : [\ell_0{:}\mathit{Int}]$$

can be obtained by rule (Eq Sub Object); this clearly shows that '$\leftrightarrow$' is not convertibility, since a, b and $[\ell_0 = \varsigma(x_0^B)1]$ are distinct normal forms and the reduction is confluent.

In our setting, we can show that $a \simeq_{\emptyset}^{\mathcal{L}} b : [\ell_0{:}\mathit{Int}]$ as well, and this is the effect of restricting to the language $\mathcal{L}_{[\ell_0:\mathit{Int}]}$: in fact the only non-trivial predicates in $\mathcal{L}_{[\ell_0:\mathit{Int}]}$ that we can derive for either a or b are $\langle \ell_0{:}\omega{\to}1\rangle$ (or greater than this w.r.t. $\leq$).

Theorem 13 is a first evidence of the consistency of the predicate assignment system with respect to the subtyping relation. It is however not enough, and we need to establish the following.

Corollary 22. If $a \simeq_E^{\mathcal{L}} b : A$ and $E \vdash A{<:}B$ then $a \simeq_E^{\mathcal{L}} b : B$.

We conclude this section by establishing that equality in $\mathsf{FOb}_{1<:\mu}$ system implies logical equivalence, proving that what we have seen in the Example 21 actually holds in general.

Theorem 23. *If $E \vdash a \leftrightarrow b : A$ then $a \simeq_E^{\mathcal{L}} b : A$.*

6 Observational Semantics and Adequacy

Observational semantics for the $\mathsf{FOb}_{1<:\mu}$ calculus has been defined in [9] in Morris-style, called there "contextual equivalence". In the same paper it has been shown that it coincides with a notion of bisimulation which is stronger than '$\leftrightarrow$'. We will adopt a slightly more general definition (we will write a^A for a closed term a such that $\vdash a{:}A$).

Definition 24 (Convergence). Given any (well formed) closed term a^A we say that it *converges* to value v, written $a \Downarrow v$, if $a \xrightarrow{*} v$. Moreover we say that a^A is convergent ($a \Downarrow$) if there exists a value v such that $a \Downarrow v$.

We will write $\_\,{:}A \vdash C[\_]{:}B$ to express that the closed context $C[\_]$ is well typed with type B, under the assumption that the "hole $\_$" has type A; $C[a]$ is the result of replacing '$\_$' by a in $C[\_]$.

Definition 25 (Observational Equivalence). Two closed terms a and b are called *observationally equivalent at type* A, written $a \simeq^{\mathcal{O}}_A b$, if both a^A and b^A, and for any ground type K and value v_K it is the case:

$$\forall C[\_].\ \_\,{:}A \vdash C[\_]{:}K \;\Rightarrow\; (C[a] \Downarrow v_K \Longleftrightarrow C[b] \Downarrow v_K).$$

This differs from definition of contextual equivalence in [9] in some respect. First, we consider contexts of any ground type as an "experiment"; moreover, we do not consider reduction rules for constants as "if then else"; as a consequence we cannot discriminate between different constants like **true** and **false**. It is for that reason that we use in Definition 20 the predicate $a \Downarrow v$ instead of $a \Downarrow$.

We claim that, when restricted to closed terms, logical equivalence is included in observational equivalence. To this aim we establish an adequacy result of the logical semantics w.r.t. convergence, by means of a realizability interpretation of predicates, proving that the characterization results of [7] are preserved in the typed context of the calculus $\mathsf{FOb}_{1<:\mu}$.

In the next definition the set of labels of A is defined as $\mathit{Label}(A) = \{\ell_i \mid i \in I\}$ only for $A \equiv [\ell_i{:}A_i \ ^{(i\in I)}]$; it is empty in all other cases. If a^A for some object type A, $\ell_j \in \mathit{Label}(A)$ and $a \Downarrow [\ell_i = \varsigma(x_i^A)b_i \ ^{(i\in I)}]$, then, for any c^A, $a.\ell(c)$ abbreviates $b_j\{x_j \leftarrow\!\dashv c\}$.

Definition 26 (Realizability Interpretation). The *realizability interpretation* of the predicate σ is a set $[\![\sigma]\!]$ of closed terms defined by induction over the structure of predicates as follows:

1. $[\![\kappa]\!] = \{a^K \mid \kappa \in \mathcal{L}_K \ \&\ \exists v.\vdash v : K : \kappa \ \&\ \ a \Downarrow v\}$,
2. $[\![\sigma{\rightarrow}\tau]\!] = \{a^{A\rightarrow B} \mid \exists x, b.\ a \Downarrow (\lambda x^A.b) \ \&\ \forall c^A \in [\![\sigma]\!].\ b\{x \leftarrow\!\dashv c\} \in [\![\tau]\!]\}$,
3. $[\![\langle \ell{:}\sigma{\rightarrow}\tau\rangle]\!] = \{a^A \mid a \Downarrow \ \&\ \ell \in \mathit{Label}(A) \,\&\, \forall c^A \in [\![\sigma]\!].\ a.\ell(c) \in [\![\tau]\!]\}$,
4. $[\![\mu(\sigma)]\!] = \{a^{\mu X.A} \mid a \xrightarrow{*} \mathsf{fold}(\mu X.A, b) \ \&\ b_{A\{X \leftarrow\!\dashv \mu X.A\}} \in [\![\sigma]\!]\}$,
5. $[\![\omega]\!] = \{a^A \mid A \text{ is a type}\}$,
6. $[\![\sigma\wedge\tau]\!] = [\![\sigma]\!] \cap [\![\tau]\!]$.

The next lemma says that $[\![\sigma]\!]$ is closed under reduction and expansion for any σ.

Lemma 27. *If $a^A \in [\![\sigma]\!]$ then for any b^A if $a \xrightarrow{*} b$ or $b \xrightarrow{*} a$ then $b^A \in [\![\sigma]\!]$.*

Lemma 28. *If $\sigma \leq \tau$ then $[\![\sigma]\!] \subseteq [\![\tau]\!]$.*

Theorem 29 (Realizability theorem). *Let ϑ be any closed substitution, and $a\vartheta$ be the effect of applying ϑ to a (with usual conventions to avoid free and bound variable clashes) . If $\Gamma \vdash a{:}A{:}\sigma$ and for all $x{:}B{:}\tau \in \Gamma$ it is the case that $\vartheta(x) \in [\![\tau]\!]$, then $a\vartheta \in [\![\sigma]\!]$.*

It is easily seen that values v can be assigned non-trivial predicates, so that $a \Downarrow v$ implies that the same predicates can be derived for a because of Theorem 19; on the other hand a straightforward induction shows that if σ is non trivial, then any $a^A \in [\![\sigma]\!]$ converges: by this and Theorem 29 we obtain a proof of the following corollary.

Corollary 30 (Characterization of convergence). Let a^A be any closed term: then $a \Downarrow$ if and only if $\vdash a{:}A{:}\sigma$ for some non trivial σ.

Theorem 31 (Logical Equivalence and Observational Equivalence). *Suppose that for any value v of ground type K we have exactly a non trivial predicate $\kappa \in \mathcal{L}_K$, that this predicates are distinct for different values and that $\vdash v : K : \kappa$ is assumed for each v. Then for any a^A and b^A, if $a \simeq^{\mathcal{L}} b : A$ then $a \simeq^{\mathcal{O}}_A b$.*

7 Concluding Remarks

By using bisimulation and its coincidence with observational equivalence, in [9] it is shown that, taking a and b as in example 21, $a \simeq^{\mathcal{O}}_{[\ell_1:\mathit{Int}]} b$. This is intuitively clear: the only way to separate a from b is to change the value of ℓ_0, since then the fact that $b.\ell_1$ depends on such a value while $a.\ell_1$ does not, becomes apparent; but the overriding of ℓ_0 is inhibited in contexts with the hole of type $[\ell_1{:}\mathit{Int}]$, where ℓ_0 is hidden.

It is not true, however, that $a \simeq^{\mathcal{L}} b : [\ell_1{:}\mathit{Int}]$, because the predicate $\langle \ell_1{:}\omega \to 1\rangle$ is in $\mathcal{L}_{[\ell_1:\mathit{Int}]}$, it is derivable for a even at type $[\ell_1{:}\mathit{Int}]$ but cannot be derived for b at any type.

That language inclusion is not sufficient to account for subtyping of object types, while it is for record types (see [8]) is the essential reason for the presence of rule (Val Select) in our system. It is reasonable to think that the failure of equivalences like $a \simeq^{\mathcal{L}} b : [\ell_1{:}\mathit{Int}]$ from example 21 depends on the fact that no rule accounts for the hiding effect of subtyping in the case of object types. One possibility for coping with such a limitation is the following rule:

$$\frac{I \cap J = \emptyset,\ A \equiv [\ell_i{:}B_i\ ^{i \in I \cup J}],\ A' \equiv [\ell_i{:}B_i\ ^{i \in J}],\ \langle \ell : \tau \to \rho\rangle \in \mathcal{L}_A\ :\quad \Gamma \vdash a{:}A{:}\langle \ell : \langle \ell_i{:}\sigma_i\ ^{i\in I}\rangle \wedge \tau \to \rho\rangle \quad \Gamma \vdash a{:}A{:}\langle \ell_i{:}\sigma_i\ ^{i \in I}\rangle}{\Gamma \vdash a{:}A'{:}\langle \ell{:}\tau \to \rho\rangle}$$

This rule formalizes the idea that when $A <: A'$ and A and A' are object types, the methods of any object of type A not mentioned in A' are hidden: therefore if a satisfies the premise of any arrow predicate concerning the hidden part, this will never change in contexts of type A', in such a way that the latter premise can be discharged. Clearly, with reference to the example 21, by this rule one can derive $\vdash b{:}[\ell_1{:}\mathit{Int}]{:}\langle \ell_1{:}\omega{\to}1\rangle$, which makes a and b logically indiscernible at type $[\ell_1 : \mathit{Int}]$.

The soundness with respect to observational equivalence of the system resulting by adding such a rule to the predicate assignment system can be proved by means of a modified realizability interpretation of predicates, but at the time of writing we do not know to what extent it actually solves the problem.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, 1996.
2. S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.
3. H. P. Barendregt, M. Coppo, and M. Dezani. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48:931–940, 1983.
4. V. Breazu-Tannen, T. Coquand, C. A. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991.
5. K. B. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87:196–240, 1990.
6. K. B. Bruce and J. C. Mitchell. Per models of subtyping, recursive types and higher-order polymorphism. In *Proc. of POPL*, 1992.
7. U. de'Liguoro. Characterizing convergent terms in object calculi via intersection types. *Lecture Notes in Computer Science*, 2004:315–328, 2001.
8. U. de'Liguoro. Subtyping in logical form. In *ITRS'02*, ENTCS 70. Elsevier, 2002.
9. A. Gordon and G. Rees. Bisimilarity for first-order calculus of objects with subtyping. In *Proc. of POPL'96*, pages 386–395, 1996.
10. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
11. S. van Bakel. Intersection Type Assignment Systems. *Theoretical Computer Science*, 151(2):385–435, 1995.
12. S. van Bakel and U. de'Liguoro. Logical semantics of the first order sigma-calculus. *Lecture Notes in Computer Science*, 2841:202–215, 2003.
13. S. van Bakel and U. de'Liguoro. Subtyping object and recursive types logically. `www.di.unito.it/~deligu/papers/`, July 2005.

The Language $\mathcal{X}$: Circuits, Computations and Classical Logic

(Extended Abstract)

Steffen van Bakel[1,⋆], Stéphane Lengrand[2], and Pierre Lescanne[3]

[1] Department of Computing, Imperial College London,
180 Queen's Gate London SW7 2BZ, U.K
svb@doc.ic.ac.uk

[2] School of Computer Science, University of St Andrews, North Haugh, St Andrews
Fife KY16 9SS, Scotland
SL@dcs.St-Andrews.ac.uk

[3] École Normale Supérieure de Lyon, 46 Allée d'Italie,
69364 Lyon 07, France
Pierre.Lescanne@ens-lyon.fr

Abstract. We present the syntax and reduction rules for $\mathcal{X}$, an untyped language that is well suited to describe structures which we call "circuits" and which are made of parts that are connected by wires. To demonstrate that $\mathcal{X}$ gives an expressive platform, we will show how, even in an untyped setting, that we can faithfully embed algebraic objects and elaborate calculi, like the naturals, the λ-calculus, Bloe and Rose's calculus of explicit substitutions $\lambda\mathbf{x}$, and Parigot's $\lambda\mu$.

1 Introduction

In the past, the study of the relation between *computation*, *programming languages* and *logic* has concentrated mainly on *natural deduction systems*. In fact, these carry the predicate '*natural*' deservedly; in comparison with, for example, *sequent style systems*, natural deduction systems are easy to understand and reason about. This holds most strongly in the context of *non-classical* logics. For example, the relation between *Intuitionistic Logic* and the *Lambda Calculus* (with types) is well-studied and understood, and has resulted in a vast and well-investigated area of research, resulting in, amongst others, functional programming languages. In an other direction, the deep study of sequent calculus resulted in Linear Logic.

Expressing classical logic in a natural deduction system comes with handicaps. This can be observed by looking at [20], where Gentzen commented that even his intuitionistic Natural Deduction calculus "lacks a certain formal elegance", while its Classical counterpart fares still worse, breaking the symmetry between the introduction and elimination rules. Because of these technical difficulties, Gentzen found that to achieve the main results of [19], "I had to provide a logical calculus especially suited to the purpose. For this the natural [deduction] calculus proved unsuitable."

⋆ Partially supported by École Normale Supérieure de Lyon, France.

In this paper, we will try and break a spear for the sequent-style approach, and make some further steps towards the development of a programming language based on cut-elimination for the sequent calculus for classical logic. Essentially following [17], we will present a language called $\mathcal{X}$ that describes circuits, and its reduction rules that join circuits. The logic we will consider contains only implication, but that is mainly because we, in this initial phase, aim for simplicitly; cut-elimination in sequent calculi is notorious for the great number of rules, which will only increase manifold when considering more logical connectors.

To break with the natural deduction paradigm comes with a price, in that no longer *abstraction* and *application* (corresponding to *introduction of implication* and *modes ponens*) are the basic tools of the extracted language. In fact, the language we obtain is more of a *continuation* style, that models both the *parameter* as well as the *context* call. However, abstraction and application can be faithfully implemented, and we will show how $\mathcal{X}$ can be used to describe the behaviour of functional programming languages at a very low level of granularity.

$\mathcal{X}$ as a Language for Describing Circuits. The basic pieces of $\mathcal{X}$ can be understood as components with entrance and exit wires and ways to describe how to connect them to build larger circuits. Those component will be quickly surveyed in the introduction and receive a more detailed treatment in Section 2. We call *"circuits"* the structures we build, because they are made of components connected by wires.

$\mathcal{X}$ as a Syntax for the Sequent Calculus. Starting from the proof of Dragalin [12], Herbelin proposed in his PhD [15] a Curry-Howard correspondence; this was more elaborated in [9] leading to the definition of the language $\overline{\lambda}\mu\tilde{\mu}$. Among other approaches we need to mention [14,10,11,7]; more generally, this work has connections with linear logic [13]. The relation between CBN and CBV in the context of $\overline{\lambda}\mu\tilde{\mu}$ was studied in detail in [23].

The origins of the language $\mathcal{X}$ we discuss in this paper lie in an observation made on the structure of derivations in $\overline{\lambda}\mu\tilde{\mu}$ in [9]. This that was picked up by Lengrand [17], who introduced $\mathcal{X}$ and investigated in-depth the relation between $\mathcal{X}$ and $\overline{\lambda}\mu\tilde{\mu}$. Later it became apparent that $\mathcal{X}$ also has strong connections with the notations for the sequent calculus as presented first by Urban in his PhD thesis [22]. With respect to [22] an improvement of this paper is to make the notation more intuitive and readable by moving to an infix notation, and to insist on the computational aspect. This is achieved by studying $\mathcal{X}$ in the context of the normal functional programming languages paradigms, but, more importantly, to cut the link between $\mathcal{X}$ and Classical Logic, in that we also consider circuits that do *not* correspond to proofs.

This main step forward with respect to previous work is achieved by moving to an *untyped* language (both [17] and [22] consider only well-typed objects) which serves as an expressive framework for representing the untyped lambda calculus, the untyped calculus of explicit substitutions and the untyped language $\overline{\lambda}\mu\tilde{\mu}$. In particular, in our setting we can model infinite computations.

In the future, we aim to study $\mathcal{X}$ outside the context of Classical Logic in much the same way as the λ-calculus is studied outside the context of Intuitionistic Logic.

$\mathcal{X}$ as a Fine Grained Operational Model of Computation. When taking the the λ-calculus as a model for programming languages, the operational behaviour is provided by *β-contraction*. As is well known, β-contraction expresses how to calculate the value of a function applied to a parameter. In this, the parameter is used to instantiate occurrences of the bound variable in the body via the process of *substitution*. This description is rather basic as it says nothing on the actual *cost* of the substitution, which is quite high at run-time. Usually, a calculus of *explicit substitutions* [8,1,18,16] is considered better suited for an accurate account of the substitution process and its implementation. When we refer to the calculus of explicit substitution we rather intend the calculus of *explicit substitution with explicit names* $\lambda\mathbf{x}$, due to Bloo and Rose [8]. $\lambda\mathbf{x}$ gives a better account of substitution as it integrates substitutions as first class citizens, decomposes the process of inserting a term into atomic actions, and explains in detail how substitutions are distributed through terms to be eventually evaluated at the variable level.

In this paper, we will show that the level of description reached by explicit substitutions can in fact be greatly refined. In $\mathcal{X}$, we reach a 'subatomic' level by decomposing explicit substitutions into smaller components. At this level, the calculus $\mathcal{X}$ explains how substitutions and terms interact.

The calculus is actually symmetric [5] and, unlike $\lambda\mathbf{x}$ where a substitution is applied to a term, a term in $\mathcal{X}$ can also be applied to a substitution. Their interaction percolates (propagates) subtly and gently through the term or substitution according to the direction that has been chosen. We will see that the these two kinds of interaction have a direct connection with call-by-value and call-by-name reduction, that both have a natural description in $\mathcal{X}$.

A notion of principal contexts for $\mathcal{X}$ has been defined in [4]; a tool [3,4] to study $\mathcal{X}$ has been developed (see `http://www.doc.ic.ac.uk/~jr200/X`) that allows to input circuits from $\mathcal{X}$ and have fine control over reduction.

The Ingredients of the Syntax. It is important to note that $\mathcal{X}$ does *not* have variables[1] –like the λ-calculus or $\overline{\lambda}\mu\tilde{\mu}$– as possible places where terms might be inserted; instead, $\mathcal{X}$ has *wires*, also called *connectors*, that can occur free or bound in a term. As for the λ-calculus, the binding of a wire indicates that it is *active* in the computation; other than in the λ-calculus, however, the binding is not part of a term that is involved in the interaction, but is part of the interaction itself.

There are two kinds of wires: *sockets* and *plugs* (corresponding to *variables* and *co-variables*, respectively, in [23]) that are reminiscent of values and continuations. Wires are not supposed to denote a location in a term like variables in the λ-calculus. Rather, they can be *connected* with wires in other components.

One specificity of $\mathcal{X}$ is that syntactic constructors bind *two* wires, one of each kind. In $\mathcal{X}$, bound wires receive a hat, so to show that x is bound we write $\hat{x}$ [24,25]. That a wire is bound in a term implies, naturally, that this wire is unknown outside that term, but also that it 'interacts' with another 'opposite' wire that is bound into another term. The interaction differs from one constructor to another, and is ruled by basic reductions (see Section 2). In addition to bound wires an introduction rule exhibits a free wire, that is exposed; this can correspond to the creation of the wire, which is then connectable.

[1] We encourage the reader to not become confused by the use of names like x for the class of connectors that are called plugs; these names are, in fact, inherited from $\overline{\lambda}\mu\tilde{\mu}$.

Contents of this Paper. In this paper we will present the formal definitions for $\mathcal{X}$, via syntax and reduction rules, and will show that the system is well behaved by stating a number of essential properties. We will define a notion of simple type assignment for terms in $\mathcal{X}$, in that we will define a system of derivable judgements for which the terms of $\mathcal{X}$ are witnesses; we will show a soundness result for this system by showing that a subject-reduction result holds.

We will also compare $\mathcal{X}$ with a number of its predecessors. In fact, we will show that a number of well-know calculi are easily, elegantly and surprisingly effectively implementable in $\mathcal{X}$. For anyone familiar with the problem of expressibility, in view of the fact that $\mathcal{X}$ is substitution-free, these result are truly novel. With the exception of the calculus $\overline{\lambda}\mu\tilde{\mu}$, the converse is unobtainable. This can easily be understood from the fact that the vast majority of calculi in our area is confluent (Church-Rosser), whereas $\mathcal{X}$ is not.

2 The $\mathcal{X}$-Calculus

The circuits that are the objects of $\mathcal{X}$ are built with three kinds of building stones, or constructors, called *capsule*, *export* and *mediator*. We define an operator *cut*, which is handy for describing circuit construction, and which will be eliminated eventually by *rules*. In addition we give *congruence among circuits*.

2.1 The Operators

Circuits are connected through *wires* that are named. In our description wires are directed: we know in which direction the 'ether running through our circuits' moves, and can say when a wire provides an entrance to a circuit or when a wire provides an exit. Thus we make the distinction between exit wires which we call *plugs* and entry wires which we call *sockets*; we will use the word *connectors* for either sockets or plugs.

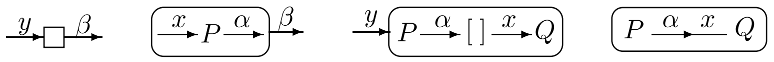
When connecting two circuits P and Q by the operator we may suppose that P has a plug α and Q has a socket x which we want to connect together to create a flow from P to Q. After the link has been established, the wires have been plugged, and the names of the connectors are forgotten; in fact, those names are *bound* in the link. We use the *"hat"*-notation to express binding, writing $\hat{x}$ to say that x is bound, keeping in line with the old tradition of *Principia Mathematica* [24]. The notion of free and bound connectors is defined as usual. We will normally adopt Barendregt's convention (called convention on variables by Barendregt, but here it will be a convention on names). An exception to that convention is the definition of natural numbers in Section 3.

Definition 1 (Syntax). The circuits of the $\mathcal{X}$-calculus are defined by the following grammar, where $x, y, \ldots$ range over the infinite set of *sockets*, and $\alpha, \beta, \ldots$ over the infinite set of *plugs*.

$$P, Q ::= \langle y.\beta\rangle \mid \widehat{x}P\widehat{\alpha}\cdot\beta \mid P\widehat{\alpha}\,[y]\,\widehat{x}Q \mid P\widehat{\alpha}\dagger\widehat{x}Q$$

Notice that, using Barendregt's convention, for example, the connector α in $P\widehat{\alpha}\,[y]\,\widehat{x}Q$ is supposed not to occur free in Q.

Diagrammatically, we represent the basic circuits as:



2.2 The Reduction Rules

The calculus, defined by the reduction rules below, explains in detail how cuts are distributed through circuits to be eventually erased at the level of capsules.

It is important to know when a connector is introduced, i.e. is connectable, i.e. is exposed and unique; this will play an important role in the reduction rules. Informally, a circuit P introduces a socket x if P is constructed from subcircuits which do not contain x as free socket, so x only occurs at the "top level." This means that P is either a mediator with a middle connector $[x]$ or a capsule with left part x. Similarly, a circuit introduces a plug α if it is an export that "creates" α or a capsule with right part α. We say now formally what it means for a terms to *introduce* a connector.

Definition 2 (Introduction).

P **introduces** x**:** $P = \langle x.\beta\rangle$ or $P = R\widehat{\alpha}\,[x]\,\widehat{y}Q$, with $x \notin \mathit{fs}(R,Q)$.
P **introduces** α**:** $P = \langle y.\alpha\rangle$ or $P = \widehat{x}Q\widehat{\beta}\cdot\alpha$ with $\alpha \notin \mathit{fp}(Q)$.

We first present a simple family of reduction rules, that specify how to reduce a cut with sub-circuits that both introduce the connectors mentioned in the cut.

Definition 3 (Logical Reduction). Assume that the terms of the left-hand sides of the rules *introduce the socket x and the plug α.*

$$\begin{array}{lrcl}
(\mathit{var}): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \to & \langle y.\beta\rangle\\
(\mathit{exp}): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x.\gamma\rangle & \to & \widehat{y}P\widehat{\beta}\cdot\gamma\\
(\mathit{med}): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{z}R) & \to & Q\widehat{\beta}\,[y]\,\widehat{z}R\\
(\mathit{ins}): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) & \to & Q\widehat{\gamma}\dagger\widehat{y}P\widehat{\beta}\dagger\widehat{z}R
\end{array}$$

The diagrammatical representation of these rules is given in Figure 1

Notice that, in rule (ins), in addition to the conditions for introduction of the connectors that are active in the cut ($\alpha \notin \mathit{fp}(P)$ and $x \notin \mathit{fs}(Q,R)$) we can also state that $\beta \notin \mathit{fp}(Q)\backslash\{\gamma\}$, as well as that $y \notin \mathit{fs}(R)\backslash\{z\}$, due to Barendregt's convention.

Still in rule (ins) the reader may have noticed that we did not put parenthesis in the expression $Q\widehat{\gamma}\dagger\widehat{y}P\widehat{\beta}\dagger\widehat{z}R$, which therefore is officially not a circuit. Instead, we should have given both the circuits $(Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R$ and $Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R)$ as result of the rewriting. However there is, in fact, a kind of associativity at play which means that we can omit the parenthesis; this will be made more clear in the next section.

We now need to define how to reduce a cut when one of its sub-circuits does not introduce a connector mentioned in the cut. This requires to extend the syntax with two new operators that we call *activated* cuts:

$$P ::= \ldots \mid P\widehat{\alpha}\nearrow\widehat{x}Q \mid P\widehat{\alpha}\nwarrow\widehat{x}Q$$

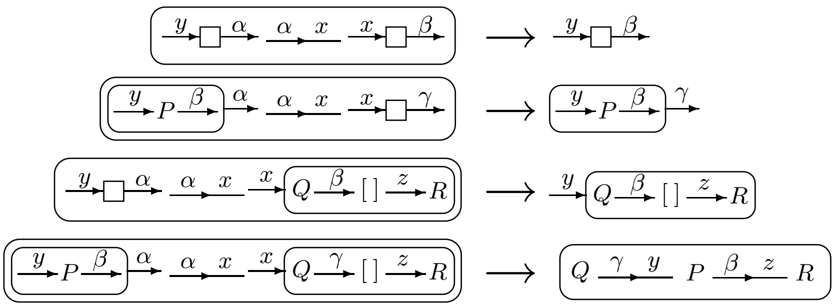


Fig. 1. The diagrammatical representation for the logical rules

Definition 4 (Activating the cuts).

$$
\begin{array}{ll}
(\textit{act-}\textsc{l}) : & P\widehat{\alpha} \dagger \widehat{x}Q \;\to\; P\widehat{\alpha} \not{\dagger}\, \widehat{x}Q, \;\textit{if } P \textit{ does not introduce } \alpha \\
(\textit{act-}\textsc{r}) : & P\widehat{\alpha} \dagger \widehat{x}Q \;\to\; P\widehat{\alpha} \backslash\!\!\dagger\, \widehat{x}Q, \;\textit{if } Q \textit{ does not introduce } x
\end{array}
$$

Notice that both side-conditions can be valid simultaneously, thereby validating both rewrite rules at the same moment. This gives, in fact, a *critical pair* or *superposition* for our notion of reduction, and is the cause for the loss of confluence. This notion of activation is related to the notion of colour in [11].

Circuits where cuts are not activated are called *pure* (the diagrammatical representation of activated cuts is the same as that for not activated cuts). Activated cuts are propagated through the terms, to reach a position where a logical rule can be applied.

Definition 5 (Propagation). The rules of propagation are given in Figure 2.

We will now define how to propagate a cut through sub-circuits. The direction of the activating shows in which direction the cut should be propagated, hence the two sets of reduction rules.

We will subscript the arrow that represents our reduction to indicate certain sub-systems, defined by a sub-reduction: for example, we will write $\to_{\text{A}}$ for the reduction that uses only rules in *Left propagation* or *Right propagation*. In fact, $\to_{\text{A}}$ is the reduction that pushes $\not{\dagger}$ and $\backslash\!\!\dagger$ inward.

The rules $(\textsc{l}2)$ and $(\textsc{r}3)$ deserve some attention. For instance, in the left-hand side of $(\textsc{l}2)$, α is not introduced, hence α occurs more than once in $\widehat{y}Q\widehat{\beta}\cdot\alpha$, that is once after the dot and again in Q. The occurrence after the dot is dealt with separately by creating the new name γ. Note that the cut associated with that γ is then unactivated; this is because, after the activated cut has been pushed through $\widehat{y}(Q\widehat{\alpha} \not{\dagger}\, \widehat{x}P)\widehat{\beta}\cdot\gamma$ (so leaves a circuit with no activated cut), the resulting term $(\widehat{y}R\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}P$ needs to be considered in its entirety: although we now that now γ is introduced, we do not know if x is. So, in any case, it would be wrong to activate the cut before the result of $Q\widehat{\alpha} \not{\dagger}\, \widehat{x}P$ (i.e. R) is known. The same thing holds for x in $(\textsc{r}3)$ and a new name z is created and the external cut is unactivated.

Left propagation

$$\begin{array}{lrcll}
(d\textsc{l}): & \langle y.\alpha\rangle\widehat{\alpha} \nearrow \widehat{x}P & \to & \langle y.\alpha\rangle\widehat{\alpha} \dagger \widehat{x}P & \\
(\textsc{l}1): & \langle y.\beta\rangle\widehat{\alpha} \nearrow \widehat{x}P & \to & \langle y.\beta\rangle, & \beta \neq \alpha \\
(\textsc{l}2): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha} \nearrow \widehat{x}P & \to & (\widehat{y}(Q\widehat{\alpha} \nearrow \widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}P, & \gamma\ \mathit{fresh} \\
(\textsc{l}3): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha} \nearrow \widehat{x}P & \to & \widehat{y}(Q\widehat{\alpha} \nearrow \widehat{x}P)\widehat{\beta}\cdot\gamma, & \gamma \neq \alpha \\
(\textsc{l}4): & (Q\widehat{\beta}\,[z]\,\widehat{y}R)\widehat{\alpha} \nearrow \widehat{x}P & \to & (Q\widehat{\alpha} \nearrow \widehat{x}P)\widehat{\beta}\,[z]\,\widehat{y}(R\widehat{\alpha} \nearrow \widehat{x}P) & \\
(\textsc{l}5): & (Q\widehat{\beta} \dagger \widehat{y}R)\widehat{\alpha} \nearrow \widehat{x}P & \to & (Q\widehat{\alpha} \nearrow \widehat{x}P)\widehat{\beta} \dagger \widehat{y}(R\widehat{\alpha} \nearrow \widehat{x}P) &
\end{array}$$

Right propagation

$$\begin{array}{lllll}
(d\textsc{r}): & P\widehat{\alpha} \nwarrow \widehat{x}\langle x.\beta\rangle & \to & P\widehat{\alpha} \dagger \widehat{x}\langle x.\beta\rangle & \\
(\textsc{r}1): & P\widehat{\alpha} \nwarrow \widehat{x}\langle y.\beta\rangle & \to & \langle y.\beta\rangle, & y \neq x \\
(\textsc{r}2): & P\widehat{\alpha} \nwarrow \widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) & \to & \widehat{y}(P\widehat{\alpha} \nwarrow \widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\textsc{r}3): & P\widehat{\alpha} \nwarrow \widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) & \to & P\widehat{\alpha} \dagger \widehat{z}((P\widehat{\alpha} \nwarrow \widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha} \nwarrow \widehat{x}R)), & z\ \mathit{fresh} \\
(\textsc{r}4): & P\widehat{\alpha} \nwarrow \widehat{x}(Q\widehat{\beta}\,[z]\,\widehat{y}R) & \to & (P\widehat{\alpha} \nwarrow \widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha} \nwarrow \widehat{x}R), & z \neq x \\
(\textsc{r}5): & P\widehat{\alpha} \nwarrow \widehat{x}(Q\widehat{\beta} \dagger \widehat{y}R) & \to & (P\widehat{\alpha} \nwarrow \widehat{x}Q)\widehat{\beta} \dagger \widehat{y}(P\widehat{\alpha} \nwarrow \widehat{x}R) &
\end{array}$$

Fig. 2. The propagation rules

2.3 Structural Congruences

By viewing $\mathcal{X}$ as a calculus, a natural questions to ask, and which has not been addressed in the past, is that if certain terms could be considered to be equivalne. To that purpose, we will define two congruences for † that look like associativity and commutativity.

Definition 6.

$$\begin{array}{rcll}
(\dagger\text{-}\mathit{assoc}): (P\widehat{\alpha} \dagger \widehat{x}Q)\widehat{\beta} \dagger \widehat{y}R & \overset{\mathcal{X}}{=} & P\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\beta} \dagger \widehat{y}R) & \text{if } \beta \notin \mathit{fp}(P)\ \&\ x \notin \mathit{fs}(R) \\
(\mathit{left\text{-}comm}): P\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\beta} \dagger \widehat{y}R) & \overset{\mathcal{X}}{=} & Q\widehat{\beta} \dagger \widehat{y}(P\widehat{\alpha} \dagger \widehat{x}R) & \text{if } x \notin \mathit{fs}(Q)\ \&\ y \notin \mathit{fs}(P) \\
(\mathit{right\text{-}comm}): (P\widehat{\alpha} \dagger \widehat{x}Q)\widehat{\beta} \dagger \widehat{y}R & \overset{\mathcal{X}}{=} & (P\widehat{\beta} \dagger \widehat{y}R)\widehat{\alpha} \dagger \widehat{x}Q & \text{if } \alpha \notin \mathit{fp}(R)\ \&\ \beta \notin \mathit{fp}(Q)
\end{array}$$

Rule $(\dagger\text{-}\mathit{assoc})$ allows us to write $P\widehat{\alpha} \dagger \widehat{x}Q\widehat{\beta} \dagger \widehat{y}R$ (provided the side-condition is fulfilled) since the order of the applications of cuts is irrelevant. Notice that the side-condition for this rule is the one we have indicated for (ins), and comes from the variable convention. This is consistent with the parenthesis-free notation we have used for the right-hand side of (ins). Notice that now writing $P\widehat{\alpha} \dagger \widehat{x}Q\widehat{\alpha} \dagger \widehat{x}R$ is licit. The second rule is left-commutativity and the third rule is right-commutativity.

There is another rule asserting the associativity of the mediators, given by:

$$(\mathit{med\text{-}assoc}):\ P\widehat{\alpha}\,[z]\,\widehat{x}(Q\widehat{\beta}\,[u]\,\widehat{y}R) \overset{\mathcal{X}}{=} (P\widehat{\alpha}\,[z]\,\widehat{x}Q)\widehat{\beta}\,[u]\,\widehat{y}R$$
$$\text{if } \beta \notin \mathit{fp}(P)\backslash\{\alpha\}, x \notin \mathit{fs}(R)\backslash\{y\}$$

Observe that, unsurprisingly, the side-condition is the same as for the cut. We will freely write $P\widehat{\alpha}\,[z]\,\widehat{x}Q\widehat{\beta}\,[u]\,\widehat{y}R$.

2.4 Call-by-Name and Call-by-Value

In this section we will define two sub-systems of reduction, that have a strong connection to call-by-value (CBV) and call-by-name (CBN) reduction. Notice that this is essentially different from the approach of [23], where, as in $\overline{\lambda}\mu\tilde{\mu}$, only one notion of reduction is defined; the CBN-CBV result there was obtained via different interpretation functions from CBN/CBV calculi.

As mentioned above, when P does not introduce α and Q does not introduce x, $P\widehat{\alpha} \dagger \widehat{x}Q$ is a *superposition*, meaning that two rules, namely (*act*-L) and (*act*-R), can both be fired which can lead to different irreducible terms: $\rightarrow$ is *not confluent*. The sub-systems of reduction we will introduce explicitly favour one kind of activating whenever the above critical pair occurs; these were shown to be confluent in [17] when restricted to typeable terms, and we conjecture that this result can be extended to untyped terms.

Definition 7.
- We write $P \rightarrow_{\text{V}} Q$ for the sub-reduction system that only activates a cut via $(act\text{-L})$ when it could be activated in two ways.
- Likewise, we write $P \rightarrow_{\text{N}} Q$ for the sub-reduction system that only activates such a cut via $(act\text{-R})$.

The reason to use the names CBV and CBN in fact comes from the fact that these systems successfully implement their counterparts in the λ-calculus (see Theorem 23). And, in fact, the use of the terminology CBV is justifiable, when at the same time calling *values* those circuits that introduce a plug. But, in contrast to the case for the λ-calculus, our two systems are really *dual*, and we actually should use a terminology like 'call-by-$*$', where '$*$' is a name for those circuits that introduce a socket. At the moment, there is no clear idea on what '$*$' should be, so we will use the (misnomer) CBN.

We will now state some basic properties, which essentially show that the calculus is well behaved. Recall that a term is pure if it contains no activated cuts.

Lemma 8 (Cancellation).

1. *$P\widehat{\alpha} \dagger \widehat{x}Q \rightarrow_{\text{V}} P$ if $\alpha \notin fp(P)$ and P is pure.*
2. *$P\widehat{\alpha} \dagger \widehat{x}Q \rightarrow_{\text{N}} Q$ if $x \notin fs(Q)$ and Q is pure.*

We will now show that a cut with a capsule leads to renaming.

Lemma 9 (Renaming).

1. *$P\widehat{\delta} \dagger \widehat{z}\langle z.\alpha\rangle \rightarrow P[\alpha/\delta]$, if P is pure.*
2. *$\langle z.\alpha\rangle\widehat{\alpha} \dagger \widehat{x}P \rightarrow P[z/x]$, if P is pure.*

These results motivate the extension (in both sub-systems) of the reduction rules, formulating new rules in the shape of the above results.

3 Expressing the Natural Numbers in $\mathcal{X}$

The example of expressing natural numbers into $\mathcal{X}$ that we will give in this section is interesting in two respects. Firstly, it shows how a basic structure can be embedded in $\mathcal{X}$. Secondly, it shows many features and among them α-conversion.

A natural number is represented in $\mathcal{X}$ by a sequence of capsules connected by mediating sockets, i.e., with the same used names.

We assume that natural numbers have two free sockets x and f and one free plug α, with x as entry socket, α as exit plug and f as mediating socket. We define 0 as $\langle x.\alpha\rangle$ and *succ*(N) as $N\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle$ where N itself is a natural number (which violates Barendregt's convention and should be removed in actual use). By induction it follows that x and α have both a unique occurrence in each natural number.

Lemma 10. *If N is a natural number, then $N\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle = \langle x.\alpha\rangle\widehat{\alpha}\,[f]\,\widehat{x}N$.*

Lemma 11. *If N_1 and N_2 are natural numbers, then ($\rightarrow_{\text{S}}$ is either $\rightarrow_{\text{V}}$ or $\rightarrow_{\text{N}}$)*

1. $(N_1\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle)\widehat{\alpha}\dagger\widehat{x}N_2 \;\rightarrow_{\text{S}}\; N_1\widehat{\alpha}\,[f]\,\widehat{x}N_2$
2. $N_1\widehat{\alpha}\dagger\widehat{x}(N_2\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle) \;\rightarrow_{\text{S}}\; N_1\widehat{\alpha}\,[f]\,\widehat{x}N_2$

Definition 12 (Addition and multiplication).

$$\mathit{add}(N_1,N_2) \;=\; N_1\widehat{\alpha}\dagger\widehat{x}N_2 \qquad \mathit{times}(N_1,N_2) \;=\; (\widehat{x}N_1\widehat{\alpha}\cdot\beta)\widehat{\beta}\dagger\widehat{f}N_2$$

Using this definition, we can show that the normal properties for addition hold.

Lemma 13 (Properties of *add*).

$$\begin{aligned}
\mathit{add}(\langle x.\alpha\rangle,N) &\rightarrow_{\text{A}} N\\
\mathit{add}(N,\langle x.\alpha\rangle) &\rightarrow_{\text{A}} N\\
\mathit{add}(N_1\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle,N_2) &\rightarrow_{\text{A}} \mathit{add}(N_1,N_2)\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle\\
\mathit{add}(N_1,N_2\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle) &\rightarrow_{\text{A}} \mathit{add}(N_1,N_2)\widehat{\alpha}\,[f]\,\widehat{x}\langle x.\alpha\rangle
\end{aligned}$$

From Lemma 13 we get, by induction:

$$\begin{aligned}
\mathit{add}(0,N) &= N \qquad & \mathit{add}(\mathit{succ}(N_1),N_2) &= \mathit{succ}(\mathit{add}(N_1,N_2))\\
\mathit{add}(N,0) &= N \qquad & \mathit{add}(\mathit{succ}(N_1),N_2) &= \mathit{succ}(\mathit{add}(N_1,N_2)).
\end{aligned}$$

From them we can prove:
$$\begin{aligned}
\mathit{add}(N_1,N_2) &= \mathit{add}(N_2,N_1)\\
\mathit{add}(N_1,\mathit{add}(N_2,N_3)) &= \mathit{add}(\mathit{add}(N_1,N_2),N_3).
\end{aligned}$$

The key point of the definition of *times* is that the *ins* rule copies N_2 into N_1 at each of the occurences of f.

Lemma 14 (Properties of *times*).

$$\begin{aligned}
\mathit{times}(N,0) &\rightarrow_{\text{A}} 0\\
\mathit{times}(N_1,\mathit{succ}(N_2)) &\rightarrow_{\text{A}} \mathit{add}(\mathit{times}(N_1,N_2),N_1)
\end{aligned}$$

4 Typing for $\mathcal{X}$

The notion of type assignment on $\mathcal{X}$ that we present in this section is the basic implicative system for Classical Logic (Gentzen system LK). The Curry-Howard property is easily achieved by erasing all term-information.

Definition 15 (Types and Contexts).

1. The set of types is defined by the grammar: $A, B ::= \varphi \mid A{\rightarrow}B$. The types considered in this paper are normally known as *simple* (or *Curry*) types.
2. A *context of sockets* Γ is a mapping from sockets to types, denoted as a finite set of *statements* $x{:}A$, such that the *subject* of the statements (x) are distinct. When we write Γ_1, Γ_2 we mean the union of Γ_1 and Γ_2 when Γ_1 and Γ_2 are coherent (if Γ_1 contains $x{:}A_1$ and Γ_2 contains $x{:}A_2$ then $A_1 = A_2$).
   Contexts of *plugs* Δ are defined in a similar way.

Definition 16 (Typing for $\mathcal{X}$).

1. *Type judgements* are expressed via a ternary relation $P \mathrel{:\cdot} \Gamma \vdash \Delta$, where Γ is a context of *sockets* and Δ is a context of *plugs*, and P is a circuit. We say that P is the *witness* of this judgement.
2. *Type assignment for $\mathcal{X}$* is defined by the following sequent calculus:

$$(\mathit{cap}) : \frac{}{\langle y.\alpha\rangle \mathrel{:\cdot} \Gamma, y{:}A \vdash \alpha{:}A, \Delta} \qquad (\mathit{med}) : \frac{P \mathrel{:\cdot} \Gamma \vdash \alpha{:}A, \Delta \quad Q \mathrel{:\cdot} \Gamma, x{:}B \vdash \Delta}{P\widehat{\alpha}\,[y]\,\widehat{x}Q \mathrel{:\cdot} \Gamma, y{:}A{\rightarrow}B \vdash \Delta}$$

$$(\mathit{exp}) : \frac{P \mathrel{:\cdot} \Gamma, x{:}A \vdash \alpha{:}B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta \mathrel{:\cdot} \Gamma \vdash \beta{:}A{\rightarrow}B, \Delta} \qquad (\mathit{cut}) : \frac{P \mathrel{:\cdot} \Gamma \vdash \alpha{:}A, \Delta \quad Q \mathrel{:\cdot} \Gamma, x{:}A \vdash \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q \mathrel{:\cdot} \Gamma \vdash \Delta}$$

   We write $P \mathrel{:\cdot} \Gamma \vdash \Delta$ if there exists a derivation for this judgement.

Γ and Δ carry the types of the free connectors in P, as unordered sets. There is no notion of type for P itself, instead the derivable statement shows how P is connectable.

We can now provide the type naturals.

– The type of natural numbers in $\mathcal{X}$ is $N \mathrel{:\cdot} x{:}A, f{:}A \rightarrow A \vdash \alpha{:}A$.

The soundness result of simple type assignment with respect to reduction is stated as usual:

Theorem 17 (Witness reduction).

1. *If $P \mathrel{:\cdot} \Gamma \vdash \Delta$, and $P \rightarrow Q$, then $Q \mathrel{:\cdot} \Gamma \vdash \Delta$.*
2. *If $P \mathrel{:\cdot} \Gamma \vdash \Delta$, and $P \overset{\mathcal{X}}{=} Q$, then $Q \mathrel{:\cdot} \Gamma \vdash \Delta$.*

Theorem 18 (Strong normalisation [22]). *If $P \mathrel{:\cdot} \Gamma \vdash \Delta$, then P is strongly normalising.*

5 Interpreting the λ-Calculus

In this section, we illustrate the expressive power of $\mathcal{X}$ by showing that we can faithfully interpreted the the λ-calculus [6], and in the following sections we will show a similar result for λ**x** and $\lambda\mu$. Using the notion of Curry type assignment, we will show that assignable types are preserved by the interpretation.

In part, the interpretation results could be seen as variants of similar results obtained by Curien and Herbelin in [9]. Indeed, we could have defined our mappings using the mappings of the λ-calculus and $\lambda\mu$ into $\overline{\lambda}\mu\tilde{\mu}$, and concatenating those to the mapping from $\overline{\lambda}\mu\tilde{\mu}$ to $\mathcal{X}$, but our encoding is more detailed and precise than that, and deals with explicit substitution as well. In fact, we will show that our interpretation encompasses CBV and CBN reduction, something that has not been achieved in [9], and will argue that $\mathcal{X}$ in fact does more than that, like expressing explicit substitution.

One should notice that for [9] the preservation of the CBV-evaluation and CBN-evaluation relies on two *distinct* translations of terms. For instance, the CBV- and CBN-the λ-calculus can both be encoded into CPS [2], and there it is clear that what accounts for the distinction CBV/CBN is the encodings themselves, and not the way CPS reduces the encoded terms.

So, when encoding the λ-calculus in $\overline{\lambda}\mu\tilde{\mu}$, the distinction between CBV and CBN mostly relies on Curien and Herbelin's two distinct encodings rather than the features of $\overline{\lambda}\mu\tilde{\mu}$ (the same holds for [23]). Whereas there the CBN-translation seems intuitive, they apparently need to twist it in a more complex way in order to give an accurate interpretation of the CBV-the λ-calculus, since the CBV-interpretation of a term M reduces to its CBN-interpretation. This is a bit disappointing since the CBN-encoding turns out to be more refined than the CBV-encoding, breaking the nice symmetry.

In contrast, in $\mathcal{X}$ we have no need of two separate interpretation functions, but will define only *one*. Combining this with the two sub-reduction systems $\rightarrow_{\mathrm{V}}$ and $\rightarrow_{\mathrm{N}}$ we can encode the the CBV- and CBN-the λ-calculus. We can compare this to what is done by Danos, Joinet, Shellinx, when they write (before section 3.1.2: "Note that choosing colours has nothing to do with imposing a strategy. We don not select redexes, but rather the way we want to reduce them ..."

We first define the direct encoding of the λ-calculus into $\mathcal{X}$:

Definition 19 (Interpretation of the λ-calculus in $\mathcal{X}$).

$$\begin{aligned}
[\![x]\!]^{\lambda}_{\alpha} &= \langle x.\alpha\rangle \\
[\![\lambda x.M]\!]^{\lambda}_{\alpha} &= \widehat{x}[\![M]\!]^{\lambda}_{\beta}\widehat{\beta}\cdot\alpha \\
[\![MN]\!]^{\lambda}_{\alpha} &= [\![M]\!]^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}([\![N]\!]^{\lambda}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle)
\end{aligned}$$

Observe that every sub-term of $[\![M]\!]^{\lambda}_{\alpha}$ has exactly one free plug.

Definition 20 (Curry type assignment for the λ-calculus).

$$(Ax):\ \frac{}{\Gamma, x{:}A \vdash_{\lambda} x : A} \qquad (\rightarrow I):\ \frac{\Gamma, x{:}A \vdash_{\lambda} M : B}{\Gamma \vdash_{\lambda} \lambda x.M : A\rightarrow B}$$

$$(\rightarrow E):\ \frac{\Gamma \vdash_{\lambda} M : A\rightarrow B \quad \Gamma \vdash_{\lambda} N : A}{\Gamma \vdash_{\lambda} MN : B}$$

We can now show that typeability is preserved by $[\![\cdot]\!]^{\lambda}_{\alpha}$:

Theorem 21. *If $\Gamma \vdash_{\lambda} M : A$, then $[\![M]\!]^{\lambda}_{\alpha} :\cdot \Gamma \vdash \alpha{:}A$.*

$$\begin{array}{lr}
\llbracket \Delta\Delta \rrbracket^{\lambda}_{\beta} \ \triangleq\ \llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \triangleq \\
(\widehat{x}\llbracket xx \rrbracket^{\lambda}_{\alpha}\widehat{\alpha}\cdot\delta)\widehat{\delta} \dagger \widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow\ (\mathit{ins}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{x}(\llbracket xx \rrbracket^{\lambda}_{\alpha}\widehat{\alpha} \dagger \widehat{y}\langle y.\beta\rangle) & \rightarrow\ (\mathit{9}\text{-}\mathbf{??}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{x}\llbracket xx \rrbracket^{\lambda}_{\beta} & \triangleq \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{x}(\langle x.\delta\rangle\widehat{\delta}\ [x]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow\ (\mathit{act}\text{-}\mathrm{R}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \nwarrow \widehat{x}(\langle x.\delta\rangle\widehat{\delta}\ [x]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow\ (\mathrm{R3}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{z}((\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \nwarrow \widehat{x}\langle x.\delta\rangle)\widehat{\delta}\ [z]\ \widehat{y}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \nwarrow \widehat{x}\langle y.\beta\rangle)) & \rightarrow\ (\mathrm{L1}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{z}((\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \nwarrow \widehat{x}\langle x.\delta\rangle)\widehat{\delta}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow\ (d\mathrm{R}\ \&\ \mathit{9}\text{-}\mathbf{??}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\delta}\widehat{\delta}\ [z]\ \widehat{y}\langle y.\alpha\rangle) & \triangleq\ \llbracket \Delta\Delta \rrbracket^{\lambda}_{\alpha}
\end{array}$$

Fig. 3. Reduction of the interpretation of the lambda term $(\lambda x.xx)(\lambda x.xx)$

When encoding the CBV-the λ-calculus, we also use the $\llbracket\cdot\rrbracket^{\lambda}_{\alpha}$ interpretation. And, in contrast to $\overline{\lambda}\mu\tilde{\mu}$, we can get an accurate interpretation of the CBV-the λ-calculus into $\mathcal{X}$ by using the $\rightarrow_{\mathrm{V}}$ system, which we can reformulate as the reduction system obtained by replacing rule $(\mathit{act}\text{-}\mathrm{R})$ by:

$$(\mathit{act}\text{-}\mathrm{R}') :\ P\widehat{\alpha} \dagger \widehat{x}Q \ \rightarrow\ P\widehat{\alpha} \nwarrow \widehat{x}Q,\ \textit{if } P \textit{ introduces } \alpha \textit{ and } Q \textit{ does not introduce } x$$

Lemma 22. $\llbracket N \rrbracket_{\delta}\widehat{\delta} \nwarrow \widehat{x}\llbracket M \rrbracket_{\alpha} \rightarrow_{\mathrm{A}} \llbracket M[N/x] \rrbracket_{\alpha}$.

Theorem 23 (Simulation of the λ-calculus).

1. *If* $M \rightarrow_{\mathrm{V}} N$ *then* $\llbracket M \rrbracket^{\lambda}_{\gamma} \rightarrow_{\mathrm{V}} \llbracket N \rrbracket^{\lambda}_{\gamma}$.
2. *If* $M \rightarrow_{\mathrm{N}} N$ *then* $\llbracket M \rrbracket^{\lambda}_{\gamma} \rightarrow_{\mathrm{N}} \llbracket N \rrbracket^{\lambda}_{\gamma}$.

Now notice that $(\lambda x.M)(PQ)$ is not an redex in the CBV-λ-calculus. We get

$$\llbracket (\lambda x.M)(PQ) \rrbracket^{\lambda}_{\alpha} \ \rightarrow\ (\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma} \dagger \widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle))\widehat{\gamma} \dagger \widehat{x}\llbracket M \rrbracket^{\lambda}_{\alpha}$$

In particular, γ is not introduced in the outer-most cut, so $(\mathit{act}\text{-}\mathrm{L})$ can be applied. What the call-by-value reduction should block, however, is that $(\mathit{act}\text{-}\mathrm{R}')$ can be applied; then the propagation of $\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma} \dagger \widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle)$ into $\llbracket M \rrbracket^{\lambda}_{\alpha}$ is blocked (which would produce $\llbracket M[(PQ)/x] \rrbracket^{\lambda}_{\alpha}$). Notice that we can only apply rule $(\mathit{act}\text{-}\mathrm{R}')$ if both $\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma} \dagger \widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle)$ introduces γ and $\llbracket M \rrbracket^{\lambda}_{\alpha}$ does not introduce x. This is not the case, since the first test fails.

On the other hand, if N is a λ-value (i.e. either a variable or an abstraction) then $\llbracket N \rrbracket^{\lambda}_{\alpha}$ introduces α (in fact, N is a value if and only if $\llbracket N \rrbracket^{\lambda}_{\alpha}$ introduces α). Then $\llbracket N \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{x}\llbracket M \rrbracket^{\lambda}_{\alpha}$ cannot be reduced by rule $(\mathit{act}\text{-}\mathrm{L})$, but by either rule $(\mathit{act}\text{-}\mathrm{R})$ or a logical rule. This enables the reduction

$$\llbracket N \rrbracket^{\lambda}_{\gamma}\widehat{\gamma} \dagger \widehat{x}\llbracket N \rrbracket^{\lambda}_{\alpha} \ \rightarrow_{\mathrm{V}}\ \llbracket N[M/x] \rrbracket^{\lambda}_{\alpha}.$$

So CBV-reduction for the λ-calculus is respected by the interpretation function, using $\rightarrow_{\mathrm{V}}$.

It is worthwhile to notice that the interpretation function $[\![\cdot]\!]^{\lambda}_{\alpha}$ does not generate a confluent sub-calculus. Indeed, we have both

$$\begin{array}{ll} [\![(\lambda x.xx)(yy)]\!]^{\lambda}_{\alpha} \to \langle y.\beta\rangle\widehat{\beta}\,[y]\,\widehat{x}(\langle x.\gamma\rangle\widehat{\gamma}\,[x]\,\widehat{v}\langle v.\alpha\rangle) & \text{and} \\ [\![(\lambda x.xx)(yy)]\!]^{\lambda}_{\alpha} \to \langle y.\beta\rangle\widehat{\beta}\,[y]\,\widehat{a}((\langle y.\gamma\rangle\widehat{\gamma}\,[y]\,\widehat{b}\langle b.\delta\rangle)\widehat{\delta}\,[a]\,\widehat{c}\langle c.\alpha\rangle) & \end{array}$$

both normal forms. This is of course not surprising, seen that $(\lambda x.xx)(yy)$ has different normal forms with respect to CBN and CBV reduction.

To conclude this section, and illustrate the expressive power of $\mathcal{X}$ as abstract machine for reduction, Figure 3 shows an infinite reduction sequence in $\mathcal{X}$.

6 Interpreting λx

We will now interpret a calculus of explicit substitutions, namely λ**x** [8], where any β-reduction of the λ-calculus can be split into several more atomic steps of computation. In this section we show that $\mathcal{X}$ has a fine level of atomicity as it simulates each reduction step by describing how the explicit substitutions interact with terms.

We briefly recall here the calculus λ**x**.

Definition 24 (λx). The syntax of λ**x** is an extension of that of the λ-calculus:

$$M \;::=\; x \mid \lambda x.M \mid M_1M_2 \mid M\langle x = N\rangle$$

The reduction relation is defined by the following rules

$$\begin{array}{rcll rcll} (\lambda x.M)P & \to & M\langle x{=}P\rangle & (\mathsf{B}) & x\langle x{=}P\rangle & \to & P & (\mathsf{VarI}) \\ (MN)\langle x{=}P\rangle & \to & M\langle x{=}P\rangle N\langle x{=}P\rangle & (\mathsf{App}) & y\langle x{=}P\rangle & \to & y & (\mathsf{VarK}) \\ (\lambda y.M)\langle x{=}P\rangle & \to & \lambda y.(M\langle x{=}P\rangle) & (\mathsf{Abs}) & M\langle x{=}P\rangle & \to & M, \text{ if } x \notin fv(M) & (\mathsf{gc}) \end{array}$$

Notice that the notion of reduction λ**x** is obtained by deleting rule (gc), and the notion of reduction $\lambda\mathbf{x}_{\mathsf{gc}}$ is obtained by deleting rule (VarK). The rule (gc) is called 'garbage collection', as it removes useless substitutions. We will write $\to_{\mathrm{x}}$ for either reduction system.

Definition 25 (Interpretation of λx in $\mathcal{X}$). We define $[\![\cdot]\!]^{\mathrm{x}}_{\alpha}$ as the interpretation $[\![\cdot]\!]^{\lambda}_{\alpha}$, by adding:

$$[\![M\langle x = N\rangle]\!]^{\mathrm{x}}_{\alpha} \;=\; [\![N]\!]^{\mathrm{x}}_{\beta}\widehat{\beta}\nwarrow\widehat{x}[\![M]\!]^{\mathrm{x}}_{\alpha}.$$

Now we show that the reductions can be simulated, preserving the evaluation strategies. Our notion of CBV-λ**x** is naturally inspired by that of the λ-calculus: in a CBV-β-reduction, the argument must be a value, so that means that when it is simulated by CBV-λ**x**, all the substitutions created are of the form $M\langle x = N\rangle$ where N is a value, that is, either a variable or an abstraction, just as in the λ-calculus. Hence, we build the CBV-λ**x** by a syntactic restriction:

$$M ::= x \mid \lambda x.M \mid M_1M_2 \mid M\langle x = \lambda x.N\rangle \mid M\langle x = y\rangle.$$

Now notice that, again, N is a value if and only if $[\![N]\!]^{\mathrm{x}}_{\alpha}$ introduces α.

Theorem 26 (Simulation of rule (B)).

CBN: $[\![(\lambda x.M)N]\!]^{\mathrm{X}}_{\alpha} \rightarrow_{\mathrm{N}} [\![M\langle x = N\rangle]\!]^{\mathrm{X}}_{\alpha}$
CBV: $[\![(\lambda x.M)N]\!]^{\mathrm{X}}_{\alpha} \rightarrow_{\mathrm{V}} [\![M\langle x = N\rangle]\!]^{\mathrm{X}}_{\alpha}$ *iff N is a value.*

Theorem 27 (Simulation of the other rules). *Let $M \rightarrow N$ by any of the rules (App), (Abs), (VarI), (VarK), (gc), then $[\![M]\!]^{\mathrm{X}}_{\gamma} \rightarrow_{\mathrm{V}} [\![N]\!]^{\mathrm{X}}_{\gamma}$ and $[\![M]\!]^{\mathrm{X}}_{\gamma} \rightarrow_{\mathrm{N}} [\![N]\!]^{\mathrm{X}}_{\gamma}$.*

We can now state that $\lambda\mathbf{x}$-reduction is preserved by interpretation of terms into $\mathcal{X}$.

Theorem 28 (Simulation of λx).

1. *If $M \rightarrow_{\mathrm{V}} N$ then $[\![M]\!]^{\mathrm{X}}_{\gamma} \rightarrow_{\mathrm{V}} [\![N]\!]^{\mathrm{X}}_{\gamma}$*
2. *If $M \rightarrow_{\mathrm{N}} N$ then $[\![M]\!]^{\mathrm{X}}_{\gamma} \rightarrow_{\mathrm{N}} [\![N]\!]^{\mathrm{X}}_{\gamma}$*

7 Interpreting $\lambda\mu$

Parigot's $\lambda\mu$-calculus [21] is yet another proof-term syntax for classical logic, but expressed in the setting of Natural Deduction. Curien and Herbelin [9] have shown how the normalisation in $\lambda\mu$ can be interpreted as the cut-elimination in $\overline{\lambda}\mu\tilde{\mu}$. Using that mapping, and the interpretation of $\overline{\lambda}\mu\tilde{\mu}$ into $\mathcal{X}$ from [17], we can generate the following:

Definition 29 (Interpretation of $\lambda\mu$ in $\mathcal{X}$). We define $[\![\cdot]\!]^{\lambda\mu}_{\alpha}$ as the interpretation $[\![\cdot]\!]^{\lambda}_{\alpha}$, by adding:

$$[\![\mu\delta.[\gamma]M]\!]^{\lambda\mu}_{\alpha} = [\![M]\!]^{\lambda\mu}_{\gamma}\widehat{\delta} \dagger \widehat{x}\langle x.\alpha\rangle$$

Similarly to the previous sections, we can add:

$$\begin{aligned} [\![M[N/x]]\!]^{\lambda\mu}_{\alpha} &= [\![N]\!]^{\lambda\mu}_{\beta}\widehat{\beta} \dagger \widehat{x}[\![M]\!]^{\lambda\mu}_{\alpha} \\ [\![(\mu\delta.[\gamma]M)[N{\cdot}\delta/\delta]]\!]^{\lambda\mu}_{\alpha} &= [\![M]\!]^{\lambda\mu}_{\gamma}\widehat{\delta} \dagger \widehat{x}([\![N]\!]^{\lambda\mu}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle) \end{aligned}$$

Notice that the last alternative is justified, since

Lemma 30. *The following rule is admissible:*

$$[\![(\mu\delta.[\gamma]M))N]\!]^{\lambda\mu}_{\alpha} \rightarrow [\![M]\!]^{\lambda\mu}_{\gamma}\widehat{\delta} \dagger \widehat{x}([\![N]\!]^{\lambda\mu}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle)$$

Notice also the striking similarity between $[\![MN]\!]^{\lambda\mu}_{\alpha}$ and the result of running $[\![(\mu\delta.[\gamma]M))N]\!]^{\lambda\mu}_{\alpha}$; the difference lies only in a bound socket.

The main result for this interpretation now becomes:

Theorem 31 (Simulation of $\lambda\mu$ in $\mathcal{X}$).

1. *If $M \rightarrow_{\mathrm{V}} N$ then $[\![M]\!]^{\lambda\mu}_{\alpha} \rightarrow_{\mathrm{V}} [\![N]\!]^{\lambda\mu}_{\alpha}$.*
2. *If $M \rightarrow_{\mathrm{N}} N$ then $[\![M]\!]^{\lambda\mu}_{\alpha} \rightarrow_{\mathrm{N}} [\![N]\!]^{\lambda\mu}_{\alpha}$.*

8 Conclusions and Future Work

We have seen that $\mathcal{X}$ is a continuation-style formal language that provides a Curry-Howard-de Bruijn isomorphism for a sequent calculus for implicative classical logic. But, of more interest, we have seen $\mathcal{X}$ is very well-suited as generic abstract machine for the running of (applicative) programming languages, by building not only an interpretation for λ, $\lambda\mu$ (for $\overline{\lambda}\mu\tilde{\mu}$, see [17]), but also for $\lambda\mathbf{x}$.

A wealth of research lies in the future, of which this paper is but the first step, the seed. We intend to study normalisation, and confluence of the CBN and CBV strategies, to extend $\mathcal{X}$ in order to represent the other logical connectives, study the relation with linear logic, proofnets (both typed and untyped), the relation with π-calculus, how to express recursion, functions, etc, etc.

Acknowledgements

We would like to thank Alexander Summers, Daniel Hirschkoff, Dragisa Zunic, Harry Mairson, Jamie Gabbay, Jayshan Raghunandan, Luca Cardelli, Luca Roversi, Maria Grazia Vigliotti, Mariangiola Dezani-Ciancaglini, Philippe Audebaud, and Simona Ronchi della Rocca for many fruitful discussions on the topic of this paper.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. A. W. Appel and T. Jim. Continuation-passing, closure-passing style. In *POPL'89*, pages 293–302, 1989.
3. S. van Bakel and J. Raghunandan. Implementing $\mathcal{X}$. In *TermGraph'04*, ENTCS, 2005.
4. S. van Bakel, J. Raghunandan, and A. Summers. Term Graphs, α-conversion and Principal Types for $\mathcal{X}$, 2005.
5. F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, 1996.
6. H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, 1984.
7. H. P. Barendregt and S. Ghilezan. Lambda terms for natural deduction, sequent calculus and cut-elimination. *Journal of Functional Porgramming*, 10(1):121–134, 2000.
8. R. Bloo and K.H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN'95*, pages 62–72, 1995.
9. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP'00*, pages 233–243, 2000.
10. Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Computational isomorphisms in classical logic (extended abstract). *ENTCS* 3, 1996.
11. Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: Linear logic. *The Journal of Symbolic Logic*, 62, 1997.
12. A. G. Dragalin. *Mathematical Intuitionism: Introduction to Proof Theory*, volume 67 of *Translations of Mathematical Monographs*. American Mathematical Society, 1987.
13. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
14. J.-Y. Girard. A new constrcutive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

15. H. Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes.* Thèse d'université, Université Paris 7, 1995.
16. S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.
17. Stéphane Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *ENTCS*, volume 86. Elsevier, 2003.
18. P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In *POPL'94*, pages 60–69. ACM, 1994.
19. G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935. English translation in [20], pages 68–131.
20. M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1969.
21. M. Parigot. An algorithmic interpretation of classical natural deduction. In *LPAR'92*, LNCS 624, pages 190–201, 1992.
22. Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, 2000.
23. Philip Wadler. Call-by-Value is Dual to Call-by-Name. In *ICFP'03*, pages 189 – 201, 2003.
24. A N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, 1925.
25. A. N. Whitehead and B. Russell. *Principia Mathematica to *56.* Cambridge University Press, 1997.

Checking Risky Events Is Enough for Local Policies

Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari

Dipartimento di Informatica, Università di Pisa, Italy
{bartolet, degano, giangi}@di.unipi.it

Abstract. An extension of the λ-calculus is proposed to study history-based access control. It allows for parametrized security policies with a possibly nested, local scope. To govern the rich interplay between local policies, we propose a combination of static analysis and dynamic checking. A type and effect system extracts from programs a correct approximation to the histories obtainable at run-time. A further static analysis over these approximations determines how to instrument code so to enforce the desired security constraints. The execution monitor, based on finite-state automata, runs efficiently the instrumented code.

1 Introduction

Access control is crucial for securing the execution of mobile applications. Access control policies specify which operations can be executed by possibly untrusted components on sensible resources such as files, communication channels, and so on. Indeed, an attacker may gain full control over a system by getting improper access to critical resources.

Current software technologies enforce access control policies by exploiting different mechanisms. In the Java Virtual Machine, as well as in the .Net Common Language Runtime, *stack inspection* computes the run-time access rights of code by examining the stack of method invocations. Code includes special time-consuming instructions, called *local security checks*, that guard access to critical resources. Methods are associated with a static set of permissions, to reflect the trustedness of code. At run-time, a resource access is granted whenever all methods in the call stack have the required permission.

Stack inspection and local security checks offer a pragmatic setting for access control, with a strong bias towards implementation. However, they suffer from two main shortcomings. First, it is difficult to place the needed checks at the relevant points in the code, and even more difficult is guaranteeing that they suffice for enforcing the intended security policy. Second, stack inspection may fail to enforce some security constraints, because it relies on the call stack only. Indeed, the access rights of a certain method are no longer affected by the execution of an untrusted one, after it has been popped from the call stack. This may be harmful, e.g. when trusted code depends on the results supplied by untrusted code [12]. In other words, stack inspection and local security checks are not an appropriate abstraction for security.

Some alternatives to stack inspection have been explored. In *history-based access control*, the actual access rights of a piece of code depend on (a suitable abstraction of) the *whole* execution. This approach has been receiving major attention, at both levels of foundations [2,4,11,19] and of language design and implementation [1,9]. Another paradigm proposes to encapsulate code within wrappers that enforce *local security policies* [18]. A *wrapper* monitors the execution and aborts it when about to violate some active local policy.

We have recently proposed $\lambda^{[]}$, an extension of the λ-calculus that reconciles history-based access control with local policies [4]. Local security properties are regular properties of histories and have a possibly nested, local scope. A *policy framing* $\varphi[e]$ indicates that the program e is protected by the local policy φ, i.e. the history must always respect φ while evaluating e.

In [4] we have verified local security policies through a static analysis, based on a type and effect system [20]. The effect approximates the run-time behaviour of a program. Model checking the effect ensures that there will be no security violations at run-time. The static approach avoids the need for an execution monitor to enforce local policies. Also, it avoids keeping track of the whole execution history, which may grow unbound.

Because of the static approximation, some programs may be discarded even though they would be secure, possibly just by adding a limited amount of run-time checking. We propose here a mixed approach to access control, that efficiently combines static analysis and run-time checking. Our technique discards no programs while keeping security. Intuitively, we compile a program with policy framings into an equivalent one without framings, but instrumented with local checks. Our static analysis determines *which* checks are needed and *where* they must be inserted to obtain a program respecting the given security requirements. The execution monitor is essentially a finite-state automaton associated with the relevant security policies.

Technically, we define a novel type and effect system; the types are mostly standard and the effects are *history grammars*. These are context-free grammars representing all possible execution histories, including the scopes of the local policies. A further static analysis over history grammars, called *risky events analysis*, determines the program points where security violations may occur. The risky events analysis is then exploited to generate the set of checks needed to safely instrument the original code at the right points. The original and instrumented programs indeed satisfy the same security constraints. It is worthwhile noting that the execution monitor controlling the instrumented program only needs a finite amount of information. More precisely, it suffices to record the states of the finite-state automata associated with checks on risky events.

2 A Motivating Example

To illustrate our approach, consider a simple web browser that runs applets. Each applet must obey a general usage policy φ: only open files can be read. Moreover, a user can supply the browser with his own security policy, to be enforced on all applet executions. In this example, the user policy says that, after having read a local file, an applet can no longer connect to the network.

The browser is a function that processes the applet x and the user policy φ'. Since policies are not first class expressions in our calculus, the parameter φ' is rendered as a closure $p = \lambda y.\, \varphi'[y\, *]$, where $*$ stands for the unit value.

$$\mathit{Browser} \;=\; \lambda x.\, \lambda p.\, \varphi[p\; x]; \mathit{Browser}$$

We consider a simple editor applet, that accesses a local file y and may save it on a given site, identified by a URL u:

$$\mathit{Editor} \;=\; \lambda y.\, \lambda u.\; \mathit{open}(y); \mathit{read}(y); *;\; \big(\texttt{if}\; b\; \texttt{then}\; \mathit{connect}(u)\big);\; *; \mathit{close}(y)$$

Note that our editor is overly simplified, because we are only interested in the security-relevant events it can generate; the unit value $*$ indeed represents the sequences of operations that do not affect security.

We also have an attacker applet, that tries to spoof the browser by executing it with a very liberal user policy $\lambda y.\, y*$ which imposes no constraints. The attacker attempts to force the editor reading the secret password file `pwd` and then saving it on the attacker site `evil.org`.

$$\mathit{Attacker} \;=\; \lambda w.\; \mathit{Browser}\; \big(\lambda z.\, \mathit{Editor}\; \texttt{pwd evil.org}\big)\; (\lambda y.\, y*)$$

The following trace illustrates the behaviour of the attacker applet, sandboxed with the user policy φ' by the browser. The program states are pairs, whose first component is a history (a sequence of access events – ε denotes the empty one) and the second component is the program continuation.

$$\begin{array}{ll}
 & \varepsilon,\; \mathit{Browser}\; \mathit{Attacker}\; (\lambda y.\varphi'[y*]) \\
\to & \varepsilon,\; \varphi[(\lambda y.\, \varphi'[y\, *])\; \mathit{Attacker}]; \mathit{Browser} \\
\to & \varepsilon,\; \varphi[\varphi'[\mathit{Attacker}\, *]]; \mathit{Browser} \\
\to & \varepsilon,\; \varphi[\varphi'[\mathit{Browser}\; (\lambda z.\, \mathit{Editor}\; \texttt{pwd evil.org})\; (\lambda y.\, y\, *)]]; \mathit{Browser} \\
\to & \varepsilon,\; \varphi[\varphi'[\varphi[(\lambda y.\, y\, *)\; (\lambda z.\, \mathit{Editor}\; \texttt{pwd evil.org})]; \mathit{Browser}]]; \mathit{Browser} \\
\to & \varepsilon,\; \varphi[\varphi'[\varphi[\mathit{Editor}\; \texttt{pwd evil.org})]; \mathit{Browser}]]; \mathit{Browser} \\
\to & \varepsilon,\; \varphi[\varphi'[\varphi[\mathit{open}(\texttt{pwd}); \mathit{read}(\texttt{pwd}); *; \texttt{if}\; b\; \texttt{then}\; \mathit{connect}(\texttt{evil.org}); \cdots] \cdots \\
\to & \mathit{open}(\texttt{pwd}),\; \varphi[\varphi'[\varphi[\mathit{read}(\texttt{pwd}); *; \texttt{if}\; b\; \texttt{then}\; \mathit{connect}(\texttt{evil.org}); \cdots] \cdots \\
\to & \mathit{open}(\texttt{pwd})\mathit{read}(\texttt{pwd}),\; \varphi[\varphi'[\varphi[\texttt{if}\; b\; \texttt{then}\; \mathit{connect}(\texttt{evil.org}); *; \mathit{close}(\texttt{pwd})] \cdots
\end{array}$$

If b evaluates to true, then a security exception is thrown, because the history:

$$\mathit{open}(\texttt{pwd})\;\; \mathit{read}(\texttt{pwd})\;\; \mathit{connect}(\texttt{evil.org})$$

would violate the active policy φ'. Note however that the connect event in the editor is the *only* program point where the policy φ' is violated. Instead, *all* possible runs obey the file usage policy φ. These observations suggest us to implement the access control mechanism by inserting local checks just before *risky events*, i.e. the program points about to violate policies. In our example, the original editor will be transformed into the equivalent:

$$Editor' = \lambda x.\, \lambda u.\, open(x); read(x); *; \\ \big(\texttt{if}\, b\, \texttt{then}\, \texttt{check}\ \varphi'\ \texttt{in}\ connect(u)\big); *; close(x)$$

Furthermore, all the policy framings are removed, and expressions are instrumented to record the set of active policies. Our goal is finding therefore the risky program points and the involved policies.

3 The Language $\boldsymbol{\lambda}^{[]}$

We consider a call-by-value λ-calculus enriched with access events and local security policies. This language is called $\lambda^{[]}$, and has been first introduced in [4]. We present here an extension of the calculus that features *parametrized* access events, and a new type and effect system that is suitable for instrumentation with local checks. An *access event* $\alpha^\ell(c)$ abstracts from a security-relevant operation. The symbol $\alpha \in \mathsf{Act}$ stands for an action (e.g. reading a file), while the parameter $c \in \mathsf{Res}$ is the resource upon which the action is taken (e.g. a file name). The label $\ell \in \mathsf{Lab}$ uniquely identifies an access event in an expression. We assume the sets Act, Res and Lab to be pairwise disjoint.

Sequences η of access events are called *histories*. Security policies $\varphi \in \Pi$ are regular properties of histories, universally quantified over resources. An instantiated policy $\varphi(c)$ can be phrased as a regular expression, and enforced by a finite state automaton (see below for details). A *policy framing* $\varphi[e]$ localizes the scope of the policy φ to the expression e; framings can be arbitrarily nested. To enhance readability, our calculus comprises conditional expressions and named abstractions (z in $e' = \lambda_z x.e$ stands for e' itself within e). We omit the definition of guards b, as they are not relevant for the subsequent technical development.

$\lambda^{[]}$ **expressions**

$e, e' ::= x$	variable
c	constant
$\alpha^\ell(r)$	access event
$\texttt{if}\, b\, \texttt{then}\, e\, \texttt{else}\, e'$	conditional
$\lambda_z x.\, e$	abstraction
$e\, e'$	application
$\varphi[e]$	policy framing

Variables, constants, abstractions, and *failures* are the values v of $\lambda^{[]}$. A failure $\mathit{fail}_{\ell,\varphi(c)}$ represents a computation that is about to violate the policy $\varphi(c)$ by generating an event $\alpha^\ell(c')$. We assume that, for any expression e, label ℓ and policy $\varphi(c)$, $e\, \mathit{fail}_{\ell,\varphi(c)} = \mathit{fail}_{\ell,\varphi(c)}\, e = \varphi'[\mathit{fail}_{\ell,\varphi(c)}] = \mathit{fail}_{\ell,\varphi(c)}$. We write $*$ for a fixed, closed, event-free, non-failure value, and $\lambda.\, e$ for $\lambda x.\, e$, for $x \notin fv(e)$. The following abbreviation is standard: $e; e' = (\lambda.\, e')\, e$. Also, we write α instead of

$\alpha(c)$ when the resource parameter c is immaterial. Without loss of generality, we assume that each framing has an opening event, i.e. for all $\varphi[e]$, the expression e is of the form $\alpha(c); e'$, for some α, c and e'. This opening event can be a dummy event with no influence on security. Note that we also allow for parametrized events $\alpha^{\ell}(r)$, where r ranges over resources and variables.

We define the behaviour of $\lambda^{[]}$ expressions through the following small-step operational semantics. The configurations are pairs η, e, where e may denote both expressions and failures. A transition $\eta, e \to \eta', e'$ means that, starting from a history η, the expression e may evolve to e', possibly extending η to η'. We write $\eta \models \varphi(c)$ when the history η satisfies the policy $\varphi(c)$. We assume as given a total function $\mathcal{B}$ that evaluates the guards in conditionals, and a function $\mathit{lab}(\eta)$ that returns the last label ℓ_k from a history $\eta = \alpha_1^{\ell_1}(c_1) \cdots \alpha_k^{\ell_k}(c_k)$. Notice that there is no need to define $\ell(\varepsilon)$, because of the assumed opening event.

Operational semantics of $\lambda^{[]}$

$$\frac{\eta, e_1 \to \eta', e_1'}{\eta, e_1 e_2 \to \eta', e_1' e_2} \qquad \frac{\eta, e_2 \to \eta', e_2'}{\eta, v e_2 \to \eta', v e_2'} \qquad \eta, (\lambda_z x.e)v \to \eta, e\{v/x, \lambda_z x.e/z\}$$

$$\eta, \alpha^{\ell}(c) \to \eta\, \alpha^{\ell}(c), * \qquad \eta, \texttt{if}\ b\ \texttt{then}\ e_0\ \texttt{else}\ e_1 \to \eta, e_{\mathcal{B}(b)}$$

$$\frac{\eta, e \to \eta', e' \quad \forall c.\, \eta' \models \varphi(c)}{\eta, \varphi[e] \to \eta', \varphi[e']} \qquad \frac{\eta, e \to \eta', e' \quad \exists c.\, \eta' \not\models \varphi(c)}{\eta, \varphi[e] \to \eta, \mathit{fail}_{\mathit{lab}(\eta\ ), \varphi(c)}} \qquad \eta, \varphi[v] \to \eta, v$$

The rules above are mostly standard, except for those governing evaluation within policy framings. An expression $\varphi[e]$ can evolve to $\varphi[e']$, provided that the resulting history η' satisfies *all* the possible instantiations $\varphi(c)$; a failure occurs when $\varphi(c)$ is violated for some c. Eventually, values leave the scope of policies. More concrete operational semantics could reduce the number of instantiated policies, e.g. by looking only at the constants occurring in η', or by selecting constants with a suitable type, like file, memory region, etc.

A security policy $\varphi(c)$ is specified by a deterministic finite-state automaton $A_{\varphi(c)} = (\Sigma, Q, q_0, \delta)$, where $\Sigma = \mathsf{Act} \times \mathsf{Res}$ is the input alphabet, Q is the set of states, $q_0 \in Q$ is the start state, and $\delta : Q \times \Sigma \to Q$ is the transition function. Without loss of generality, we assume a distinguished state $q_s \in Q$ which is the only non-accepting state and is a *sink*, i.e. $\delta(q_s, \beta) = q_s$ for any $\beta \in \Sigma$. Note that $\eta \models \varphi(c)$ actually means that $\delta(q_0, \eta) \neq q_s$ in the automaton $A_{\varphi(c)}$.

Example 1. To keep small the size of our example, it is convenient to slightly simplify the browser/applet system considered in Section 2.

$$\begin{aligned} B &= \lambda_z x.\, \varphi'[x\, *]; z\, x \\ A &= \lambda.\, \texttt{if}\ b\ \texttt{then}\ \alpha_{read}(c)\ \texttt{else}\ \alpha_{connect} \\ e &= \varphi[\alpha_{open}(c); BA; \alpha_{close}(c)] \end{aligned}$$

Let ℓ be the label of the event $\alpha_{connect}$, and assume first that b is true. The evaluation of e goes as follows (for readability, the event labels are omitted):

$$\begin{aligned}
\varepsilon, e &\to \alpha_{open}(c), \varphi[BA; \alpha_{close}(c)]\\
&\to \alpha_{open}(c), \varphi[\varphi'[A\,*]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c), \varphi[\varphi'[\texttt{if}\ b\ \texttt{then}\ \alpha_{read}(c)\ \texttt{else}\ \alpha_{connect}]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c), \varphi[\varphi'[\alpha_{read}(c)]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[\varphi'[*]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[*; B\,A; \alpha_{close}(c)]
\end{aligned}$$

Assume now that b becomes false. Then, the computation proceeds as follows:

$$\begin{aligned}
\cdots &\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[\varphi'[A\,*]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[\varphi'[\texttt{if}\ b\ \texttt{then}\ \alpha_{read}(c)\ \texttt{else}\ \alpha_{connect}]; B\,A; \alpha_{close}(c)]\\
&\to \alpha_{open}(c)\,\alpha_{read}(c), \varphi[\varphi'[\alpha_{connect}]; B\,A; \alpha_{close}(c)]\\
&\to \mathit{fail}_{\ell,\varphi\ (c)}
\end{aligned}$$

The computation fails, because generating the event $\alpha_{connect}$ would make the history $\alpha_{open}(c)\,\alpha_{read}(c)\,\alpha_{connect}$ violate the policy $\varphi'(c)$.

Figure 1 displays the universally quantified automata for the usage policy φ and the user policy φ' introduced in Section 2. We only draw the arcs labelled with actions relevant to the policy in hand, while the other actions are intended to originate self-loops, omitted in the figure. The actual security automata are obtained by instantiating the parameter x to a resource c. The sink state q_s is shared by both automata, while q_0 and q_0' are the start states.

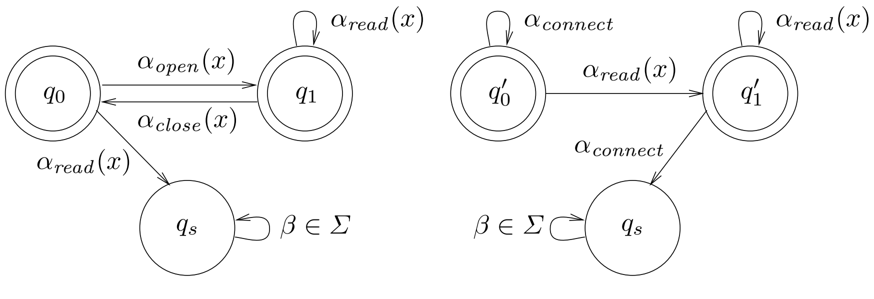


Fig. 1. Security automata $A_{\varphi(x)}$ (left) and $A_{\varphi\ (x)}$ (right)

4 Extracting History Grammars

We now introduce a type and effect system for $\lambda^{[]}$, that we shall use for approximating the aspects of the expression behaviour relevant to security.

Given an expression e, it is convenient to label each of its subexpressions. For simplicity, we shall use injective labellings only, and we coherently extend the labelling of an event $\alpha^\ell(r)$ to $\alpha^\ell(r^\ell)$. To keep track of the framings, we shall use the special *framing events* $[_\varphi$ and $]_\varphi$ that stand respectively for opening and closing the scope of the policy φ. We denote with Frm the set $\{ [_\varphi,]_\varphi \mid \varphi \in \Pi \}$.

Types τ and type environments Γ are defined as follows, where $R \subseteq \mathsf{Res}$.

Types and Type Environments

$$\tau \; ::= \; unit \mid R \mid \tau \xrightarrow{H} \tau \qquad \Gamma \; ::= \; \emptyset \mid \Gamma; x : \tau \quad (x \notin dom(\Gamma))$$

The types are much the same of the implicitly-typed λ-calculus (the set type R is used for resources), while the effects are our *history grammars.* These are context-free grammars $H = (T, N, \ell, P)$, where $T = \mathsf{Act} \cup \mathsf{Res} \cup \mathsf{Frm}$ is the set of terminal symbols, $N = \mathsf{Lab}$ is the set of non-terminals, $\ell \in N$ is the start symbol, and $P \subseteq N \times (T \cup N)^*$ is the set of productions. We shall use the following standard notation: (ℓ, P) for H, $\Rightarrow$ for the derivation relation, $\mathcal{L}(H)$ for the language generated by H, and ε for the empty string.

The effect H of an expression e represents all the possible run-time histories of e, and also records entering and exiting from the scope of security policies.

Example 2. Consider the following (labelled) expression:

$$e^0 \; = \; \alpha^1(c^2); \varphi\big[\big(\texttt{if}\, b\, \texttt{then}\, \alpha'^5(c'^6)\; \texttt{else}\; *^7\big)^4\big]^3$$

The history grammar extracted from e is as follows (the underlined label stands for the start symbol).

$$\begin{array}{ll} \underline{0} \to 1\;3 & 4 \to 5 \mid 7 \\ 1 \to \alpha\; c & 5 \to \alpha'\; c' \\ 2 \to \varepsilon & 6 \to \varepsilon \\ 3 \to [_\varphi\, 4\,]_\varphi & 7 \to \varepsilon \end{array}$$

We compare below a computation of e (left) with a derivation of H (right), assuming b true and φ always respected.

$$\begin{array}{ll} \varepsilon,\; e \to \alpha(c),\; \varphi[\texttt{if}\, b\, \texttt{then}\, \alpha'(c')\; \texttt{else}\, *] & \underline{0} \Rightarrow 1\;3 \Rightarrow \alpha\; c\; 3 \\ \quad\to \alpha(c),\; \varphi[\alpha'(c')] & \quad\Rightarrow \alpha\; c\; [_\varphi\, 4\,]_\varphi \Rightarrow \alpha\; c\; [_\varphi\, 5\,]_\varphi \\ \quad\to \alpha(c)\, \alpha'(c'),\; \varphi[*] & \quad\Rightarrow \alpha\; c\; [_\varphi\, \alpha'\; c'\,]_\varphi \end{array}$$

The framing events in $\alpha\; c\; [_\varphi\, \alpha'\; c]_\varphi$ mean that the policy φ is active when the event $\alpha'(c')$ is generated, so the history $\alpha(c)\, \alpha'(c')$ must obey φ, otherwise the computation fails.

A typing judgment $\Gamma, H \vdash e : \tau$ means that the expression e evaluates to a value of type τ, and produces a history represented by the effect H. The history grammar H in the functional type $\tau \xrightarrow{H} \tau'$ describes the latent effect associated with an abstraction, i.e. one of the histories in $\mathcal{L}(H)$ is generated when applying that abstraction to a value. The relation $\Gamma, H \vdash e : \tau$ is the least one closed under the following rules. For brevity, we shall omit immaterial labels, and we shall write $\underline{\ell} \to \gamma, P$ for $(\ell, \{\ell \to \gamma\} \cup P)$, and $\ell \to \gamma \mid \gamma'$ when $\ell \to \gamma$ and $\ell \to \gamma'$.

Type and Effect System for $\lambda^{[]}$

$$\frac{R \subseteq \mathsf{Res}}{\Gamma, \underline{\ell} \to \varepsilon \vdash c^{\ell} : \{c\} \cup R} \qquad \frac{\Gamma, (\ell', P) \vdash r : R}{\Gamma, \{\underline{\ell} \to \alpha\, c \;:\; c \in R\}, P \vdash \alpha^{\ell}(r^{\ell'}) : \mathit{unit}}$$

$$\frac{}{\Gamma, \underline{\ell} \to \varepsilon \vdash x^{\ell} : \Gamma(x)} \qquad \frac{\Gamma; x : \tau; z : \tau \xrightarrow{(\ell',P)} \tau', (\ell', P) \vdash e : \tau'}{\Gamma, \underline{\ell} \to \varepsilon \vdash (\lambda_z x.e)^{\ell} : \tau \xrightarrow{(\ell',P)} \tau'}$$

$$\frac{}{\Gamma, \underline{\ell} \to \varepsilon \vdash *^{\ell} : \mathit{unit}} \qquad \frac{\Gamma, (\ell_0, P_0) \vdash e_0 : \tau \xrightarrow{(\ell_2,P_2)} \tau' \quad \Gamma, (\ell_1, P_1) \vdash e_1 : \tau}{\Gamma, \underline{\ell} \to \ell_0\ell_1\ell_2, P_0 \cup P_1 \cup P_2 \vdash (e_0 e_1)^{\ell} : \tau'}$$

$$\frac{\Gamma, (\ell_0, P_0) \vdash e_0 : \tau \quad \Gamma, (\ell_1, P_1) \vdash e_1 : \tau}{\Gamma, \underline{\ell} \to \ell_0 \mid \ell_1, P_0 \cup P_1 \vdash (\mathtt{if}\, b\, \mathtt{then}\, e_0\, \mathtt{else}\, e_1)^{\ell} : \tau}$$

$$\frac{\Gamma, (\ell', P') \vdash e : \tau}{\Gamma, \underline{\ell} \to [_{\varphi}\, \ell'\,]_{\varphi}, P' \vdash \varphi[e]^{\ell} : \tau} \qquad \frac{\Gamma, H \vdash e : \tau}{\Gamma, H' \vdash e : \tau} \;\; \mathcal{L}(H') \supseteq \mathcal{L}(H)$$

Typing judgments are standard, and allow for weakening of both resource types (in the first rule) and effects (in the last rule). The effects in the rule for application are concatenated according to the evaluation order of the call-by-value semantics (function, argument, latent effect). The actual effect of an abstraction is the empty language, while the latent effect is equal to the actual effect of the function body.

Example 3. To clarify the role of weakening, consider the following expressions:

$$e_1 = \big((\lambda x.\, \alpha^3(x^4))^2\, (\mathtt{if}\, b\, \mathtt{then}\, c^6\, \mathtt{else}\, c'^{\,7})^5\big)^1$$
$$e_2 = \big(\mathtt{if}\, b\, \mathtt{then}\, (\lambda x.\, \alpha^{13}(c^{14}))^{12}\, \mathtt{else}\, (\lambda y.\, \alpha'^{\,16}(c^{17}))^{15}\big)^{11}$$

For e_1, let $\Gamma = \{x : \{c, c'\}\}$. The typing derivation needs weakening the set types associated with the constants c and c'. For readability, we often omit labels, the environment Γ, and we only show the last productions added to effects.

$$\dfrac{\dfrac{\dfrac{4 \to \varepsilon \vdash x : \{c, c'\}}{3 \to \alpha c \mid \alpha c' \vdash \alpha(x) : \mathit{unit}}}{2 \to \varepsilon \vdash \lambda x.\, \alpha(x) : \{c, c'\} \xrightarrow{3} \mathit{unit}} \quad \dfrac{6 \to \varepsilon \vdash c : \{c\} \cup \{c'\} \quad 7 \to \varepsilon \vdash c' : \{c'\} \cup \{c\}}{5 \to 6 \mid 7 \vdash \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c' : \{c, c'\}}}{1 \to 2\,5\,3 \vdash e_1 : \mathit{unit}}$$

For e_2, let $\Gamma = \{x : \tau, y : \tau\}$. The typing derivation weakens the effects to unify the types in the rule for conditionals (note the introduction of the new label ℓ).

$$\dfrac{\dfrac{\dfrac{\dfrac{14 \to \varepsilon \vdash c : \{c\}}{13 \to \alpha c \vdash \alpha(c) : \mathit{unit}}}{\ell \to 13 \mid 16 \vdash \alpha(c) : \mathit{unit}}}{12 \to \varepsilon \vdash \lambda x.\, \alpha(c) : \tau \xrightarrow{\ell} \mathit{unit}} \quad \dfrac{\dfrac{\dfrac{17 \to \varepsilon \vdash c : \{c\}}{16 \to \alpha' c \vdash \alpha'(c) : \mathit{unit}}}{\ell \to 13 \mid 16 \vdash \alpha'(c) : \mathit{unit}}}{15 \to \varepsilon \vdash \lambda y.\, \alpha'(c) : \tau \xrightarrow{\ell} \mathit{unit}}}{11 \to 12 \mid 15 \vdash e_2 : \tau \xrightarrow{\ell} \mathit{unit}}$$

Example 4. Consider the browser and the applet defined in Example 1. After labelling all subexpressions, we obtain:

$$\begin{aligned} e^0 &= \varphi[(\alpha^2_{open}(c^3); (B^{10} A^{20})^4; \alpha^5_{close}(c^6))^1] \\ B^{10} &= \lambda_z x.\, \left(\varphi'[(x^{14} *^{15})^{13}]^{12};\ (z^{17}\, x^{18})^{16}\right)^{11} \\ A^{20} &= \lambda y.\, \left(\texttt{if}\, b\, \texttt{then}\, \alpha^{22}_{read}(c^{23})\, \texttt{else}\, \alpha^{24}_{connect}\right)^{21} \end{aligned}$$

Let $\Gamma = \{x : \mathit{unit} \xrightarrow{\ell,P} \mathit{unit}, y : \tau, z : \tau' \xrightarrow{\ell\ ,P} \mathit{unit}\}$, for some $\ell, \ell', P, P', \tau, \tau'$. For the expression A, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\underline{23} \to \varepsilon \vdash c : \{c\}}{\underline{22} \to \alpha_{read}\, c \vdash \alpha_{read}(c) : \mathit{unit}} \quad \underline{24} \to \alpha_{connect} \vdash \alpha_{connect} : \mathit{unit}}{\underline{21} \to 22 \mid 24 \vdash \texttt{if}\, b\, \texttt{then}\, \alpha_{read}(c)\, \texttt{else}\, \alpha_{connect} : \mathit{unit}}}{\underline{20} \to \varepsilon \vdash A : \tau \xrightarrow{21, P_A} \mathit{unit}}$$

where P_A collects all the productions generated in the previous deduction:

$$\begin{array}{ll} \underline{21} \to 22 \mid 24 & 23 \to \varepsilon \\ 22 \to \alpha_{read}\, c & 24 \to \alpha_{connect} \end{array}$$

The following typing derivation is possible for the body of B (we here shortcut typing, by assuming the abbreviation for ; introduced above).

$$\dfrac{\dfrac{\dfrac{\underline{14} \to \varepsilon \vdash x : \mathit{unit} \xrightarrow{\ell,P} \mathit{unit} \quad \underline{15} \to \varepsilon \vdash * : \mathit{unit}}{\underline{13} \to 14\,15\,\ell \vdash x * : \mathit{unit}}}{\underline{12} \to [_\varphi\ 13\,]_\varphi\ \vdash \varphi'[x *] : \mathit{unit}} \quad \dfrac{\vdots}{\underline{16} \to 17\,18\,\ell' \vdash z\, x : \mathit{unit}}}{\underline{11} \to 12\,16 \vdash \varphi'[x *]; z\, x : \mathit{unit}}$$

Let P_B be the following set of productions:

$$\begin{array}{ll} \underline{11} \to 12\;16 & 15 \to \varepsilon \\ 12 \to [_\varphi\, 13\,]_\varphi & 16 \to 17\;18\;\ell' \\ 13 \to 14\;15\;\ell & 17 \to \varepsilon \\ 14 \to \varepsilon & 18 \to \varepsilon \end{array}$$

To apply the rule for abstraction, we equate τ' to $\mathit{unit} \xrightarrow{\ell,P} \mathit{unit}$, and unify (ℓ', P') with $(11, P_B)$, obtaining $H_B = (11, P_B\{11/\ell'\})$. Then, we have:

$$\frac{\{x : \mathit{unit} \xrightarrow{\ell,P} \mathit{unit}, z : (\mathit{unit} \xrightarrow{\ell,P} \mathit{unit}) \xrightarrow{H_B} \mathit{unit}\}, H_B \vdash \varphi'[x\,*]; z\,x : \mathit{unit}}{\underline{10} \to \varepsilon \vdash B : (\mathit{unit} \xrightarrow{\ell,P} \mathit{unit}) \xrightarrow{H_B} \mathit{unit}}$$

To apply B to A, we solve the constraint $\mathit{unit} \xrightarrow{\ell,P} \mathit{unit} = \tau \xrightarrow{21,P_A} \mathit{unit}$. This yields $\tau = \mathit{unit}$, $\ell = 21$, and $P = P_A$, so enabling the following judgement:

$$\frac{B : (\mathit{unit} \xrightarrow{21,P_A} \mathit{unit}) \xrightarrow{H_B\{21/\ell\}} \mathit{unit} \quad A : \mathit{unit} \xrightarrow{21,P_A} \mathit{unit}}{\underline{4} \to 10\;20\;11 \vdash BA : \mathit{unit}}$$

We can eventually reconstruct the type and effect of e (notice that we cheat again and solve two sequential compositions in one step, here and in the labelling).

$$\frac{\dfrac{\dfrac{\underline{3} \to \varepsilon \vdash c : \{c\}}{\underline{2} \to \alpha_{open}\, c \vdash \alpha_{open}(c) : \mathit{unit}} \quad \dfrac{\vdots}{\underline{4} \to 10\;20\;11 \vdash BA : \mathit{unit}} \quad \cdots}{\underline{1} \to 2\;4\;5 \vdash \alpha_{open}(c); (BA); \alpha_{close}(c) : \mathit{unit}}}{\underline{0} \to [_\varphi\, 1\,]_\varphi \vdash e : \mathit{unit}}$$

Summing up, the history grammar of e is:

$$\begin{array}{lll} \underline{0} \to [_\varphi\, 1\,]_\varphi & 10 \to \varepsilon & 20 \to \varepsilon \\ 1 \to 2\;4\;5 & 11 \to 12\;16 & 21 \to 22 \mid 24 \\ 2 \to \alpha_{open}\; c & 12 \to [_\varphi\, 13\,]_\varphi & 22 \to \alpha_{read}\; c \\ 3 \to \varepsilon & 13 \to 14\;15\;21 & 23 \to \varepsilon \\ 4 \to 10\;20\;11 & 14 \to \varepsilon & 24 \to \alpha_{connect} \\ 5 \to \alpha_{close}\; c & 15 \to \varepsilon & \\ 6 \to \varepsilon & 16 \to 17\;18\;11 & \\ & 17 \to \varepsilon & \\ & 18 \to \varepsilon & \end{array}$$

Now it is convenient to introduce the following definition. Given $w \in T^*$, let $w^\flat$ be the string obtained from w by pruning all the framing events, and by replacing each substring αc with $\alpha(c)$. Back to Example 2, if $w = \alpha\, c\, [_\varphi\, \alpha'\, c'\,]_\varphi$ then $w^\flat = \alpha(c)\, \alpha'(c')$. The next theorem ensures that our type and effect system indeed approximates the actual run-time histories.

Theorem 1 (Correctness). *Let $\Gamma, H \vdash e : \tau$ and $\varepsilon, e \to^* \eta, e'$. Then, there exist $w \in T_H^*$ and $\gamma \in (T_H \cup N_H)^*$ such that $H \Rightarrow^* w\,\gamma$, and $w^\flat = \eta$.*

5 The Risky Events Analysis

In this section we exploit history grammars to determine the program points where security violations may occur. This paves us the way for gaining efficiency by discarding policy framings. Actually, we instrument programs with local checks, guarding risky events with those policies that may be violated. Formally, an event $\alpha^\ell(c)$ of an expression e is *risky* for the policy $\varphi(c')$ when there exists a computation $\varepsilon, e \to^* \eta, \mathit{fail}_{\ell,\varphi(c\ )}$. Below, we define a static analysis that extracts an over-approximations of the risky events of e from the history grammar H of e.

Our static analysis takes the form of a transition system, whose configurations are quadruples $\gamma, \sigma, \Phi, \zeta$. The component γ is a string derivable from H, σ is a mapping from policies to security automata states, Φ is a sequence of (universally quantified) policies, and ζ is a mapping from event labels to policies. The start configuration is $\ell, \sigma_0, \varepsilon, \zeta_0$, where ℓ is the start symbol of H, σ_0 maps, for all φ and c' occurring in e, the policy $\varphi(c')$ to the start state of $A_{\varphi(c\ )}$, and ζ_0 maps each event label to the empty set. Intuitively, σ mimics the evolution of the security automata, Φ records the active policies, and ζ accumulates the risky events, i.e. whenever $\alpha^\ell(c)$ is risky for $\varphi(c')$, then eventually $\varphi(c') \in \zeta(\ell)$.

The transition relation $\leadsto$ is the least one closed under the following rules.

Risky Events Analysis

$$[_\varphi\gamma, \sigma, \Phi, \zeta \leadsto_H \gamma, \sigma, \Phi\varphi, \zeta \qquad]_\varphi\gamma, \sigma, \Phi\varphi, \zeta \leadsto_H \gamma, \sigma, \Phi, \zeta$$

$$\frac{\ell \to \alpha\, c \in H \quad \Phi' = \{\, \varphi(c') \mid \varphi \in \Phi \wedge \delta(\sigma(\varphi(c')), \alpha(c)) = q_s \,\}}{\ell, \sigma, \Phi, \zeta \leadsto_H \varepsilon, \delta(\sigma, \alpha(c)), \Phi, \zeta\{\ell \mapsto \zeta(\ell) \cup \Phi'\}}$$

$$\frac{\ell \to \gamma \in H \quad \gamma \neq \alpha\, c}{\ell, \sigma, \Phi, \zeta \leadsto_H \gamma, \sigma, \Phi, \zeta} \qquad \frac{\ell, \sigma, \Phi, \zeta \leadsto_H \gamma', \sigma', \Phi', \zeta'}{\ell\gamma, \sigma, \Phi, \zeta \leadsto_H \gamma'\gamma, \sigma', \Phi', \zeta'}$$

The first axiom appends the policy φ in the sequence Φ upon a framing event $[_\varphi$; the second deals with the symmetric case $]_\varphi$. The central rule considers an event $\alpha^\ell(c)$, and checks whether an automaton $A_{\varphi(c\ )}$ enters the (non-accepting) sink state, for some active $\varphi \in \Phi$. In that case, the mapping ζ records the association of ℓ with $\varphi(c')$, i.e. that $\alpha^\ell(c)$ is possibly risky for $\varphi(c')$. The mapping σ is updated to reflect the change of state in the automata upon the event $\alpha(c)$: the next state of the (deterministic) automaton $A_{\varphi(c\ )}$ is the one reachable through $\alpha(c)$. For notational convenience, δ maps homomorphically on σ, i.e. on the states of the automata, that are all disjoint except for q_s. The last two rules simply carry over the above on strings γ.

Below we define the set of policies $act(e)$ active in an expression e. The intuition is that a policy φ is active if reducing the current redex involves checking φ. Notice that no policy is active for values, because they cannot be further reduced, and for conditionals, because evaluating a guard requires no check.

$$act(\alpha(r)) = \emptyset \qquad act(\varphi[e]) = \{\varphi\} \cup act(e) \qquad act(\texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e') = \emptyset$$
$$act(v\, e') = act(e') \qquad act(e\, e') = act(e)\ \ (e \neq v) \qquad act(\lambda x.\, e) = \emptyset$$

The following theorem establishes the soundness of the risky events analysis, by connecting each computation in $\rightarrow$ with one in $\rightsquigarrow$. In particular, when the first leads to a failing history η, at least one of the security automata used in the second reaches the sink state, and ζ associates the violated policy with the offending event. In any case, the active policies in the configurations of $\rightarrow$ are precisely recorded by the component Φ in the configurations of $\rightsquigarrow$.

Theorem 2. *Let $\Gamma, H \vdash e_0^\ell : \tau$, and $\varepsilon, e_0^\ell \rightarrow^n \eta_n, e_n$. Then, there exist $\gamma, \sigma, \Phi, \zeta$ such that $\ell, \sigma_0, \varepsilon, \zeta_0 \rightsquigarrow_H^* \gamma, \sigma, \Phi, \zeta$, and:*

(2a) if $e_n = fail_{\ell', \varphi(c)}$ then $\varphi(c) \in \zeta(\ell')$ and $\delta(\sigma(\varphi(c)), \eta) = q_s$
(2b) otherwise, $\sigma = \delta(\sigma_0, \eta)$, and $\{\, \varphi \mid \varphi$ occurs in $\Phi \,\} = act(e_{n-1})$.

We now show that the an over-approximation to the risky events of a program can be computed in a finite amount of time. More precisely, there exists a bound on the length of the computations needed to stabilise ζ.

Theorem 3. *Let $\Gamma, H \vdash e : \tau$, for e closed. For each $n \geq 0$, define:*

$$Z_n \;=\; \{\, \zeta \mid \ell, \sigma_0, \varepsilon, \zeta_0 \rightsquigarrow_H^n \gamma, \sigma, \Phi, \zeta \,\}$$

Then, there exists k such that, for all $i \geq 0$, $Z_{k+i} = Z_k$.

Given the set Z_k of the above theorem, we hereafter denote with RE the mapping such that $\ell \mapsto \{\varphi(c) \in \zeta(\ell) \in Z_k\}$. Call ℓ H-risky for $\varphi(c)$ whenever $\varphi(c) \in \mathsf{RE}(\ell)$. Theorems 1, 2 and 3 allow us to establish the correctness of the risky events analysis: if $\alpha^\ell(c')$ is risky for $\varphi(c)$, then ℓ is H-risky for $\varphi(c)$.

Example 5. Consider the browser/applet system analysed in Example 4. The following computation discovers that 24 is a risky event for $\varphi'(c)$.

$$\begin{aligned}0, \sigma_0, \varepsilon, \zeta_0 &\rightsquigarrow^* 22\ ]_\varphi\ \ 16\ 5\ ]_\varphi, \sigma_1 = \delta(\sigma_0, \alpha_{open}(c)), \varphi\varphi', \zeta_0 \\ &\rightsquigarrow^* 24\ ]_\varphi\ \ 16\ 5\ ]_\varphi, \sigma_2 = \delta(\sigma_1, \alpha_{read}(c)), \varphi\varphi', \zeta_0 \\ &\rightsquigarrow\ ]_\varphi\ \ 16\ 5\ ]_\varphi, \delta(\sigma_2, \alpha_{connect}), \varphi\varphi', \zeta_0\{24 \mapsto \varphi'(c)\}\end{aligned}$$

6 Instrumentation with Local Checks

We eventually exploit the risky events analysis to obtain an expression with local checks only, equivalent to a given expression with local policies. The syntax and operational semantics of the target language λ^{check} follow.

The target language λ^{check}

$e, e' ::= x$	variable
c	constant
$\texttt{check}\ \Phi\ \texttt{in}\ \alpha(r)$	guarded event
$\texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e'$	conditional
$\lambda_z x.\, e$	abstraction
$e\, e'$	application
$\texttt{enter}\ \varphi\ \texttt{in}\ e$	activate policy

The configurations of the operational semantics are pairs σ, e. A transition $\Phi \vdash \sigma, e \twoheadrightarrow \sigma', e'$ means that the expression e reduces to e' and the state σ of the security automata evolves to σ', provided that Φ is the set of active policies. Note that it suffices to record in σ the states of those automata $A_{\varphi(c)}$ such that $\varphi(c) \in \mathsf{RE}(\ell)$, for some ℓ. Remarkably, the size of the configurations is bounded, unlike those of $\lambda^{[\,]}$, where histories η could grow unbound.

Operational semantics of λ^{check}

$$\frac{\Phi \vdash \sigma, e_1 \twoheadrightarrow \sigma', e_1'}{\Phi \vdash \sigma, e_1 e_2 \twoheadrightarrow \sigma', e_1' e_2} \qquad \frac{\Phi \vdash \sigma, e_2 \twoheadrightarrow \sigma', e_2'}{\Phi \vdash \sigma, v e_2 \twoheadrightarrow \sigma', v e_2'}$$

$$\Phi \vdash \sigma, (\lambda_z x.e)v \twoheadrightarrow \sigma, e\{v/x, \lambda_z x.e/z\} \qquad \Phi \vdash \sigma, \texttt{if}\ b\ \texttt{then}\ e_0\ \texttt{else}\ e_1 \twoheadrightarrow \sigma, e_{\mathcal{B}(b)}$$

$$\frac{\{\, \varphi(c') \in \Phi' \mid \varphi \in \Phi \ \wedge\ \delta(\sigma(\varphi(c')), \alpha(c)) = q_s \,\} = \emptyset}{\Phi \vdash \sigma, \texttt{check}\ \Phi'\ \texttt{in}\ \alpha(c) \twoheadrightarrow \delta(\sigma, \alpha(c)), *}$$

$$\frac{\Phi \cup \{\varphi\} \vdash \sigma, e \twoheadrightarrow \sigma', e'}{\Phi \vdash \sigma, \texttt{enter}\ \varphi\ \texttt{in}\ e \twoheadrightarrow \sigma', \texttt{enter}\ \varphi\ \texttt{in}\ e'} \qquad \Phi \vdash \sigma, \texttt{enter}\ \varphi\ \texttt{in}\ v \twoheadrightarrow \sigma, v$$

The first four rules are straightforward. The rule for an event $\alpha(c)$ guarded by Φ' requires that any policy $\varphi(c') \in \Phi'$ such that φ is active, does not lead to the sink state upon $\alpha(c)$. The statement $\texttt{enter}\ \varphi\ \texttt{in}\ e$ is similar to a block in programming languages: the policy φ is active while reducing e, and its scope is left when e eventually becomes a value.

We are now ready to instrument an expression e in $\lambda^{[\,]}$ and obtain an equivalent expression $\mathit{instr}_{\mathsf{RE}}(e)$ in λ^{check}. In the definition below, we use the risky events collected in RE by the analysis on history grammars. The first rule discards all the policy framings, but records entering the scope of a policy. The other interesting rule is that for instrumenting events: an event $\alpha^{\ell}(c)$, H-risky for Φ, becomes guarded by a check on all the policies in Φ.

Instrumentation of $\lambda^{[]}$ expressions

$$instr_{\mathsf{RE}}(\varphi[e]) = \texttt{enter}\ \varphi\ \texttt{in}\ instr_{\mathsf{RE}}(e) \qquad instr_{\mathsf{RE}}(*) = * \qquad instr_{\mathsf{RE}}(x) = x$$

$$instr_{\mathsf{RE}}(c) = c \qquad instr_{\mathsf{RE}}(\alpha^{\ell}(r)) = \texttt{check}\ \mathsf{RE}(\ell)\ \texttt{in}\ \alpha(r)$$

$$instr_{\mathsf{RE}}(\lambda_z x.\, e) = \lambda_z x.\, instr_{\mathsf{RE}}(e) \quad instr_{\mathsf{RE}}(e_0\, e_1) = instr_{\mathsf{RE}}(e_0)\, instr_{\mathsf{RE}}(e_1)$$

$$instr_{\mathsf{RE}}(\texttt{if}\ b\ \texttt{then}\ e_0\ \texttt{else}\ e_1) = \texttt{if}\ b\ \texttt{then}\ instr_{\mathsf{RE}}(e_0)\ \texttt{else}\ instr_{\mathsf{RE}}(e_1)$$

The correctness of instrumentation is stated by the following theorem: each execution step of e in $\lambda^{[]}$ corresponds exactly to one step of $instr_{\mathsf{RE}}(e)$ in λ^{check}.

Theorem 4. *For all histories η, non-failure expressions e, e', and integers n:*

$$\varepsilon, e \rightarrow^{n} \eta, e' \iff \emptyset \vdash \sigma_0, instr_{\mathsf{RE}}(e) \twoheadrightarrow^{n} \delta(\sigma_0, \eta), instr_{\mathsf{RE}}(e')$$

7 Conclusions

We have proposed a mixed approach to history-based access control. To this aim, we have exploited $\lambda^{[]}$, an extension of the λ-calculus that allows for security policies with possibly nested, local scopes [4].

We have defined a type and effect system to extract from a given program a history grammar that approximates its run-time behaviour, as far as security is concerned. A history grammar is a context-free grammar that generates execution histories while recording the scope of local policies.

Dynamic checking history-based policies is in general unfeasible, because histories may grow unbound. We have been able to transform programs to record just the abstraction of the history needed to guarantee security, i.e. where and which policies have to be checked. This relies on a (polynomial-time) static analysis on history grammars, which detects the events that are risky, and the policies they may violate. Local checks of risky events can then replace local policies, thus making the dynamic control of accesses feasible.

Colcombet and Fradet [8] and Marriot, Stuckey and Sulzmann [15] mixed static and dynamic techniques to transform programs and make them obey a given global policy. The approach of [8] abstracts a program into a control flow graph, which is then instrumented with some annotations, to track the state of the finite-state automaton that enforces the global property. A minimization phase follows, to remove the unnecessary tracking. Finally, the optimized control flow graph is converted back to a program, that is guaranteed to abort just before violating the property. The approach of [15] is based on over-approximating the run-time behaviour of a program through a context-free grammar. A finite-state automaton models the permitted resource usages. If the language generated by the grammar is not included in the language accepted by the automaton, the program is instrumented with the local checks and tracking operations needed to make it obey the policy. Compared to [8,15], our programming model allows for local policies and access events parametrized over resources, while the others only consider global policies and no parametrized events.

Igarashi and Kobayashi proposed in [13] a unified framework for analysing the usage of resources. This is based on an extension of the λ-calculus that features primitives for creating and accessing resources, and for defining their permitted usage patterns. The resource usage problem requires to approximate the *use function* that maps expressions to the sequences of possible usage patterns. An execution is resource-safe when the possible patterns are contained in the permitted ones. A type system guarantees that well-typed expressions are resource-safe. Compared to the calculus of [13], $\lambda^{[]}$ has no primitive for resource creation, but we plan to introduce it in future work. Indeed, the programming model of [13] is even too powerful: as a result, no complete algorithm exists to verify that inferred usages conform to the permitted ones. Instead, in [4] we provided $\lambda^{[]}$ with a static technique to verify when a program is secure.

Skalka and Smith proposed λ_{hist} [19], a λ-calculus with local security checks that enforce linear μ-calculus properties [7,14] on the past history. A type and effect system approximates the possible run-time histories. Type safety ensures that a typable expression will not go wrong if its effect is *valid*, i.e. all the histories it represents always pass the local security checks. The validity of effects can be statically verified by model checking μ-calculus formulae over Basic Process Algebras [6,10]. Compared to Skalka and Smith's λ_{hist}, our $\lambda^{[]}$ features a different programming construct for access control: while in λ_{hist} the access control tests are dictated by local checks inserted into programs, we have policy framings, that localize the time intervals where safety policies must be enforced.

Walker [21] explored an alternative approach to access control, that mixes static and dynamic techniques with proof-carrying code [16]. Security properties are specified by *security automata* [5,17]. When a security-unaware program is compiled, a centralized security policy tells where to insert local checks, in order to obtain provably-secure compiled code. An optimization phase follows: whenever a security check is removed, it is replaced by a proof that the optimized code is still safe. This is done through typed compilation schemata: types encode assertions about program security, ensuring that no run-time violation of the security properties will occur. Before executing a piece of code, a certified verification software ensures that it respects the centralized security policy. Thus, compilers are no longer required to belong to the trusted computing base.

Our previous work [4] has a type and effect system, similar to the present one, whose effects are instead *history expressions*, equivalent to Basic Process Algebras. These effects are model checked with specially-tailored Büchi automata, to detect whether the program under analysis never goes wrong. In [3] we further refined this model to include liveness properties and call-by-contract service invocation, thus providing a framework for secure service composition. Compared with [4,3], the present paper never rejects programs that possibly go wrong, by mechanically adding the necessary run-time checks only.

Acknowledgments. We wish to thank the anonymous referees for their insightful comments. Research partially supported by the Project FET-GC II SENSORIA.

References

1. M. Abadi and C. Fournet. Access control based on execution history. In *Proc. 10th Annual Network and Distributed System Security Symposium*, 2003.
2. A. Banerjee and D. A. Naumann. History-based access control and secure information flow. In *Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart Cards (CASSIS)*, 2004.
3. M. Bartoletti, P. Degano, and G. L. Ferrari. Enforcing secure service composition. In *Proc. 18th Computer Security Foundations Workshop (CSFW)*, 2005.
4. M. Bartoletti, P. Degano, and G. L. Ferrari. History based access control with local policies. In *Proc. Fossacs*, 2005.
5. L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Foundations of Computer Security (FCS '02)*, 2002.
6. J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.
7. J. C. Bradfield. On the expressivity of the modal mu-calculus. In *Proc. International Symposium on Theoretical Aspects of Computer Science*, 1996.
8. T. Colcombet and P. Fradet. Enforcing trace properties by program transformation. In *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2000.
9. G. Edjlali, A. Acharya, and V. Chaudhary. History-based access control for mobile code. In *Secure Internet Programming*, 1999.
10. J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. 19th Int. Colloquium on Trees in Algebra and Programming*, 1994.
11. P. W. Fong. Access control by tracking shallow execution history. In *IEEE Symposium on Security and Privacy*, 2004.
12. C. Fournet and A. D. Gordon. Stack inspection: theory and variants. *ACM Transactions on Programming Languages and Systems*, 25(3):360–399, 2003.
13. A. Igarashi and N. Kobayashi. Resource usage analysis. In *Proc. 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2002.
14. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. K. Marriott, P. J. Stuckey, and M. Sulzmann. Resource usage verification. In *Proc. First Asian Programming Languages Symposium*, 2003.
16. G. C. Necula. Proof-carrying code. In *Proc. 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1997.
17. F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.
18. P. Sewell and J. Vitek. Secure composition of untrusted code: box-π, wrappers and causality types. *Journal of Computer Security*, 11(2), 2003.
19. C. Skalka and S. Smith. History effects and verification. In *Asian Programming Languages Symposium*, 2004.
20. J.-P. Talpin and P. Jouvelot. The type and effect discipline. In *Proc. 7th IEEE Symposium on Logic in Computer Science*, 1992.
21. D. Walker. A type system for expressive security policies. In *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2000.

The Graph Rewriting Calculus: Confluence and Expressiveness

Clara Bertolissi

LORIA & UHP, BP 239 54506, Vandoeuvre-lès-Nancy, Cedex, France
clara.bertolissi@loria.fr

Abstract. Introduced at the end of the nineties, the Rewriting Calculus (ρ-calculus, for short) is a simple calculus that uniformly integrates term-rewriting and λ-calculus. The ρ_g-Calculus has been recently introduced as an extension of the ρ-calculus, handling structures with cycles and sharing. The calculus over terms is naturally generalized by using unification constraints in addition to the standard ρ-calculus matching constraints. This leads to a term-graph representation in an equational style where terms consist of unordered lists of equations. In this paper we show that the (linear) ρ_g-Calculus is confluent. The proof of this result is quite elaborated, due to the non-termination of the system and to the fact that we work on equivalence classes of terms. We also show that the ρ_g-Calculus can be seen as a generalization of first-order term-graph rewriting, in the sense that for any term-graph rewrite step a corresponding sequence of rewritings can be found in the ρ_g-Calculus.

1 Introduction

Term rewriting is a general framework for specifying and reasoning about computations that combines elements of automated theorem proving, universal algebra and functional programming. It provides very efficient methods for reasoning with equations and it can be regarded as a powerful abstract computational model. In particular, term rewriting systems can be used for software verification: the behavior of a functional or rewrite-based program can be described by analyzing some properties of the associated term rewriting system. For example, the confluence property ensures that the output of the program, if it exist, is unique for any given input data.

In this framework, the rewriting calculus (ρ-calculus, for short) has been introduced in the late nineties as a natural generalization of term rewriting and of the λ-calculus [9]. The rewrite rules, acting as elaborated abstractions, their application and the obtained structured results are first-class objects of the calculus. The evaluation mechanism, generalizing beta-reduction, strongly relies on term matching in various theories. Several variants of the calculus have been already studied, such as typed versions [5], extensions with explicit substitutions [8] or with imperative features [15].

In the term rewriting setting, terms are often seen as trees but in order to improve the efficiency of the implementation of functional languages, it is of fundamental interest to think and implement terms as graphs [4]. In this case, the possibility of sharing subterms allows one to save space and time during the computation. Moreover, the possibility to define cycles leads to an increased expressive power that allows one to represent naturally regular infinite data structures. Cyclic term-graph rewriting has been widely studied from different points of view: operational [4], categorical [11] or equational [1] (see [18] for a survey on term-graph rewriting).

Following the last approach, we proposed in [7] a system, called ρ_{g}-Calculus, that generalizes the standard ρ-calculus in order to deal with higher-order cyclic terms. The first contribution of this paper shows that the ρ_{g}-Calculus, under some linearity assumptions, is confluent. In the ρ_{g}-Calculus terms can have an associated list of constraints which is composed of recursion equations, used to express sharing and cycles, and matching constraints, arising from the fact that computations related to the matching are made explicit and performed at the object-level. The order of constraints is irrelevant and therefore the conjunction operator is considered to be commutative and associative, with the the empty constraint as neutral element. Moreover, the idempotence axiom is used to avoid the duplication of constraints.

The fact that reductions take place over equivalence classes of terms rather than over single terms must be considered when proving the confluence of the calculus, and this makes the result more difficult to achieve. The proof method generalizes the proof of confluence of the cyclic λ-calculus [2] to the setting of rewriting modulo an equational theory [16] and moreover it adapts the proof to deal with terms containing patterns and matching equations. More precisely, the proof, which is only sketched in the paper, uses the concept of "developments" and the property of "finiteness of developments" as defined in the theory of classical λ-calculus [3]. The interested reader can find the complete proof in [6].

The ρ_{g}-Calculus is an expressive formalism that has already been shown to be a generalization of both the plain ρ-calculus and the λ-calculus extended with explicit recursion, providing an homogeneous framework for pattern matching and higher-order graphical structures. In this paper we show how the ρ_{g}-Calculus can be naturally seen also as an extension of term-graph rewriting. More specifically, we prove that matching in the ρ_{g}-Calculus is well-behaved *w.r.t.* the notion of homomorphism on term-graphs and that any reduction step in a term-graph rewrite system can be simulated in the ρ_{g}-Calculus.

The paper is organized as follows. In Section 2 we describe the syntax and the small-step semantics of the ρ_{g}-Calculus. In Section 3 we outline the proof of confluence for the ρ_{g}-Calculus. In Section 4 we review first-order term-graph rewriting in the equational approach, showing that term-graph reductions can be simulated in the ρ_{g}-Calculus. We conclude in Section 5 by presenting some perspectives of this work.

2 The ρ_{g}-Calculus: Syntax and Semantics

The syntax of the ρ_{g}-Calculus presented in Fig. 1 extends the syntax of the standard ρ-calculus and of the ρ_{x}-calculus [8], *i.e.* the ρ-calculus with explicit matching and substitution application. As in the plain ρ-calculus, λ-abstraction is generalized by a rule abstraction $P \twoheadrightarrow G$, where P is in general an arbitrary term. There are two different application operators: the functional application operator, denoted simply by concatenation, and the constraint application operator, denoted by as “$\_\ [\_]$”. Terms can be grouped together into *structures* built using the operator “$\_;\_$”. Depending on the theory behind this operator a structure can represent, e.g., a multi-set (when “;” is associative and commutative) or a set (when “;” is associative, commutative and idempotent) of terms.

In the ρ_{g}-Calculus constraints are conjunctions (built using the operator “$\_,\_$”) of match equations of the form $\mathcal{P} \ll \mathcal{G}$ and recursion equations of the form $\mathcal{X} = \mathcal{G}$. The empty constraint is denoted by ϵ. The operator “$\_,\_$” is supposed to be associative, commutative and idempotent, with ϵ as neutral element.

We assume that the application operator associates to the left, while the other operators associate to the right. To simplify the syntax, operators have different priorities. Here are the operators ordered from higher to lower priority: “$\_\ \_$”, “$\_ \twoheadrightarrow \_$”, “$\_;\_$”, “$\_\ [\_]$” , “$\_ \ll \_$”, “$\_ = \_$” and “$\_,\_$”.

The symbols $G, H, P \ldots$ range over the set $\mathcal{G}$ of terms, $x, y, \ldots$ range over the set $\mathcal{X}$ of variables, $a, b, \ldots$ range over a set $\mathcal{K}$ of constants. The symbols $E, F, \ldots$ range over the set $\mathcal{C}$ of constraints. We call *algebraic* the terms of the form $(((f\ G_1)\ G_2) \ldots)\ G_n$, with $f \in \mathcal{K}$, $G_i \in \mathcal{X} \cup \mathcal{K}$ or G_i algebraic for $i = 1 \ldots n$, and we usually denote them by $f(G_1, G_2, \ldots, G_n)$.

We denote by $\bullet$ (black hole) a constant, already introduced in [1] using the equational approach and also in [11] using the categorical approach, to give a name to “undefined” terms that correspond to the expression $x\ [x = x]$ (self-loop). The notation $x =_{\circ} x$ is an abbreviation for the sequence $x = x_1, \ldots, x_n = x$. We use the symbol $\mathsf{Ctx}\{\square\}$ for a context with exactly one hole $\square$. We say that a ρ_{g}-term is *acyclic* if it contains no sequence of constraints of the form $\mathsf{Ctx}_0\{x_0\} \lll \mathsf{Ctx}_1\{x_1\}, \mathsf{Ctx}_2\{x_1\} \lll \mathsf{Ctx}_3\{x_2\}, \ldots, \mathsf{Ctx}_m\{x_n\} \lll \mathsf{Ctx}_{m+1}\{x_0\}$, with $n, m \in \mathbb{N}$ and $\lll \in \{=, \ll\}$. A sequence of this kind is called a *cycle.*

For the purpose of this paper we restrict to left-hand sides of abstractions and match equations that are acyclic, algebraic terms, with all their subterms algebraic and not containing constraints. The set of all these terms, called *patterns*, is denoted by $\mathcal{P}$. For instance, the ρ_{g}-term $(f(y)\ [y = g(y)] \twoheadrightarrow a)$ is not allowed since the abstraction has a cyclic left-hand side. We call a ρ_{g}-term *well-formed* if each variable occurs at most once as left-hand side of a recursion equation. All the ρ_{g}-terms considered in the sequel will be implicitly well-formed.

Terms

$\mathcal{G}, \mathcal{P} ::=$	$\mathcal{X}$	(Variables)
	$\mid \mathcal{K}$	(Constants)
	$\mid \mathcal{P} \twoheadrightarrow \mathcal{G}$	(Abstraction)
	$\mid \mathcal{G}\ \mathcal{G}$	(Functional application)
	$\mid \mathcal{G}; \mathcal{G}$	(Structure)
	$\mid \mathcal{G}\ [\mathcal{C}]$	(Constraint application)

Constraints

$\mathcal{C} ::=$	ϵ	(Empty constraint)
	$\mid \mathcal{X} = \mathcal{G}$	(Recursion equation)
	$\mid \mathcal{P} \ll \mathcal{G}$	(Match equation)
	$\mid \mathcal{C}, \mathcal{C}$	(Conjunction)

Fig. 1. Syntax of the ρ_{g}-Calculus

The notions of free and bound variables of ρ_{g}-terms take into account the three binders of the calculus: abstraction, recursion and match. Intuitively, variables on the left hand-side of any of these operators are bound by the operator. The set of free variables of a ρ_{g}-term G is denoted by $\mathcal{FV}(G)$. Moreover, given a constraint $\mathcal{C}$ we will refer to the set $\mathcal{DV}(\mathcal{C})$, of variables "defined" in $\mathcal{C}$. This set includes, for any recursion equation $x = G$ in $\mathcal{C}$, the variable x and for any match $P \ll G$ in $\mathcal{C}$, the set of free variables of P. For a formal definition, see [7].

We work modulo *α-conversion* and we use Barendregt's *"hygiene-convention"*, *i.e.* free and bound variables have different names [3]. Note that the scope of a recursion variable is limited to the ρ_{g}-terms appearing in the list of constraints where such variable is defined and the ρ_{g}-term to which this list is applied. For example, in $f(x, y)\ [x = g(y)\ [y = a]]$ the variable y defined in the recursion equation binds its occurrence in $g(y)$ but not in $f(x, y)$. In fact, the term does not satisfy the naming conditions since y occurs both free and bound. This naming convention allows us to apply replacements (like for the evaluation rules in Fig. 2) quite straightforwardly, since no variable capture is possible.

We define next an order over variables bound by a match or an equation. This order will be later used in the definition of the substitution rule of the calculus, which will allow one only upward substitutions. As we will see later, this is essential for obtaining the confluence of the calculus. We denote by $\leq$ the least pre-order on recursion variables such that $x \geq y$ if $x = \mathsf{Ctx}\{y\}$, for some context $\mathsf{Ctx}\{\square\}$. The equivalence induced by the pre-order is denoted $\equiv$ and we say that x and y are cyclically equivalent ($x \equiv y$) if $x \geq y \geq x$ (they lie on a common cycle). We write $x > y$ if $x \geq y$ and $x \not\equiv y$.

Example 1 (Some ρ_{g}-terms).

1. In the rule $(2 * f(x)) \rightarrow ((y + y)\ [y = f(x)])$ the sharing in the right-hand side avoids the copying of the object instantiating $f(x)$, when the rule is applied to a ρ_{g}-term.
2. The ρ_{g}-term $x\ [x = \mathit{cons}(0, x)]$ represents an infinite list of zeros.
3. The ρ_{g}-term $f(x, y)\ [x = g(y), y = g(x)]$ is an example of twisted sharing that can be expressed using mutually recursive constraints (to be read as a `letrec` construct). We have that $x \geq y$ and $y \geq x$, hence $x \equiv y$.

The complete set of evaluation rules of the ρ_{g}-Calculus is presented in Fig. 2. As in the plain ρ-calculus, in the ρ_{g}-Calculus the application of a rewrite rule to a term is represented as the application of an abstraction. A redex can be activated using the ρ rule in the Basic rules, which creates the corresponding matching constraint. The computation of the substitution which solves the matching is then performed explicitly by the Matching rules and, if the computation is successful, the result is a recursion equation added to the list of constraints of the term. This means that the substitution is not applied immediately to the term but it is kept in the environment for a delayed application or for deletion if useless, as expressed by the Graph rules.

More precisely, the first two rules ρ and δ come from the ρ-calculus. The rule δ distributes the application over the the structures built with the ";" operator.

Basic rules:

$$\begin{array}{lll}
(\rho) & (P \twoheadrightarrow G_2)\ G_3 & \to_\rho G_2\ [P \ll G_3] \\
 & (P \twoheadrightarrow G_2)\ [E]\ G_3 & \to_\rho G_2\ [P \ll G_3, E] \\
(\delta) & (G_1; G_2)\ G_3 & \to_\delta G_1\ G_3; G_2\ G_3 \\
 & (G_1; G_2)\ [E]\ G_3 & \to_\delta (G_1\ G_3; G_2\ G_3)\ [E]
\end{array}$$

Matching rules:

$$\begin{array}{lll}
(\mathit{propagate}) & P \ll (G\ [E]) & \to_p\ P \ll G, E \quad \mathit{if}\ P \neq x \\
(\mathit{decompose}) & K(G_1, \ldots, G_n) \ll K(G'_1, \ldots, G'_n) & \to_{dk}\ G_1 \ll G'_1, \ldots, G_n \ll G'_n \\
 & & \quad \mathit{with}\ n \geq 0 \\
(\mathit{solved}) & x \ll G, E & \to_s\ x = G, E \quad \mathit{if}\ x \notin \mathcal{DV}(E)
\end{array}$$

Graph rules:

$$\begin{array}{lll}
(\mathit{external\ sub}) & \mathsf{Ctx}\{y\}\ [y = G, E] & \to_{es}\ \mathsf{Ctx}\{G\}\ [y = G, E] \\
(\mathit{acyclic\ sub}) & G\ [P \lll \mathsf{Ctx}\{y\}, y = G_1, E] & \to_{ac}\ G\ [P \lll \mathsf{Ctx}\{G_1\}, y = G_1, E] \\
 & & \quad \mathit{if}\ x > y,\ \forall x \in \mathcal{FV}(P) \\
 & & \quad \mathit{where} \lll \in \{=, \ll\} \\
(\mathit{garbage}) & G\ [E, x = G'] & \to_{gc}\ G\ [E] \\
 & & \quad \mathit{if}\ x \notin \mathcal{FV}(E) \cup \mathcal{FV}(G) \\
 & G\ [\epsilon] & \to_{gc}\ G \\
(\mathit{black\ hole}) & \mathsf{Ctx}\{x\}\ [x =_\circ x, E] & \to_{bh}\ \mathsf{Ctx}\{\bullet\}\ [x =_\circ x, E] \\
 & G\ [P \lll \mathsf{Ctx}\{y\}, y =_\circ y, E] & \to_{bh}\ G\ [P \lll \mathsf{Ctx}\{\bullet\}, y =_\circ y, E] \\
 & & \quad \mathit{if}\ \ x > y,\ \forall x \in \mathcal{FV}(P)
\end{array}$$

Fig. 2. Small-step semantics of the ρ_{g}-Calculus

The rule ρ triggers the application of a rewrite rule to a ρ_{g}-term by applying the appropriate constraint to the right-hand side of the rule. For each of these rules, an additional rule dealing with the presence of constraints is considered.

The Matching rules and in particular the rule *decompose* are strongly related to the theory modulo which we want to compute the solutions of the matching. In this paper we consider the syntactical matching, which is known to be decidable, but extensions to more elaborated theories are possible. Due to the assumptions on the left-hand sides of rewrite rules and of constraints, we only need to decompose algebraic terms. The goal of this set of rules is to produce a constraint of the form $x_1 = G_1, \ldots, x_n = G_n$ starting from a matching equation. Some replacements might be needed (as defined by the Graph rules) as soon as the terms contain some sharing. The *propagate* rule performs a flattening of a list of constraints which are propagated to the top level. The rule *solved* transforms a matching constraint $x \ll G$ into a recursion equation $x = G$. The proviso asking that x is not defined elsewhere in the constraint is necessary in the case of matching problems involving non-linear constraints. For example, the constraint $x \ll a, x \ll b$ should not be reduced showing that the original (non-linear) matching problem has no solution.

The Graph rules are inherited from the cyclic λ-calculus [2]. The first two rules make a copy of a ρ_{g}-term associated to a recursion variable into a term that is inside the scope of the corresponding constraint. This is important to make a redex explicit (*e.g.* in $x\ a\ [x = a \twoheadrightarrow b]$) or or to solve a match equation (*e.g.* in $a\ [a \ll x, x = a]$). As already mentioned, the substitution rule allows one to make the copies only upwards *w.r.t.* the order defined on the variables of

ρ_g-terms. In the cyclic λ-calculus this is needed for the confluence of the system (see [2] for a counterexample) and it will be one of the key ingredients also for the confluence of the ρ_g-Calculus. The *garbage* rules get rid of recursion equations that represent non-connected parts of the term. Matching constraints are not eliminated, keeping thus the trace of matching failures during an unsuccessful reduction. The *black hole* rules replace the undefined ρ_g-terms, intuitively corresponding to self-loop graphs, with the constant $\bullet$.

Note that all the evaluation steps are performed modulo the underlying theory associated to the "_, _" operator, as we will detail in the next section.

Example 2. [A simple reduction]

$$\begin{array}{l}
\quad (f(a,a) \twoheadrightarrow a)\ (f(y,y)\ [y=a]) \\
\mapsto_{\rho}\ a\ [f(a,a) \ll f(y,y)\ [y=a]] \ \mapsto_{p}\ a\ [f(a,a) \ll f(y,y), y=a] \\
\mapsto_{dk}\ a\ [a \ll y, a \ll y, y=a] \ = \ a\ [a \ll y, y=a] \quad \textit{(by idempotency)} \\
\mapsto_{ac}\ a\ [a \ll a, y=a] \ \mapsto_{dk}\ a\ [y=a] \ \mapsto_{gc}\ a
\end{array}$$

3 Confluence of the ρ_g-Calculus

The confluence for higher-order systems dealing with non-linear matching is difficult to get since we usually obtain non-joinable critical pairs as shown in [14] in the setting of the λ-calculus. Klop's counterexample can be encoded in the ρ-calculus [19] to show that ρ-calculus is not confluent if no evaluation strategy is used. The counterexample is still valid when generalizing the ρ-calculus to the ρ_g-Calculus. For this reason, we restrict in the following to a linear ρ_g-Calculus.

Definition 1 (Linear ρ_g-Calculus). *A pattern is called* linear *if it does not contain two occurrences of the same variable. We say that a constraint* $[P_1 \lll G_1, \ldots, P_n \lll G_n]$ *is* linear *if all patterns are linear and if* $\bigcap_{i=1}^{n} \mathcal{FV}(P_i) = \emptyset$.

The linear ρ_g-Calculus *is the ρ_g-Calculus where all the patterns in the left-hand side of abstractions and all constraints are linear.*

As mentioned before, the "_, _" operator is supposed to be associative, commutative and idempotent, with ϵ as a neutral element. However, in the linear ρ_g-Calculus, idempotency is not needed since constraints of the form $x \ll G, x \ll G$ are not allowed (and cannot arise from reductions). Therefore, in the ρ_g-Calculus, rewriting must be thought of as acting over equivalence classes of ρ_g-terms with respect to $\sim_{AC_\epsilon}$, the congruence relation generated by the associativity and commutativity axioms for the "_, _" operator, and the neutrality axiom for ϵ. If ρ_g denotes the rewrite system in Fig. 2, then the relation induced over AC_ϵ-equivalence classes is denoted $\mapsto_{\rho_g/AC_\epsilon}$ and formally defined by $T_1 \mapsto_{\rho_g/AC_\epsilon} T_2$ if $T_1 \sim_{AC_\epsilon} \mathsf{Ctx}\{\sigma(L)\}$ and $T_2 \sim_{AC_\epsilon} \mathsf{Ctx}\{\sigma(R)\}$ with $L \to R$ any rule in ρ_g and σ a substitution.

Concretely, in most of the proofs we will use the notion of rewriting modulo AC_ϵ à la Peterson and Stickel [17], denoted $\mapsto_{\rho_g,AC_\epsilon}$. In this case rewrite rules act on terms instead than on equivalence classes of terms, and matching modulo AC_ϵ is performed at each step of the reduction. Formally $T_1 \mapsto_{\rho_g,AC_\epsilon} T_2$ if

$$\begin{tikzcd} \cdot \arrow[rr, mapsto, "R"] \arrow[d, mapsto, "R"'] & & \cdot \arrow[d, dotted, mapsto, "R"] \\ \cdot \arrow[r, dotted, mapsto, "R"'] & \cdot \arrow[r, phantom, "\sim_{AC_\epsilon}"] & \cdot \end{tikzcd}$$

Diamond

$$\begin{tikzcd} \cdot \arrow[rr, mapsto, twoheadrightarrow, "R"] \arrow[d, mapsto, twoheadrightarrow, "R"'] & & \cdot \arrow[d, dotted, mapsto, twoheadrightarrow, "R"] \\ \cdot \arrow[r, dotted, mapsto, twoheadrightarrow, "R"'] & \cdot \arrow[r, phantom, "\sim_{AC_\epsilon}"] & \cdot \end{tikzcd}$$

Confluence

$$\begin{tikzcd} \cdot \arrow[rr, mapsto, twoheadrightarrow, "R_2"] \arrow[d, mapsto, twoheadrightarrow, "R_1"'] & & \cdot \arrow[d, dotted, mapsto, twoheadrightarrow, "R_2"] \\ \cdot \arrow[r, dotted, mapsto, twoheadrightarrow, "R_1"'] & \cdot \arrow[r, phantom, "\sim_{AC_\epsilon}"] & \cdot \end{tikzcd}$$

Commutation

$$\begin{tikzcd} \cdot \arrow[r, phantom, "\sim_{AC_\epsilon}"] \arrow[d, mapsto, "R"'] & \cdot \arrow[d, dotted, mapsto, "R"] \\ \cdot \arrow[r, phantom, "\sim_{AC_\epsilon}"] & \cdot \end{tikzcd}$$

Compatibility

Fig. 3. Properties of rewriting modulo $\sim_E$

$T_1 = \mathsf{Ctx}\{T\}$ with $T \sim_{AC_\epsilon} \sigma(L)$ and $T_2 = \mathsf{Ctx}\{\sigma(R)\}$. On one hand, this notion of rewriting is more convenient, from a computational point of view, than AC_ϵ-class rewriting. In fact in the latter case the entire (possibly infinite) class must be explored looking for reducible terms. On the other hand, as we will see later, to prove the confluence of the $\rho_{\mathrm{g}}, AC_\epsilon$ relation is sufficient to get the confluence for the $\rho_{\mathrm{g}}/AC_\epsilon$ relation.

Notation. In pictures and in the rest of the section, $\mapsto_R$ denotes the one step reduction and $\mapsto\!\!\!\!\twoheadrightarrow_R$ its reflexive and transitive closure, with R any subset of rules of the ρ_{g}-Calculus. We often simply write AC_ϵ for $\sim_{AC_\epsilon}$ and $\mapsto_R$ for $\mapsto_{R,AC_\epsilon}$.

In Fig. 3 we give a graphical representation of some properties of the $\rho_{\mathrm{g}}, AC_\epsilon$ relation that will be referred to in the sequel. Their general definition can be found in [16]. The first three are ordinary properties from term rewriting, with the difference that the diagrams are closed with a step of equivalence modulo AC_ϵ. The last one says that if there exists a rewrite step from a term T, then the same step can be performed starting from any term AC_ϵ-equivalent to T.

The confluence proof, detailed in [6], is quite elaborated. This is mainly due to the non-termination of the system and to the fact that equivalence modulo AC_ϵ on terms has to be considered. We prove first a lemma showing the compatibility (see Fig. 3) of $\rho_{\mathrm{g}}, AC_\epsilon$ with AC_ϵ and thus ensuring that this relation is particularly well-behaved *w.r.t.* the congruence relation AC_ϵ. Then, we proceed by proving a number of lemmas that lead to the confluence of $\rho_{\mathrm{g}}, AC_\epsilon$ and finally we conclude on the confluence of $\rho_{\mathrm{g}}/AC_\epsilon$ using the mentioned compatibility lemma stated next.

Lemma 1 (Compatibility of $\rho_{\mathrm{g}}, AC_\epsilon$). *Compatibility with AC_ϵ holds for any rule in ρ_{g}.*

We point out that since compatibility holds for any rule of the ρ_{g}-Calculus, then it also holds for any subset of evaluation rules of the ρ_{g}-Calculus and, in particular, for the entire ρ_{g}-Calculus semantics.

In order to prove the confluence of $\rho_{\mathrm{g}}, AC_\epsilon$, we proceed using a proof technique inspired from [2], but the larger number of evaluation rules of the ρ_{g}-Calculus and the explicit treatment of the congruence relation on terms make the proof for the ρ_{g}-Calculus more complex. The main idea is to split the rules into two subsets, to show separately their confluence and to prove the confluence of the union using a commutation lemma for the two sets of rules.

In the ρ_{g}-Calculus the first subset, called the Σ-rules, consists of the two substitution rules *external sub* and *acyclic sub*, plus the δ rule. The second subset of rules, called the τ-rules, consists of all the remaining rules of the ρ_{g}-Calculus. The substitution rules represent the non-terminating rules of the ρ_{g}-Calculus. The δ rule is included in the Σ-rules, although it could be added to the τ-rules keeping this set of rules terminating. This choice is due to the fact that, because of its non-linearity, adding the δ rule to the τ-rules would have caused relevant problems in the proof of the final commutation lemma.

An important role in this part of the proof is played by the following proposition which follows immediately from [16].

Proposition 1. *1. A terminating relation locally confluent modulo AC_ϵ and compatible with AC_ϵ is confluent modulo AC_ϵ.*
2. The union of two relations commuting modulo AC_ϵ, both confluent modulo AC_ϵ and compatible with AC_ϵ, is confluent modulo AC_ϵ.

We start by showing the confluence modulo AC_ϵ of the τ-rules. This is done using Proposition 1.1). To prove the termination of the τ-rules classical but not trivial rewriting techniques are applied [12]: the lexicographic product of a polynomial interpretation on the ρ_{g}-terms and a path ordering induced by a given precedence on the operators of the ρ_{g}-Calculus syntax is used. The local confluence modulo AC_ϵ is rather easy to prove by analysis of the critical pairs.

Proposition 2. *The relation τ is confluent modulo AC_ϵ.*

Secondly, we show the confluence modulo AC_ϵ of the Σ-rules and this is the most elaborated part of the proof. The difficulties arise from the fact that the rewrite relation induced by the Σ-rules is not strongly normalizing. Thus proving local confluence is not sufficient to get confluence. In particular, both substitution rules are not terminating in the presence of cycles:

$$x\ [x = f(y), y = g(y)] \to_{ac} x\ [x = f(g(y)), y = g(y)] \to_{ac} \cdots$$
$$y\ [y = g(y)] \to_{es} g(y)\ [y = g(y)] \to_{es} \cdots$$

The idea is to prove the confluence of the relation Σ by applying the complete development method of the λ-calculus, which consists first in defining a new rewrite relation *Cpl* with the same transitive closure as Σ and secondly in proving that the relation *Cpl* satisfies the diamond property modulo AC_ϵ (see Fig. 3). Intuitively, a step of *Cpl* rewriting on a term G consists in the complete reduction of a set of redexes fixed initially in G. In other words, some redexes are marked in G and a complete development of these redexes is performed by the *Cpl* relation. Concretely, an underlining function is used to mark the redexes and reductions on underlined redexes are then performed using the following underlined version of the Σ-rules:

$$\begin{array}{llll}
(\underline{\textit{external sub}}) & \mathsf{Ctx}\{\underline{y}\}\ [y = G, E] & \to_{\underline{es}} & \mathsf{Ctx}\{G\}\ [y = G, E] \\
(\underline{\textit{acyclic sub}}) & G\ [G_0 \lll \mathsf{Ctx}\{\underline{y}\}, y = G_1, E] & \to_{\underline{ac}} & G\ [G_0 \lll \mathsf{Ctx}\{G_1\}, y = G_1, E] \\
 & & & \textit{if } x > \underline{y},\ \forall x \in \mathcal{FV}(G_0) \\
(\underline{\delta}) & (G_1 \underline{;} G_2)\ G_3 & \to_{\underline{\delta}} & G_1\ G_3; G_2\ G_3 \\
 & (G_1 \underline{;} G_2)\ [E]\ G_3 & \to_{\underline{\delta}} & (G_1\ G_3; G_2\ G_3)\ [E]
\end{array}$$

The term $x\ [x = f(\underline{y}), y = g(y)]$, for example, reach the $\underline{\Sigma}$ normal form $x\ [x = f(g(y)), y = g(y)]$ in one step. The *Cpl* rewrite relation is then defined as follows.

Definition 2. *Given the terms G_1 and G_2 in the Σ-calculus, we have that $G_1 \mapsto_{Cpl} G_2$ if there exists an underlining G_1' of G_1 such that $G_1' \mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}} G_2$ and G_2 is in normal form w.r.t. the relation $\underline{\Sigma}$.*

For example, we have $G_1 = x\ [f(x,y) \ll f(\underline{z},\underline{z}), z = g(\underline{w}), w = a] \mapsto_{\underline{\Sigma}} x\ [f(x,y) \ll f(\underline{z},\underline{z}), z = g(a), w = a] \mapsto_{\underline{\Sigma}} x\ [f(x,y) \ll f(g(a),g(a)), z = g(a), w = a] = G_2$ and thus $G_1 \mapsto_{Cpl} G_2$.

To ensure that for every choice of redexes in G_1 there exists a *Cpl* reduction, we prove that $\underline{\Sigma}$ is weakly normalizing, by defining an appropriate reduction strategy. This property, in addition to the confluence property modulo AC_ϵ for $\underline{\Sigma}$ which can be proved using Proposition 1.1), allows us to prove the diamond property modulo AC_ϵ of the *Cpl* relation.

Lemma 2. *The relation Cpl satisfies the diamond property modulo AC_ϵ.*

Notice that if the relation *Cpl* has the diamond property modulo AC_ϵ, so does its transitive closure. The confluence of the Σ relation then follows easily by noticing that the Σ relation and the *Cpl* relation have the same transitive closure, that is $\mapsto_\Sigma \subseteq \mapsto_{Cpl} \subseteq \mapsto\!\!\!\!\rightarrow_\Sigma$. The first inclusion can be proved by underling the redex reduced by the Σ step. The second inclusion follows trivially from the definition of the *Cpl* relation.

Proposition 3. *The relation Σ is confluent modulo AC_ϵ.*

Finally, we consider the union of the subsets of rules τ and Σ. General confluence holds by the previous results and by the fact that we can prove the commutation of the τ-rules with the Σ-rules (see Proposition 1.2)).

Theorem 1 (Confluence of $\rho_{\mathrm{g}}, AC_\epsilon$). *The rewrite relation $\rho_{\mathrm{g}}, AC_\epsilon$ is confluent modulo AC_ϵ.*

As mentioned at the beginning, what we aim at is a more general result about rewriting on AC_ϵ-equivalence classes of ρ_{g}-terms. This can be achieved using the confluence of $\rho_{\mathrm{g}}, AC_\epsilon$ (Theorem 1) and the compatibility property of $\rho_{\mathrm{g}}, AC_\epsilon$ with AC_ϵ (Lemma 1).

Corollary 1 (Confluence of $\rho_{\mathrm{g}}/AC_\epsilon$). *The linear ρ_{g}-Calculus is confluent.*

4 Term-Graph Rewriting in the ρ_{g}-Calculus

The standard ρ-calculus can be seen as a natural generalization of both term rewriting and λ-calculus, integrating the pattern matching capabilities of the first formalism, with the abstraction mechanism of the second one. This fact has been formalized by showing that term rewriting can be simulated in the ρ-calculus [9,10].

The ρ_g-Calculus has been already shown to be a quite expressive formalism which allows one to simulate both the plain ρ-calculus and the cyclic λ-calculus providing an homogeneous framework for pattern matching and higher-order graphical structures [7]. The possibility of representing structures with cycles and sharing naturally leads to the question asking whether first-order term-graph rewriting (TGR) can be simulated in this context. In this section we give a first positive answer. The complete proofs can be found in [6].

Several presentations have been proposed for TGR (see [18] for a survey). Here we consider an equational presentation in the style of [1], which is closer to the approach used in the ρ_g-Calculus. Given a set of variables $\mathcal{X}$ and a first-order signature $\mathcal{F}$ with symbols of fixed arity, a term-graph over $\mathcal{X}$ and $\mathcal{F}$ is a system of equations of the form $G = \{x_1 \mid x_1 = t_1, \ldots, x_n = t_n\}$ where $t_1, \ldots, t_n$ are first-order terms over $\mathcal{X}$ and $\mathcal{F}$ and the recursion variables x_i are pairwise distinct. The variable x_1 on the left represents the root of the term-graph. We call the list of equations the *body* of the term-graph and we denote it by E_G, or simply E, when the graph G is clear from the context. The empty list is denoted by ϵ. The variables $x_1, \ldots, x_n$ are bound in the term-graph by the associated recursion equation. The other variables occurring in the term-graph G are called free and the set of free variables is denoted by $\mathcal{FV}(G)$. A term-graph without free variables is called closed. We denote the collection of variables appearing in G by $\mathcal{V}ar(G)$. Two α-equivalent graphs, *i.e.* two graphs which differ only for the name of bound variables, are considered equal. Cycles may appear in the system and degenerated cycles, *i.e.* equations of the form $x = x$, are replaced by $x = \bullet$ (black hole). A term-graph is said to be in *flat form* if all its recursion equations are of the form $x = f(x_1, \ldots, x_n)$, where the variables $x, x_1, \ldots, x_n$ are not necessarily distinct from each other. In the following we will consider only term-graphs in flat form and without useless equations (garbage) that we remove automatically during rewriting. A term-graph in flat form can be easily interpreted and depicted as a graph taking the set of variables as nodes. We will use the graphical interpretation to help the intuition in the examples.

Rewriting is done by means of term-graph rewrite rules. A *term-graph rewrite rule* is a pair of term-graphs (L, R) such that L and R have the same root, L is not a single variable and $\mathcal{FV}(R) \subseteq \mathcal{FV}(L)$. We say that a rewrite rule is *left-linear* if L is a tree. In the sequel we will restrict to left-linear rewrite rules.

Definition 3 (Substitution). *A substitution $\sigma = \{x_1/G_1, \ldots, x_n/G_n\}$ is a map from variables to term-graphs. Its application to a term-graph G, denoted $\sigma(G)$ is defined as follows:*

$$\sigma(x) = \begin{cases} G_i & \text{if } x = x_i \in \{x_1, \ldots, x_n\} \\ x & \text{otherwise} \end{cases} \qquad \sigma(f(G_1, \ldots, G_n)) = f(\sigma(G_1), \ldots, \sigma(G_n))$$

$$\sigma(\{x_1 \mid x_1 = G_1, \ldots, x_n = G_n\}) = \{\sigma(x_1) \mid \sigma(x_1) = \sigma(G_1), \ldots, \sigma(x_n) = \sigma(G_n)\}$$

A rewrite rule can be applied to a term-graph G if there exists a match between its left-hand side and the graph. Formally, a *homomorphism (matching)* from a term-graph L to a term-graph G is a substitution σ such that $\sigma(L) \subseteq G$

where the inclusion means that all the recursion equations in $\sigma(L)$ are present also in G. Notice that in case of term-graphs in flat form, the homomorphism σ is simply a variable renaming.

A *redex* in a term-graph G is a pair $((L,R),\sigma)$ where (L,R) is a rule and σ is an homomorphism from the left-hand side L of the rule to G. If x is the root of L, we call $\sigma(x)$ the head of the redex.

Definition 4 (Path, position). *A* path *in a closed graph G is a sequence of function symbols interleaved by integers $p = f_1 i_1 f_2 \dots i_{n-1} f_n$ such that f_{j+1} is the i_j-th argument of f_j, for all $j = 0 \dots n$. The sequence of integers $i_1 \dots i_{n-1}$ is called the* position *of the node labeled f_n and still denoted with the letter p. By the context notation $G_{\lceil E_G \rceil_p}$ we specify that G contains the body of a graph G' at the position p.*

The notions of path and position are used to define a rewrite step. Let $((L,R),\sigma)$ be a redex occurring in G at the position p. A *rewrite step* consists in removing the equation specified by the head of the redex and in replacing it by the body of $\sigma(R)$, with a fresh choice of bound variables. Using a context notation we write $G_{\lceil \sigma(x)=t \rceil_p} \rightarrow G_{\lceil \sigma(E_R) \rceil_p}$.

The set of term-graphs of a TGR is a strict subset of the set of terms of the ρ_{g}-Calculus, modulo some obvious syntactic conventions. By abuse of notation, in the following we will consider equivalent the two notations $\{x \mid E\}$ and $x\ [E]$. A rewrite rule $(L_i, R_i) \in \mathcal{R}$ is translated into the corresponding ρ_{g}-term $L_i \twoheadrightarrow R_i$. The application of a substitution $\sigma = \{x_1/G_1, \dots x_n/G_n\}$ to a term-graph L corresponds in the ρ_{g}-Calculus to the addition of a list of constraints to the term L, that is $L\ [E]$ where $E = (x_1 = G_1, \dots, x_n = G_n)$.

A ρ_{g}-term is, in general, more complex than a flat term-graph, *i.e.* it can have garbage and nested lists of constraints. We define next the canonical form of a ρ_{g}-term G containing no abstractions and no match equations.

Definition 5 (Canonical form). *Let G be a ρ_{g}-term containing no abstractions and no match equations. We say that G is in canonical form, denoted $\overline{G}$, if it is in flat form and it contains neither garbage equations nor trivial equations of the form $x = y$.*

To reach the canonical form, we first perform the flattening and merging of the lists of equations of G and we introduce new recursion equations with fresh variables for every subterm of G. We obtain in this way a ρ_{g}-term in flat form, where the notion of flat form is similar to the one defined for term-graphs. The canonical form can then be obtained from the flat form by removing the useless equations, by means of the two substitution rules and the garbage collection rule of the ρ_{g}-Calculus. We point out that the canonical form of a ρ_{g}-term is unique, and a ρ_{g}-term in canonical form corresponds to a term-graph in flat form. Before proving the correspondence of rewritings, we need a lemma showing that matching in the ρ_{g}-Calculus is well-behaved *w.r.t.* the notion of homomorphism.

Lemma 3 (Matching). *Let G be a closed term-graph and let (L,R) be a rewrite rule, with $\mathcal{V}ar(L) = \{x_1, \dots, x_m\}$, such that L is homomorphic to G using the variable renaming $\sigma = \{x_1/x'_1, \dots, x_m/x'_m\}$.*

$$\begin{tikzcd}
 & add \arrow[dl] \arrow[dd] & \longrightarrow & s \arrow[d] & \\
s \arrow[d] & & & add \arrow[d] \arrow[dr] & \\
y_1 & y_2 & & y_1 & y_2
\end{tikzcd}$$

(a) rewrite rule: $L \to R$

$$\begin{tikzcd}
s \arrow[d] & \longrightarrow & s \arrow[d] \\
add \arrow[d] \arrow[dd, bend left] & & s \arrow[d] \\
s \arrow[d] & & add \arrow[d, bend right] \arrow[d, bend left] \\
0 & & 0
\end{tikzcd}$$

(b) rewrite step: $G \to G'$

Fig. 4. Example of rewriting

Let $E = (x_1 = x_1', \ldots, x_n = x_n', E_G)$ *with* $\{x_1, \ldots, x_n\} = \mathcal{FV}(L)$. *Then in the* ρ_{g}*-Calculus we have* $L \ll G \mapsto\!\!\!\!\!\twoheadrightarrow_{\rho g} E$ *and* $\overline{L\ [E]}$ *is homomorphic to* G.

This result guarantees the fact that if there exists an homomorphism, *i.e.* a variable renaming, between two term-graphs, in the ρ_{g}-Calculus we obtain the variable renaming (in the form of a list of recursion equations) as result of the evaluation of the matching problem generated from the two graphs. In other words, this means that if a rewrite rule can be applied to a term-graph, the application is still possible when passing to the ρ_{g}-Calculus side.

Example 3 (Matching).

Consider the two term-graphs $L = \{x_1 \mid x_1 = add(x_2, y_1), x_2 = s(y_2)\}$ and $G = \{z_0 \mid z_0 = add(z_1, z_2),\ z_1 = s(z_2), z_2 = 0\}$ (see Figure 4). Then the substitution $\sigma = \{x_1/z_0,\ x_2/z_1,\ y_1/z_2,\ y_2/z_2\}$ is an homomorphism from L to G. We show how the substitution can be obtained in the ρ_{g}-Calculus starting from the matching problem $L \ll G$. We use the notation $\mapsto\!\!\!\!\!\twoheadrightarrow_{r,s}$ to express two steps $\mapsto_r \mapsto_s$, where r and s are two ρ_{g}-rules.

$$\begin{array}{lll}
L \ll G & \mapsto_p & L \ll z_0, E_G \\
 & \mapsto\!\!\!\!\!\twoheadrightarrow_{es,gc} & add(s(y_2), y_1) \ll z_0, E_G \\
 & = & add(s(y_2), y_1) \ll z_0, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
 & \mapsto_{ac} & add(s(y_2), y_1) \ll add(z_1, z_2), z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
 & \mapsto_{dk} & s(y_2) \ll z_1, y_1 \ll z_2, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
 & \mapsto\!\!\!\!\!\twoheadrightarrow_{ac,dk} & y_2 \ll z_2, y_1 \ll z_2, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
 & \mapsto\!\!\!\!\!\twoheadrightarrow_{s} & y_2 = z_2, y_1 = z_2, E_G
\end{array}$$

We can verify then that $\overline{L\ [y_2 = z_2, y_1 = z_2, E_G]}$ is homomorphic to G. In fact, the transformation into the canonical form leads to the graph $x_1\ [x_1 = add(x_2, z_2), x_2 = s(z_2), z_2 = 0]$ and it is easy to see that the substitution $\tau = \{x_1/z_0, x_2/z_1\}$ makes this graph homomorphic to G.

We next prove that any term-graph rewrite step can be simulated in the ρ_{g}-Calculus. Since in the ρ_{g}-Calculus the rule application is at the object-level, we need to define a ρ_{g}-term encoding the position of the redex in the given term-graph.

Definition 6 (Position trace graph). *Let* $p = f\ i\ p'$ *be a path in a graph* G *and* $x_0, \ldots, x_j, \ldots$ *be a set of fresh dinstinct variables. The* position trace graph $P_p(G)$ *is recursively defined as* $P_\epsilon(G) = x_0$ *and* $P_p(G) = f(x_1, \ldots, P_p\ (G), \ldots, x_n)$

where f is of arity n and has $P_p\,(G)$ as i-th argument. We assume that every variable x_j is used only once in the construction of $P_p(G)$.

The position trace graph is then used to build a ρ_{g}-term H that pushes the rewrite rule down to the correct application position, according to the given term-graph rewrite step.

Lemma 4 (Simulation). *Given a term-graph G and a rewrite rule (L, R) such that $G_{\lceil\sigma(z)=t\rceil_p} \to G_{\lceil\sigma(E_R)\rceil_p} = G'$. Let $H = y \twoheadrightarrow (P_p(G)_{\lceil x\rceil_p} \twoheadrightarrow P_p(G)_{\lceil y\ x\rceil_p})$, then in the ρ_{g}-Calculus we have the reduction $(H\ (L \twoheadrightarrow R)\ G) \mapsto\!\!\!\to_{\rho g} G''$ with $\overline{G''}$ homomorphic to G'.*

The final ρ_{g}-term we obtain is not exactly the same as the term-graph resulting from the ρ_{g}-reduction in the TGR, and this is due to some unsharing steps that may occur in the reduction. In general, we have an homomorphism between the two graphs and this corresponds to the fact that, in presence of cycles, the ρ_{g}-term is possibly more “unraveled” than the term-graph G'.

Example 4 (Addition). Let (L, R), where $L = \{x_1 \mid x_1 = add(x_2, y_1),\ x_2 = s(y_2)\}$ and $R = \{x_1 \mid x_1 = s(x_2),\ x_2 = add(y_1, y_2)\}$, be a rewrite rule describing the addition of natural numbers. We apply this rule to the term-graph $G = \{z \mid z = s(z_0), z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0\}$ using the variable renaming $\sigma = \{x_1/z_0, x_2/z_1, y_1/z_2, y_2/z_2\}$. We obtain $G' = \{z \mid z = s(z_0), z_0 = s(z_1'), z_1' = add(z_2, z_2), z_2 = 0\}$. For a graphical representation see Fig. 4.

The corresponding reduction in the ρ_{g}-Calculus is as follows. First of all, since the rule is not applied at the head position of G, we need to define the ρ_{g}-term $H = y \twoheadrightarrow s(x) \twoheadrightarrow s(y\ x)$ that pushes down the rewrite rule to the right application position, *i.e.* under the symbol s. We obtain the reduction

$$
\begin{array}{ll}
 & (y \twoheadrightarrow s(x) \twoheadrightarrow s(y\ x))\ (L \twoheadrightarrow R)\ G \\
\mapsto\!\!\!\to_{\rho g} & s(x) \twoheadrightarrow s((L \twoheadrightarrow R)\ x)\ \ G \\
\mapsto_{\rho} & s((L \twoheadrightarrow R)\ x)\ [s(x) \ll G] \\
\mapsto_{p} & s((L \twoheadrightarrow R)\ x)\ [s(x) \ll z, E_G] \\
\mapsto\!\!\!\to_{\rho g} & s((L \twoheadrightarrow R)\ z_0)\ [E_G] \\
\mapsto_{\rho} & s(R\ [L \ll z_0])\ [E_G] \\
\mapsto\!\!\!\to_{\rho g} & s(R\ [y_1 = z_2, y_2 = z_2])\ [E_G] \\
= & s(x_1\ [x_1 = s(x_2), x_2 = add(y_1, y_2)]\ [y_1 = z_2, y_2 = z_2])\ [E_G] \\
\mapsto\!\!\!\to_{\rho g} & s(x_1\ [x_1 = s(x_2), x_2 = add(z_1, z_2)])\ [E_G] = G''
\end{array}
$$

The canonical form of G'' is then obtained by removing the useless recursion equations in E_G and merging the lists of constraints. We get $\overline{G''} = x\ [x = s(x_1), x_1 = s(x_2), x_2 = add(z_1, z_2), z_0 = 0]$. The graph $\overline{G''}$ is homomorphic (in this case even equal up to variable renaming) to the term-graph G'.

5 Conclusions

The ρ_{g}-Calculus is an extension of the ρ-calculus that allows one to represent and compute over regular infinite entities. It represents a common framework where higher-order capabilities, graphical structures and matching are primitive features, leading to a quite expressive calculus. The ρ_{g}-Calculus terms are grouped into equivalence classes defined according to the theory specified for the constraint conjunction operator, which in general is the associative, commutative, idempotent theory with neutral element ϵ. If we restrict to a linear ρ_{g}-Calculus, since all constraints are linear and this property is obviously preserved by reduction, we do not need to work modulo the idempotency axiom. We have shown here that, choosing this underlying theory for the conjunction operator for constraints, the linear ρ_{g}-Calculus enjoys the confluence property on equivalence classes of terms.

In [7] the ρ_{g}-Calculus has been shown to be an expressive formalism able to simulate both the standard ρ-calculus and the cyclic λ-calculus. We have shown in this paper that also term-graph rewrite systems can be encoded in the ρ_{g}-Calculus. More precisely we have shown that for every rewriting step in a TGR we can build a ρ_{g}-term which simulates such rewriting as a sequence of reductions in the ρ_{g}-Calculus. We have not investigated here the conservativity issue, but we believe that a positive result can be obtained exploiting the confluence property of the ρ_{g}-Calculus. The main difference between the two systems lies in the fact that rewrite rules and their control (application position) are defined at the object-level of the ρ_{g}-Calculus while in the TGR the reduction strategy is left implicit. The possibility of controlling the application of rewrite rules is particularly useful when the rewrite system is not terminating. It would be certainly interesting to define in the ρ_{g}-Calculus iteration strategies and strategies for the generic traversal of terms in order to simulate TGR rewritings guided by a given reduction strategy. A similar work has already been done for representing first-order term rewriting reductions in a typed version of the ρ-calculus [10]. Intuitively, the ρ-term encoding a first-order rewrite systems is a ρ-structure consisting of the corresponding term rewrite rules wrapped in an iterator that allows for the repetitive application of the rules. We conjecture that this approach can be adapted and generalized for handling term-graphs and simulate term-graphs reductions.

References

1. Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4):207–240, 1996.
2. Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139(2):154–233, 1997.
3. H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier. 1984. Second edition.
4. H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In *Proc. of PARLE'87*, vol. 259 of *LNCS*, pp. 141–158, Eindhoven, 1987. Springer-Verlag.
5. G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Pattern Type Systems. In *Proc. of POPL'03*, pp. 250–261. ACM Press, 2003.
6. C. Bertolissi. The graph rewriting calculus: proof of confluence and simulation of TGRs. Technical report, INRIA-LORIA, 2005.

7. C. Bertolissi, P. Baldan, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. In *Proc. of TERMGRAPH'04*, vol. 127(5) of *ENTCS*, pp. 21–41, 2005.
8. H. Cirstea, G. Faure, and C. Kirchner. A ρ-calculus of explicit constraint application. In *Proc. of WRLA'04*, vol. 117 of *ENTCS*, pp. 51–67, 2004.
9. H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, 2001.
10. H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. In *Proc. of TYPES'03*, vol. 3085 of *LNCS*, pp. 147–171, 2003.
11. A. Corradini. Term rewriting in CT_{Σ}. In *Proc. of TAPSOFT'93*, vol. 668 of *LNCS*, pp. 468–484, 1993.
12. N. Dershowitz. Termination of rewriting. *Journal Symbolic Computation*, 3(1-2):69–116, 1987.
13. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. In *Proc. of POPL'84*, pp. 83–92. ACM Press, 1984.
14. J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, CWI, 1980.
15. L. Liquori and B. Serpette. iRho: an Imperative Rewriting Calculus. In *Proc. of PPDP'04*, pp. 167–178. ACM Press, 2004.
16. E. Ohlebusch. Church-Rosser theorems for abstract reduction modulo an equivalence relation. In *Proc. of RTA'98*, vol. 1379 of *LNCS*, pp. 17–31. Springer, 1998.
17. G. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28:233–264, 1981.
18. M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, eds. *Term graph rewriting: theory and practice.* Wiley, London, 1993.
19. B. Wack. Klop counter example in the ρ-calculus. Draft notes, LORIA, Nancy, 2003.

Safe Object Composition in the Presence of Subtyping*

Lorenzo Bettini[1], Viviana Bono[2], and Silvia Likavec[2]

[1] Dipartimento di Sistemi ed Informatica, Università di Firenze, Viale Morgagni 65, 50134 Firenze, Italy
bettini@dsi.unifi.it

[2] Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Torino, Italy
{bono, likavec}@di.unito.it

Abstract. Object composition arises as a natural operation to combine objects in an object-based setting. In our incomplete objects setting it has a strong meaning, as it may combine objects with different internal states. In this paper we study how to make object composition safe in the presence of width subtyping, we propose two solutions, and discuss the alternative ones.

1 Introduction

Object composition is often advocated as an alternative to class inheritance, in that it is defined at run-time and enables dynamic object code reuse by assembling the existing components: "Ideally, you shouldn't have to create new components to achieve reuse. You should be able to get all the functionality you need just by assembling existing components through object composition." [9].

This paper is about combining safely object composition and width subtyping on objects, as their co-existence introduces run-time conflicts between methods that might have been hidden by subsumption, in situations where statically there would be no conflict. Suppose we have two objects O_1 and O_2 that we want to compose. Object O_1 has a method m_1 that calls a method m, which might be hidden by subsumption (i.e., its name does not appear in the type of the object O_1). Object O_2 has a method m_2 that calls a method m, also possibly hidden by subsumption. (Notice that it is enough if the method m is hidden in at least one of the objects.) When these two objects are composed, there is no explicit name clash at the type level, but methods m_1 and m_2 both call a method with the same name m and it is necessary: (*i*) to ensure that, after object composition, methods m_1 and m_2 continue calling the method m they were calling originally, before the composition; (*ii*) to guarantee that if one of the m's is not hidden, we expose the reference to the right one in the resulting object's "public interface". Note that if both m's were hidden by subsumption, none of them would be available to the external users anymore, and if none were hidden there would be a *true* conflict, ruled out statically.

This situation is an instance of the "width subtyping versus method addition" problem (well known in the object-based setting, see for instance [8]). This kind of name clash (named *dynamic name clash*, as opposed to the above mentioned true conflict)

* This work has been partially supported by MIUR project EOS.

should not be considered an error, but we must make sure that we solve all ambiguities, in such a way that accidental overrides do not occur.

We tackle this problem in Bono et al.'s calculus of classes and objects [4] setting, enriched with *abstract* classes (i.e., classes that can declare some methods without providing their implementation) and *incomplete* objects. In our calculus, abstract classes can be seen as *incomplete classes*, and their instances are *incomplete objects* that can be completed in an object-based fashion (by providing the bodies for the abstract methods). On the one hand, since our primary goal is to model object composition in the presence of subtyping (on complete objects only), we decided not to model any form of class-based inheritance, this being an orthogonal issue. On the other hand, we decided to work on a hybrid calculus, instead of on a pure object-based calculus, because: (*i*) abstract classes give rise to a natural notion of incomplete objects; (*ii*) this is a study we would like to incorporate in the work presented in [1,2], where we introduced a hybrid mixin-based calculus, with the aim to integrate flexible inheritance mechanisms both at the level of classes and of objects. The present calculus is simpler than the mixin-based one, but the conflict composition-subtyping we introduce and solve is the same as in the mixin-based one.

Incomplete objects can be completed in an object-based fashion via *method addition* and/or *object composition* (that composes an incomplete object with a complete one), thus providing a form of object-based inheritance. Our form of method addition does not introduce any problems with respect to subtyping, as we can add one by one only those methods that are required explicitly by the incomplete object, that is, we have total type information about the methods to be added (coming directly from the corresponding well-typed classes) *before* the actual addition takes place (see Sections 4 and 5). The conflict arises, instead, with object composition where the complete object may have more methods than the ones required by the incomplete object, and these methods may clash with some of the methods defined in the incomplete object. Notice that this problem is exactly the same as the one introduced by the general object composition example described above. Our approach to solving this problem is based on the idea of preserving the object generator together within each object. In order to avoid undesired interactions between methods while allowing the expected rebinding, every object carries the list of its methods and the list of the methods that it is still expecting.

One of the possible approaches to solving the problem seemed to be exploiting the *dictionaries* of Riecke and Stone [11]. Unfortunately, their mapping "internal label-external label" does not solve completely the ambiguities introduced by object composition in the presence of subtyping described above. In particular, there is still an ambiguity when only one of the m's is hidden by subsumption. It has to be said, however, that the original dictionaries setting is stateless, therefore method composition can be simulated by successive method additions, and dictionaries would be sufficient to model object composition. In our setting, instead, all objects (complete and incomplete) have a state (i.e., an initialized field), and object composition cannot be linearized via any form of repeated method additions. On a side note which will be useful later, we would like to recall that the calculus of [11] is "late-binding", i.e., the host object is substituted to self (in order to solve the self autoreferences) at method-invocation time, whereas our calculus is "early-binding", i.e., the host object is bound to self at object-creation time.

Table 1. Syntax of the core calculus: expressions and values

$$
\begin{array}{ll}
e ::= & const \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \mathit{fix} \\
 & \mid \mathsf{ref} \mid ! \mid := \mid \{x_i = e_i\}^{i \in I} \mid e.x \mid \mathsf{H}\ h.e \\
 & \mid \mathsf{class} \\
 & \quad \mathsf{method}\ m_j = v_{m_j};\ ^{(j \in \mathsf{M})} \\
 & \quad \mathsf{abstract}\ m_i;\ ^{(i \in \mathsf{A})} \\
 & \quad \mathsf{constructor}\ v_c \\
 & \mathsf{end} \\
 & \mid \mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A}\rangle \mid \mathsf{new}\ e \\
 & \mid \mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A}\rangle \\
 & \mid \mathsf{obj}\langle v_g, \mathsf{M}, \{m_i = v_{m_i}\}^{i \in \mathsf{M}}\rangle \\
 & \mid e_1 \longleftarrow\!\!\!+\ m_i = e_2 \mid e_1 \longleftarrow\!\!\!+\ e_2
\end{array}
$$

$$
\begin{array}{ll}
v ::= & const \mid x \mid \lambda x.e \mid \mathit{fix} \mid \mathsf{ref} \mid ! \\
 & \mid := \mid := v \mid \{x_i = v_i\}^{i \in I} \\
 & \mid \mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A}\rangle \\
 & \mid \mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A}\rangle \\
 & \mid \mathsf{obj}\langle v_g, \mathsf{M}, \{m_i = v_{m_i}\}^{i \in \mathsf{M}}\rangle
\end{array}
$$

To the best of our knowledge, it is not possible to remove all the ambiguities without either carrying along the additional information on the methods hidden by subsumption, or restricting the width subtyping. We discarded immediately the solution of re-labelling method names at object composition time, as this is untidy from a semantical point of view and impractical from an implementation one. (By re-labelling we mean the actual physical renaming of method names, and therefore all method invocations within method bodies.)

In this version of the calculus, we decided to allow width subtyping only on *complete objects*, and we present two solutions for the object composition problem. The first one is an "early-binding" version of the dictionaries approach, where the notion of "privacy-via-subsumption" of [11] is completely implemented (see Sections 4 and 5). The second one solves the conflict "method composition versus width subtyping" by relaxing the above mentioned notion (see Section 6). We argue that the first solution is more elegant formally, while the second solution is less restrictive from the point of view of the typing, and it might give better performances if implemented. In Section 7 we hint to other possible solutions of the conflict "method composition versus width subtyping".

2 Syntax

Starting from the imperative calculus of classes and objects of [4], we add the constructs to work with incomplete objects. The lambda-calculus related forms in Table 1 are standard. We describe below the other forms:

- ref, !, $:=$ are operators[1] for defining a reference to a value, for dereferencing a reference and for assigning a new value to a reference, respectively.
- $\{x_i = e_i\}^{i \in I}$ is a record and $e.x$ is the record selection operation.
- h is a set of pairs $h ::= \{\langle x, v\rangle^*\}$, where x is a variable and v is a value (first components of the pairs are all distinct). The set of pairs h is the *store*, or *heap*, found in the expression form $\mathsf{H}h.e$, where it is used for evaluating imperative side effects.

[1] Introducing ref, !, $:=$ as operators rather than standard forms such as ref$e$, $!e$, $:=e_1e_2$, simplifies the definition of evaluation contexts and proofs of properties. As noted in [12], this is just a syntactic convenience, as it is the curried version of $:=$.

In the expression $\mathsf{H}\langle x_1, v_1\rangle \dots \langle x_n, v_n\rangle.e$, H binds variables $x_1, \dots x_n$ in $v_1, \dots, v_n$ and in e.

- class
  method $m_j = v_{m_j}$; $^{(j \in \mathsf{M})}$
  abstract m_i; $^{(i \in \mathsf{A})}$
  constructor v_c
  end
  is a class written directly by the programmer. It contains two sorts of method declarations: methods m_j are the methods implemented in the class and methods m_i are the names of abstract methods introduced by the class. Each method body v_{m_j} is a function of a private field, *field*, and of *self*, which will be bound to the newly created object at instantiation time. Notice that the field does not appear explicitly in the syntax, as we model it as a lambda-abstracted variable in method bodies. For the sake of simplicity, we consider only one (private) field for each class, but this is not a restriction, as the field could be a tuple. Also, the calculus does not enforce the field to be available to all the methods of the class, but this is easily obtained by declaring it to be of type ref. If so, the field behaves like a proper instance variable (it is non-accessible, not only non-visible). To understand fully how, we refer the reader to [12]. The constructor value v_c is a function of one argument that returns the initialization value f for the private field (see Section 4 for its usage).
- $\mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A}\rangle$ is a *class value*, the result of evaluating a class expression. The function Gen_c is the *generator* used to generate its instances, the set M contains the indices of the methods defined in the class, and the set A contains the indices of the class' abstract methods. In our calculus method names are of the shape m_i, where i ranges over an index set. They are univocally identified by the index, i.e., $m_i = m_j$ if and only if $i = j$. Therefore, method names are identified with their indices.
- new e creates a function that returns a new incomplete object.
- $\mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A}\rangle$ is an incomplete object, where v_g is a generator function, M contains the indices of the methods defined in the class, and the set A contains the indices of the class's abstract methods. If the set A is empty the incomplete object becomes a complete object.
- $\mathsf{obj}\langle v_g, \mathsf{M}, \{m_i = v_{m_i}\}^{i \in \mathsf{M}}\rangle$ is a fully-fledged object that is obtained by completing an incomplete object or by reducing an object obtained via instantiation of a class that does not contain abstract methods. Its first component is a generator function (kept also for complete objects, since they can be used to complete the incomplete ones), the second component M contains the indices of the methods of the object, and the third component is the record of invocable methods.
- $e_1 \longleftarrow\!\!+\ m_i = e_2$ is the method addition operation: it adds the definition of method m_i with body e_2 to the (incomplete) object to which e_1 evaluates. It associates to the left.
- $e_1 \longleftarrow\!\!+\ e_2$ is the object composition operation: it composes the (incomplete) object to which e_1 evaluates with the complete object to which e_2 evaluates. It associates to the right.

Class values and object forms are not intended to be written directly, but are used to define the semantics of programs.

3 Examples

In this section, we provide some examples to show how incomplete objects, and object completion via method addition and object composition, can be used to design complex systems.

For readability, we will use here a slightly simplified syntax with respect to the calculus presented in Section 2: (*i*) the method parameters are listed in between "()"; (*ii*) $e_1; e_2$ is interpreted as $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$, $x \notin FV(e_2)$, coherently with a call-by-value semantics; (*iii*) references are not made explicit, thus $\mathsf{let}\ x = e\ \mathsf{in}\ x.m()$ should be intended as $\mathsf{let}\ x = \mathsf{ref}\, e\ \mathsf{in}\ (!x).m()$; (*iv*) method bodies are only sketched. Finally, $x \longleftarrow\!\!+\ e$ should be intended as $x{:}{=}(x \longleftarrow\!\!+\ e)$.

In the first example, we present a scenario where it is useful to add some functionalities to existing objects. Let us consider the development of an application that uses widgets such as graphical buttons, menus, and keyboard shortcuts. These widgets are usually associated to an event listener (e.g., a callback function), that is invoked when the user sends an event to that specific widget (e.g., one clicks the button with the mouse or chooses a menu item).

The design pattern *command* [9] is useful for implementing these scenarios, since it allows parameterization of widgets over the event handlers, and the same event handler can be reused for similar widgets (e.g., the handler for the event "save file" can be associated with a button, a menu item, or a keyboard shortcut). However, in such a context, it is convenient to simply add a function without creating a new class just for this aim. Indeed, the above mentioned pattern seems to provide a solution in pure class-based languages that normally do not supply the object method addition operation.

Within our approach, this problem can be solved with the language constructs for method addition and completion (in order to provide further functionalities needed by the prototype). For instance, we could implement the solution as in Table 2. The incomplete object `button` expects a method `onClick` that is internally called when the user clicks on the button (e.g., by the window where it is inserted, in our example the dialog `mydialog`). The incomplete object is then completed with the event listener `ClickHandler` (by using method addition). This listener is a function that has the parameter `doc` already bound to the application main document. At this point the object is completed and we can call methods on it. Notice that the added method can rely on methods of the host object (e.g., `setEnabled`). The same listener can be installed (by using method addition again) to other incomplete objects, e.g., the menu item `"Save"` and the keyboard shortcut for saving functionalities. Moreover, since we are able to act directly on instances here, our proposal enables customization of objects at runtime.

Another way to implement the same functionalities is via object composition. For instance, if saving the document requires further and complex operations, instead of including all of these in a method, it can be more convenient to include them in an object (with other methods than the one requested by the incomplete object). In particular, the incomplete object only requires the method `onClick`: the object used for completion can have more methods (hidden by subsumption). Moreover, the additional methods will be hidden in order to avoid name clashes.

Table 2. Widgets and event handler

```
let Button =                let MenuItem =             let ShortCut =
 class                       class                      class
  method display = ...        method show = ...          method setEnabled = ...
  method setEnabled = ...     method setEnabled = ...    abstract onClick;
  abstract onClick;           abstract onClick;          ...
  ...                         ...                       end in
 end in                      end in
    let ClickHandler =
     (λ doc. λ self. ... doc.save() ... self.setEnabled(false)) mydoc
     in
      let button = new Button("Save") in
      let item = new MenuItem("Save") in
      let short = new ShortCut("Ctrl+S") in
       button ←+ (OnClick = ClickHandler);
       button.display();
       button.setEnabled(true);
       mydialog.addButton(button); // now it is complete
       item ←+ (OnClick = ClickHandler);
       item.setEnabled(true);
       mymenu.addItem(item);
       short ←+ (OnClick = ClickHandler);
       short.setEnabled(true);
       system.addShortCut(short);
```

For instance, we can define the class:

```
let SaveDocument =
 class
  method onClick = λdoc.λself....
  method format = λdoc.λself....
  method save = λdoc.λself....
  method compress = λdoc.λself....
  method display = λdoc.λself....
  constructor λdoc.ref doc
 end in
```

If we instantiate this class we obtain a complete object (since there are no abstract methods), that can be used to complete the incomplete objects in Table 2. In particular, the method `display` in the complete object type will be hidden by subsumption, therefore it will not interfere with the method `display` of the class `Button` (indeed, they perform different operations). Notice that the constructor of `SaveDocument` returns a reference to the passed document instance; this is the private field that all the methods in the class can use.

4 Operational Semantics

Our approach is the one of giving the calculus a semantics as close as possible to an implementation. This has the advantage of minimizing the gap between the formal se-

Table 3. Reduction rules for standard expressions and heap expressions

$const\ v \longrightarrow \delta(const, v)$	(δ)	$\mathsf{ref}\,v \longrightarrow \mathsf{H}\langle x, v\rangle.x$	(ref)
if $\delta(const, v)$	is defined	$\mathsf{H}\langle x, v\rangle h.R[!x] \longrightarrow \mathsf{H}\langle x, v\rangle h.R[v]$	(deref)
$(\lambda x.e)\ v \longrightarrow [v/x]\ e$	(β_v)	$\mathsf{H}\langle x, v\rangle h.R[:=xv'] \longrightarrow \mathsf{H}\langle x, v'\rangle h.R[v']$	(assign)
$fix\ (\lambda x.e) \longrightarrow [fix(\lambda x.e)/x]e$	(fix)	$R[\mathsf{H}\ h.e] \longrightarrow \mathsf{H}\ h.R[e],\ R \neq [\,]$	(lift)
$\{\dots, x = v, \dots\}.x \longrightarrow v$	(select)	$\mathsf{H}\ h.\mathsf{H}\ h'.e \longrightarrow \mathsf{H}\ h\ h'.e$	(merge)

Table 4. Reduction contexts

$$R ::= [\ ] \mid R\ e \mid v\ R \mid R.x \mid \mathsf{new}\ R \mid R \diamond e \mid v \diamond R \mid R \longleftarrow\!\!\!+\ m = e \mid R \longleftarrow\!\!\!+\ e \mid v \longleftarrow\!\!\!+\ m = R \mid v \longleftarrow\!\!\!+\ R$$
$$\mid \{m_1 = v_{m_1}, \dots, m_{i-1} = v_{m_{i-1}}, m_i = R, m_{i+1} = e_{m_{i+1}}, \dots, m_n = e_{m_n}\}^{1 \le i \le n}$$

mantics and the actual implementation, thus reducing the risk of introducing errors caused by implementation issues. In fact, with this semantics, a direct implementation of our calculus in a functional programming language is quite straightforward (we are working on the implementation in OCaml).

The formal operational semantics is a set of rewriting rules including some standard rules for a lambda calculus with store, and some rules that evaluate the object-oriented related forms to records and functions, according to the object-as-record approach and Cook's class-as-generator-of-object principle. This operational semantics can be seen as something close to a denotational description for objects and classes, and this "identification" of implementation and semantical denotation is, in our opinion, a good by-product of our approach. The semantics is also intuitive since it is based on functions and records.

The operational semantics extends the one of the core calculus of classes and objects [4], therefore exploits the *Reference ML* of Wright and Felleisen [12] treatment of side-effects. To abstract from a precise set of constants, we only assume the existence of a partial function $\delta : Const \times ClosedVal \rightharpoonup ClosedVal$ that interprets the application of functional constants to closed values and yields closed values.

In Table 3, *R*'s are *reduction contexts* [5,6,10] and their definition can be found in Table 4. Reduction contexts are necessary to provide a minimal relative linear order among the creation, dereferencing and updating of heap locations, since side effects need to be evaluated in a deterministic order. We assume the reader is familiar with the treatment of imperative side-effects via reduction contexts and we refer to [3,12] for a description of the related rules.

The meaning of the class related rules in Table 5 is as follows. The rule (*class*) turns a class *expression* into a class *value* (notice that objects are created by instantiating class values). Given the parameter x for the constructor v_c of the class expression, the class generator returns a (partial) object generator that passes to the private field of the method bodies v_{m_j} the value f (returned by the constructor v_c). Recall that method bodies take parameters *field* and *self*. The record returned by the object generator has "dummy" method bodies for abstract methods, in such a way the generator is thus a function from *self* to *self*. Also, for all the methods in all generator functions, the method bodies are wrapped inside $\lambda y. \cdots y$ to delay evaluation in our call-by-value calculus. The above generator is called "partial" since it returns an object that contains

Table 5. Reduction rules for class related forms

$$\begin{array}{l}\mathsf{class}\\ \quad \mathsf{method}\ m_j = v_{m_j};\ ^{(j\in \mathsf{M})}\\ \quad \mathsf{abstract}\ m_i;\ ^{(i\in \mathsf{A})} \qquad \longrightarrow \mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A}\rangle \qquad (class)\\ \quad \mathsf{constructor}\ v_c\\ \mathsf{end}\end{array}$$

where

$$Gen_c \overset{\triangle}{=} \lambda x.\mathsf{let}\ \mathsf{f} = v_c(x)\ \ \mathsf{in}\ \lambda self.\left\{\begin{array}{ll} m_j = \lambda y.v_{m_j}\ \mathsf{f}\ self\ y & ^{(j\in \mathsf{M})}\\ m_i = \lambda y.\ self.m_i\ y & ^{(i\in \mathsf{A})}\end{array}\right\}$$

$$\mathsf{new}\ \mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A}\rangle \longrightarrow \lambda w.\mathsf{obj}\langle (Gen_c\ w), \mathsf{M}, \mathsf{A}\rangle \qquad (new\ class)$$

Table 6. Reduction rules for object manipulation

$$\mathsf{obj}\langle v_g, \mathsf{M}, \{\ldots, m_i = v_{m_i}, \ldots\}\rangle.m_i \longrightarrow v_{m_i} \qquad (obj\ sel)$$

$$\begin{array}{l}\mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A}\rangle \longleftarrow\!\!+\ \ (m_l = v_{m_l}) \longrightarrow\\ \quad \mathsf{let\ incgen} = \lambda self.\left\{\begin{array}{ll} m_j = \lambda y.\ (v_g\ self).m_j\ y & ^{(j\in \mathsf{M})}\\ m_l = \lambda y.\ v_{m_l}\ self\ y & \\ m_i = \lambda y.\ self.m_i\ y & ^{(i\in \mathsf{A}-\{l\})}\end{array}\right\}\ \mathsf{in} \qquad (meth\ add)\\ \quad\ \ \mathsf{obj}\langle \mathrm{incgen}, \mathsf{M}\cup\{l\}, \mathsf{A}-\{l\}\rangle \quad \text{where } l\in \mathsf{A}\end{array}$$

$$\begin{array}{l}\mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A}\rangle \longleftarrow\!\!+\ \ \mathsf{obj}\langle v'_g, \mathsf{P}, \{m_i = v_{m_i}\}^{i\in \mathsf{P}}\rangle \longrightarrow\\ \quad \mathsf{let\ incgen} = \mathsf{let}\ \mathrm{gen}_1 = \lambda s_1.\lambda s_2.\left\{\begin{array}{ll} m_l = \lambda y.\ (v'_g\ s_2).m_l\ y & ^{(l\in \mathsf{P}-\mathsf{A})}\\ m_r = \lambda y.\ s_1.m_r\ y & ^{(r\in \mathsf{P}\cap \mathsf{A})}\end{array}\right\}\ \mathsf{in}\\ \quad \lambda self.\left\{\begin{array}{ll} m_j = \lambda y.\ (v_g\ self).m_j\ y & ^{(j\in \mathsf{M})}\\ m_i = \lambda y.\ (v'_g\ fix(\mathrm{gen}_1\ self)).m_i\ y & ^{(i\in \mathsf{A})}\end{array}\right\}\ \mathsf{in} \qquad (obj\ comp)\\ \quad\ \ \mathsf{obj}\langle \mathrm{incgen}, \mathsf{M}\cup \mathsf{A}, fix(\mathrm{incgen})\rangle\end{array}$$

$$\mathsf{obj}\langle v_g, \mathsf{M}, \emptyset\rangle \longrightarrow \mathsf{obj}\langle v_g, \mathsf{M}, fix(v_g)\rangle \qquad (completed)$$

abstract methods that cannot be invoked (present as "dummy" methods). The actual implementation of these methods can be provided by (*meth add*), and/or (*obj comp*) given in Table 6.

The rule (*new class*) creates incomplete objects from class values. First, it applies the class generator Gen_c to an argument w, thus initializing the private field in the methods defined in the class and providing access to the object generator, that is a function from *self* to a record of methods. The application of the fixpoint operator to the object generator will create a recursive record of invokable methods (when the object is complete, see rule (*completed*) in Table 6).

The rules in Table 6 are the basic rules for manipulating objects. The rule (*obj sel*) performs method invocation on a complete object. The rules used on incomplete objects enable completing them with the method definitions they need. The rule (*meth add*) adds to an incomplete object a method m_l not yet present in the object (but required). The newly created generator function incgen (incremental generator) maps *self* to a record of methods, where concrete method definitions are taken from the object generator v_g, the abstract methods (excluding m_l) remain "dummy", and the method m_l is added. The incgen function is part of the reduct because it must be carried along in the evaluation process, in order to enable future method additions and/or object compositions. The only requirement for m_l is that the body v_{m_l} must be a function of *self*.

The rule (*obj comp*) combines two objects in such a way that the object o_2 (which must be already complete) completes the incomplete object o_1 and makes it fully functional. After completion, it will be possible to invoke all the methods that were in the interface of the *incomplete* object, i.e., those in $\mathsf{M} \cup \mathsf{A}$. The record of methods in incgen is built by taking the concrete methods from the incomplete object o_1 and by taking the concrete version of the abstract methods from the complete object o_2. During this operation we must make sure that:

(*i*) methods from the complete object o_2 that are requested by the incomplete object o_1 get their *self* rebound to the new resulting composed object (this is the reason why we need to keep the generator also for complete object values).
(*ii*) methods of o_2 that are not requested by o_1 (we call these methods *additional*) are not subject to accidental overrides.

The second point, in particular, is crucial in our context, where *additional methods* in the complete object, "hidden" because of subsumption, may clash with methods already present in the incomplete object (i.e., those in M). The above two goals are achieved altogether using the generator component gen_1 inside incgen. This generator component builds a record where the *additional* methods (i.e., the ones belonging to $\mathsf{P} - \mathsf{A}$) are correctly bound, once and for all, to their implementation in the complete object (through s_2 that will be propagated with the auto-binding of self via fixpoint). The other methods (those requested by the incomplete object, i.e., belonging to $\mathsf{P} \cap \mathsf{A}$) rely on the rebound *self*, which, in turns, uses s_1 as a "handle" to hook onto the complete object method implementations. This gen_1 is therefore exploited to supply to v'_g (the generator of the complete object o_2) the "self" record, obtained by passing the new *self* to gen_1 and then applying the fixpoint. This realizes the main idea that the method bodies of the complete object will use as implementations of the *additional* methods the ones from the complete object and not possibly accidental homonyms from the incomplete object.

The rule (*completed*) transforms an incomplete object, for which all the missing methods are provided, or which is created by instantiating the class without abstract methods, into a corresponding complete one. Since at this stage the object is complete (i.e., it does not contain any abstract methods) we can apply the fixpoint operator to obtain the recursive record of methods invokable on that object. Notice that also in the complete object value the generator is still present since it can be used in further object compositions.

It might be tempting to argue that object composition is just syntactic sugar, i.e., it can be derived via an appropriate sequence of method additions, but this is not true. In fact, when adding a method, the method does not have a state, while a complete object used in an object composition has its own internal state (i.e., it has a private field, properly initialized when the object was created via "new" from the class). Being able to choose to complete an object via composition or via a sequence of method additions (of the same methods appearing in the complete object used in the composition) gives our calculus an extra bit of flexibility.

4.1 An Example of Reduction

Let us show how the object completion works through an example. Suppose we have the following objects (for simplicity we leave out the parameter of the methods, the private field, $\lambda y.\ldots y$, and dummy methods):

$$\begin{aligned}
&o_1 = \mathsf{obj}\langle v_g^1, \mathsf{M} = \{1\}, \mathsf{A} = \{2\}\rangle \\
&o_2 = \mathsf{obj}\langle v_g^2, \mathsf{M} = \{1,2\}, \{m_1 = \lambda \mathit{self}.\ (\mathit{self}.m_1), m_2 = \lambda \mathit{self}.\ (\mathit{self}.m_1)\}\rangle \\
&o = o_1 \longleftrightarrow\!\!\!\!+\ o_2
\end{aligned}$$

where m_1 in o_2 is "hidden" (i.e., the type for o_2 will not contain the type of the method m_1 because of subsumption, see Section 5 for types). The object o_1 has m_2 as abstract, and it uses m_2 inside m_1 (with definition $m_1 = \lambda \mathit{self}.\ (\mathit{self}.m_2)$, visible in v_g^1 below). The program o loops infinitely, by first calling m_1 of o_1 (we recall that only methods belonging to the incomplete object interface are made accessible once completion has been performed). From rules (*new class*) and (*class*) we obtain the following generator for o_1 (we recall that m_2 is abstract in o_1):

$$v_g^1 = \lambda \mathit{self}.\begin{Bmatrix} m_1 = (\lambda \mathit{self}.\ (\mathit{self}.m_2))\ \mathit{self} \\ m_2 = \mathit{self}.m_2 \end{Bmatrix}$$

By applying rule (*obj comp*), o has the shape $\mathsf{obj}\langle \mathrm{incgen}, \{1,2\}, \mathit{fix}(\mathrm{incgen})\rangle$, where incgen is as follows:

$$\begin{aligned}
&\mathsf{let}\ \mathrm{incgen} = \\
&\quad \mathsf{let}\ \mathrm{gen}_1 = \lambda s_1.\lambda s_2.\begin{Bmatrix} m_1 = (v_g^2\ s_2).m_1 \\ m_2 = s_1.m_2 \end{Bmatrix} \quad \mathsf{in} \quad \lambda \mathit{self}.\begin{Bmatrix} m_1 = (v_g^1\ \mathit{self}).m_1 \\ m_2 = (v_g^2\ \mathit{fix}(\mathrm{gen}_1\ \mathit{self})).m_2 \end{Bmatrix}
\end{aligned}$$

In the following we use the notation $o_i{::}m_j$ to refer to the (fully qualified) implementation of m_j in object (or incomplete object) o_i. If we invoke m_1 on o we want that $o_1{::}m_1$ is executed, then $o_2{::}m_2$, then $o_2{::}m_1$ (i.e., no accidental override take place), which will then loop on itself. We make explicit the reduction steps performed upon the invocation of the method m_1 on object o. (we denote $\mathrm{gen}_1\mathit{fix}(\mathrm{incgen})$ by gg):

$$\begin{aligned}
&o.m_1 \to \mathit{fix}(\mathrm{incgen}).m_1 \to (v_g^1\ \mathit{fix}(\mathrm{incgen})).m_1 \to (\lambda \mathit{self}.\ (\mathit{self}.m_2))\mathit{fix}(\mathrm{incgen}) \qquad o_1{::}m_1\text{: OK} \\
&\to \mathit{fix}(\mathrm{incgen}).m_2 \to (v_g^2\ \mathit{fix}(\mathrm{gen}_1\ \mathit{fix}(\mathrm{incgen}))).m_2 \to (v_g^2\ \mathit{fix}(gg)).m_2 \to \\
&(\lambda \mathit{self}.\ (\mathit{self}.m_1))\mathit{fix}(gg) \qquad o_2{::}m_2\text{: OK} \\
&\to \mathit{fix}(gg).m_1 \to (v_g^2\ \mathit{fix}(gg)).m_1 \to (\lambda \mathit{self}.\ (\mathit{self}.m_1))\mathit{fix}(gg) \qquad o_2{::}m_1\text{: OK} \\
&\to \mathit{fix}(gg).m_1 \to (v_g^2\ \mathit{fix}(gg)).m_1 \to (\lambda \mathit{self}.\ (\mathit{self}.m_1))\mathit{fix}(gg) \qquad o_2{::}m_1\text{: OK} \\
&\ldots
\end{aligned}$$

We can see that each time the right implementation of the method was invoked and no accidental override took place, due to the usage of the additional generator gen_1.

5 Type System

Besides functional, record, and reference types, our type system has class types and object types (both for complete and incomplete objects):

$$\tau ::= \iota \mid \tau_1 \to \tau_2 \mid \tau\ \mathsf{ref} \mid \{m_i : \tau_{m_i}\}^{i \in I} \mid \mathsf{class}\langle \tau, \Sigma_\mathsf{M}, \Sigma_\mathsf{A}\rangle \mid \mathsf{obj}\langle \Sigma \rangle \mid \mathsf{obj}\langle \Sigma_\mathsf{M}, \Sigma_\mathsf{A}\rangle$$

Table 7. Typing rules for expressions

$$\frac{\mathit{typeof}(\mathit{const}) = \tau}{\Gamma \vdash \mathit{const} : \tau}\ (\mathit{const}) \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau}\ (\mathit{proj}) \qquad \frac{\Gamma, x : \tau \vdash e : \sigma}{\Gamma \vdash \lambda x.e : \tau \to \sigma}\ (\lambda)$$

$$\frac{\Gamma \vdash e_1 : \tau \to \sigma \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \sigma}\ (\mathit{app}) \qquad \frac{}{\Gamma \vdash \mathit{fix} : (\sigma \to \sigma) \to \sigma}\ (\mathit{fix}) \qquad \frac{\Gamma \vdash e : \tau \quad \Gamma \vdash \tau <: \sigma}{\Gamma \vdash e : \sigma}\ (\mathit{sub})$$

$$\frac{\Gamma \vdash e_i : \tau_i}{\Gamma \vdash \{x_i = e_i\}^{i \in I} : \{x_i : \tau_i\}}\ (\mathit{record}) \qquad \frac{\Gamma \vdash e : \{x : \sigma\}}{\Gamma \vdash e.x : \sigma}\ (\mathit{lookup})$$

$$\frac{}{\Gamma \vdash \mathsf{ref} : \tau \to \tau\ \mathsf{ref}}\ (\mathit{ref}) \qquad \frac{}{\Gamma \vdash\ ! : \tau\ \mathsf{ref} \to \tau}\ (!) \qquad \frac{}{\Gamma \vdash := \ : \tau\ \mathsf{ref} \to \tau \to \tau}\ (\mathit{assign})$$

$$\frac{\Gamma' = \Gamma, x_1 : \tau_1\ \mathsf{ref}, \ldots, x_n : \tau_n\ \mathsf{ref} \quad \Gamma' \vdash v_i : \tau_i \quad \Gamma' \vdash e : \tau}{\Gamma \vdash H\langle x_1, v_1\rangle \ldots \langle x_n, v_n\rangle.e : \tau}\ (\mathit{heap})$$

where ι is a constant type, $\to$ is the functional type operator, τ ref is the type of locations containing a value of type τ. Σ (possibly with a subscript) denotes a record type of the form $\{m_i : \tau_{m_i}\}^{i \in I}$, $I \subseteq \mathbb{N}$. If $m_i : \tau_{m_i} \in \Sigma$ we say that the *label m_i occurs* in Σ (with type τ_{m_i}). $\mathit{Labels}(\Sigma)$ denotes the set of all the labels occurring in Σ.

Typing environments are defined as $\Gamma ::= \varepsilon \mid \Gamma, x : \tau \mid \Gamma, \iota_1 <: \iota_2$ where $x \in \mathit{Var}$, τ is a well-formed type, ι_1, ι_2 are constant types, and $x, \iota_1 \notin \mathit{dom}(\Gamma)$. *Typing judgments* are the following: $\Gamma \vdash \tau_1 <: \tau_2$ (τ_1 is a subtype of τ_2), $\Gamma \vdash e : \tau$ (e has type τ).

Typing rules for lambda expressions, rules for expressions dealing with imperative side-effects via stores and rules for typing records are given in Table 7. We do not need any form of recursive types because we do not use a polymorphic *MyType* to type *self* (see, for instance, [7]). This prevents typing binary methods, but it still allows to type methods that modify *self*, which can be modelled as "void" methods.

Typing rules for class related forms are given in Table 8. In rule (*T class val*), $\mathsf{class}\langle\gamma, \{m_j : \tau_{m_j}\}^{j \in \mathsf{M}}, \{m_i : \tau_{m_i}\}^{i \in \mathsf{A}}\rangle$ is the class type where γ is the type of the generator's argument. The two record types represent types of defined methods and abstract methods respectively. Thus, $\{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}}$ is a record type representing the interface of the objects instantiated from the class. Rules (*T class val*) and (*T class*) assign the same type to their respective expressions, although deduced in a different way.

Table 9 shows the typing rules for manipulating objects. Incomplete objects are typed with the record type of defined methods and the record type of abstract methods (rule (*T inc obj*)). Notice that the type assigned to an incomplete object is similar to the one of the class the object is the instance of, but it does not contain information about the constructor. This is consistent with the fact that the constructor has already been called when an incomplete object has been created. We recall now the "dummy" methods introduced in Section 4, to justify their existence according to the typing rules: when typing an incomplete object value, "dummy" methods allow us to assign the type $\Sigma \to \Sigma$ to the generator v_g (the generator being a function from *self* to *self*). In fact, we recall that the body of "dummy" methods is a simple call to the homonym method on *self*, so the type inferred for abstract methods is consistent with the types of "dummy"

Table 8. Typing rules for class related forms

$$\frac{\text{For } j \in \mathsf{M}\text{: } \Gamma \vdash v_{m_j} : \eta \to \{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}} \to \tau_{m_j} \quad \Gamma \vdash v_c : \gamma \to \eta}{\Gamma \vdash \begin{array}{l}\mathsf{class} \\ \quad \mathsf{method}\ m_j = v_{m_j};\ ^{(j \in \mathsf{M})} \\ \quad \mathsf{abstract}\ m_i;\ ^{(i \in \mathsf{A})} \\ \quad \mathsf{constructor}\ v_c \\ \mathsf{end}\end{array} : \mathsf{class}\langle \gamma, \{m_j : \tau_{m_j}\}^{j \in \mathsf{M}}, \{m_i : \tau_{m_i}\}^{i \in \mathsf{A}} \rangle} \quad (T\ class)$$

$$(T\ class\ val)\quad \frac{\Gamma \vdash Gen_c : \gamma \to \{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}} \to \{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}}}{\Gamma \vdash \mathsf{classval}\langle Gen_c, \mathsf{M}, \mathsf{A} \rangle : \mathsf{class}\langle \gamma, \{m_j : \tau_{m_j}\}^{j \in \mathsf{M}}, \{m_i : \tau_{m_i}\}^{i \in \mathsf{A}} \rangle}$$

$$(T\ class\ inst)\quad \frac{\Gamma \vdash e : \mathsf{class}\langle \gamma, \Sigma_{\mathsf{M}}, \Sigma_{\mathsf{A}} \rangle}{\Gamma \vdash \mathsf{new}\ e : \gamma \to \mathsf{obj}\langle \Sigma_{\mathsf{M}}, \Sigma_{\mathsf{A}} \rangle}$$

Table 9. Typing rules for object-related forms

$$\frac{\Gamma \vdash v_g : \{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}} \to \{m_i : \tau_{m_i}\}^{i \in \mathsf{M} \cup \mathsf{A}}}{\Gamma \vdash \mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A} \rangle : \mathsf{obj}\langle \{m_j : \tau_{m_j}\}^{j \in \mathsf{M}}, \{m_i : \tau_{m_i}\}^{i \in \mathsf{A}} \rangle} \quad (T\ inc\ obj)$$

$$\frac{\begin{array}{l}\Gamma \vdash \{m_i = v_{m_i}\}^{i \in \mathsf{M}} : \{m_i : \tau_{m_i}\}^{i \in \mathsf{M}} \\ \Gamma \vdash v_g : \{m_i : \tau_{m_i}\}^{i \in \mathsf{M}} \to \{m_i : \tau_{m_i}\}^{i \in \mathsf{M}}\end{array}}{\Gamma \vdash \mathsf{obj}\langle v_g, \mathsf{M}, \{m_i = v_{m_i}\}^{i \in \mathsf{M}} \rangle : \mathsf{obj}\langle \{m_i : \tau_{m_i}\}^{i \in \mathsf{M}} \rangle} \quad (T\ obj) \qquad \frac{\Gamma \vdash e : \mathsf{obj}\langle \Sigma \rangle \qquad m_i : \tau_{m_i} \in \Sigma}{\Gamma \vdash e.m_i : \tau_{m_i}} \quad (T\ sel)$$

$$\frac{\Gamma \vdash e : \mathsf{obj}\langle \Sigma_{\mathsf{M}}, \Sigma_{\mathsf{A}} \rangle \quad m_l : \tau_{m_l} \in \Sigma_{\mathsf{A}} \quad \Gamma \vdash v_{m_l} : \Sigma_1 \to \tau_{m_l} \quad \Gamma \vdash (\Sigma_{\mathsf{M}} \cup \Sigma_{\mathsf{A}}) <: \Sigma_1}{\Gamma \vdash e \longleftarrow\!\!+\ (m_l = v_{m_l}) : \mathsf{obj}\langle \Sigma_{\mathsf{M}} \cup \{m_l : \tau_{m_l}\}, \Sigma_{\mathsf{A}} - \{m_l : \tau_{m_l}\} \rangle} \quad (T\ meth\ add)$$

$$\frac{\Gamma \vdash e_1 : \mathsf{obj}\langle \Sigma_{\mathsf{M}}, \Sigma_{\mathsf{A}} \rangle \quad \Gamma \vdash e_2 : \mathsf{obj}\langle \Sigma_{\mathsf{P}} \rangle \quad \Gamma \vdash \Sigma_{\mathsf{P}} <: \Sigma_{\mathsf{A}} \quad Labels(\Sigma_{\mathsf{P}}) \cap Labels(\Sigma_{\mathsf{M}}) = \emptyset}{\Gamma \vdash e_1 \longleftarrow\!\!+\ e_2 : \mathsf{obj}\langle \Sigma_{\mathsf{M}} \cup \Sigma_{\mathsf{A}} \rangle} \quad (T\ obj\ comp)$$

method bodies. Dummy methods appear only in the run-time semantics and are invisible to the programmer, thus they cannot be invoked.

Rule (*T obj*) says that the type of a complete object is the record of its method types. Notice that complete objects do not have a simple record type Σ, but an object type $\mathsf{obj}\langle \Sigma \rangle$. This is useful for distinguishing standard complete objects, which can be used for completing incomplete objects, from their internal auto-reference *self*, that has type Σ (in particular, this is to avoid *self-inflicted* object completions, unsound in our calculus). Note also that in the object expression, the first component v_g is a function from *self* to *self* (therefore typed with $\Sigma \to \Sigma$), because it works on the third component of the object, which is the record of object's methods. The only operation allowed on complete objects is method selection and it is typed as a record component selection (rule (*T sel*)).

A method m_l can be added to an incomplete object (rule (*T meth add*)), only if this method is expected by the incomplete object (abstract method). Method addition in this

Table 10. Subtyping for objects

$$\frac{}{\Gamma, \iota_1 <: \iota_2 \vdash \iota_1 <: \iota_2}\ (<:\ proj) \qquad \frac{}{\Gamma \vdash \tau <: \tau}\ (<:\ refl)$$

$$\frac{\Gamma \vdash \tau_1 <: \tau_2 \quad \Gamma \vdash \tau_2 <: \tau_3}{\Gamma \vdash \tau_1 <: \tau_3}\ (<:\ trans) \qquad \frac{\Gamma \vdash \tau' <: \tau \quad \Gamma \vdash \sigma <: \sigma'}{\Gamma \vdash \tau \to \sigma <: \tau' \to \sigma'}\ (<:\ arrow)$$

$$\frac{J \subseteq I}{\Gamma \vdash \{m_i : \sigma_i\}^{i \in I} <: \{m_j : \sigma_j\}^{j \in J}}\ (<:\ record) \qquad \frac{\Gamma \vdash \Sigma <: \Sigma'}{\Gamma \vdash \mathsf{obj}\langle\Sigma\rangle <: \mathsf{obj}\langle\Sigma'\rangle}\ (<:\ cobj)$$

case presents a sort of symmetry: the added method completes the functionalities of some already present methods, and may invoke some of them as well. Therefore, m_l's *self* type Σ_1 imposes some constraints on the type of the incomplete object that m_l is supposed to complete. Hence, the incomplete object must provide all the methods listed in Σ_1, on which the added method is parameterized. Σ_1 is inferred from m_l's body.

In the rule (*T obj comp*), Σ_{P} contains the type signatures of all the methods supported by the complete object (which may have more methods than those that are abstract in the incomplete object). The condition $\Sigma_{\mathsf{P}} <: \Sigma_{\mathsf{A}}$ ensures that the complete object contains at least all the methods needed to complete the incomplete object, and $Labels(\Sigma_{\mathsf{P}}) \cap Labels(\Sigma_{\mathsf{M}}) = \emptyset$ guarantees statically that there is no explicit name clash, i.e., there is no *true* conflict, as described in the Introduction. The type system does not rule out hidden conflicts introduced by subsumption, as these are not considered errors as long as they are taken care of dynamically, which is the goal of our run-time semantics. The resulting complete object contains the signatures of all the methods of the incomplete object.

The subtyping relation for record and object types is given in Table 10. For uniformity with respect to object types, we define only width subtyping on record types as well. However, modifying the subtyping rules in order to allow depth subtyping on record types only would be just a technicality and an orthogonal issue with respect to the subject of the paper, therefore we leave this modification out for the sake of clarity.

6 A More Flexible Solution

In the solution presented so far, the interface of an object resulting from an object composition is dictated by the incomplete object only, in the sense that, in the resulting composed object, only the methods of the incomplete object are invocable by an external user. Such a restriction on object composition was not present in the previous versions of the incomplete objects [1,2] and it is not necessary to solve the problems of dynamic name clashes, although it actually simplifies its treatment, and it allows to implement an "early-binding" version of the "privacy-via-subsumption" notion of [11].

In this section, we present an alternative solution that removes this restriction and is thus more flexible (in the sense that the interface of the composed object will contain more methods). From the point of view of typing, all we have to do is to change the typing rule for object composition in order to include all these object methods, see rule (*T obj comp*) in Table 11 (which mirrors the original rule of [2]). Now, the semantic

Table 11. The typing and reduction rule to change

$$\frac{\Gamma \vdash e_1 : \mathsf{obj}\langle \Sigma_\mathsf{M}, \Sigma_\mathsf{A} \rangle \quad \Gamma \vdash e_2 : \mathsf{obj}\langle \Sigma_\mathsf{P} \rangle \quad \Gamma \vdash \Sigma_\mathsf{P} <: \Sigma_\mathsf{A} \quad \mathit{Labels}(\Sigma_\mathsf{P}) \cap \mathit{Labels}(\Sigma_\mathsf{M}) = \emptyset}{\Gamma \vdash e_1 \longleftrightarrow\!\!\!\!\!+\; e_2 : \mathsf{obj}\langle \Sigma_\mathsf{M} \cup \Sigma_\mathsf{P} \rangle} \quad (T\ obj\ comp)$$

$$\begin{array}{l}
\mathsf{obj}\langle v_g, \mathsf{M}, \mathsf{A} \rangle \longleftrightarrow\!\!\!\!\!+\; \mathsf{obj}\langle v'_g, \mathsf{P}, \{m_i = v_{m_i}\}^{i \in \mathsf{P}} \rangle \rightarrow \\
\mathsf{let\ incgen} = \\
\quad \mathsf{let\ gen}_1 = \lambda s_1.\lambda s_2. \left\{ \begin{array}{ll} m_l = \lambda y.\ (v'_g\ s_2).m_l\ y & (l \in \mathsf{P} \cap \mathsf{M}) \\ m_r = \lambda y.\ s_1.m_r\ y & (r \in \mathsf{P} - \mathsf{M}) \end{array} \right\} \quad \mathsf{in} \\
\quad \lambda \mathit{self}. \left\{ \begin{array}{ll} m_j = \lambda y.\ (v_g\ \mathit{self}).m_j\ y & (j \in \mathsf{M}) \\ m_i = \lambda y.\ (v'_g\ \mathit{fix}(\mathsf{gen}_1\ \mathit{self})).m_i\ y & (i \in \mathsf{P} - \mathsf{M}) \end{array} \right\} \ \mathsf{in} \\
\quad \mathsf{obj}\langle \mathsf{incgen}, \mathsf{M} \cup \mathsf{A}, \mathit{fix}(\mathsf{incgen}) \rangle
\end{array} \quad (obj\ comp)$$

rule for object composition must be changed, since we do not hide all the *additional* methods: those that do not clash with methods defined by the incomplete object must be visible in the resulting object (while those that clash are obviously hidden). All we have to do is to modify slightly the rule (*obj comp*), obtaining the one given in Table 11. Notice that all of the above is enough to obtain a fully-fledged second solution.

7 Conclusions

In this paper we presented two possible solutions to solve the "method composition versus width subtyping" conflict. We remark that the high-level ideas underpinning our solutions are general. In particular, the idea of having self basically "split" into two parts when composing two objects, one taking care of the statically bound methods, the other one dealing with the dynamically bound ones, can be applied within any setting presenting the same problem. We would also like to stress the flexibility of our approach, i.e., giving the incomplete object-calculus an ML-like based operational semantics: working directly with class-as-generator-functions and objects-as-records (and their respective types) allows us to alternate between one version of the calculus and another with minimal changes, both in the typing rules and in the semantics. In particular, since the operational semantics is a set of rewriting rules into functions, we can manipulate functions to achieve our goals, instead of using ad-hoc changes to the semantics. Moreover, with respect to the dictionaries of [11] in the late-binding setting (where the host object is substituted for self in order to solve the self autoreferences at method-invocation time), our early-binding setting (where the host object is bound to self at object-creation time) allows a corresponding solution that is more oriented to an implementation and, in particular, would not suffer from overheads due to dictionary management and lookups, as the original calculus does, as pointed out in [11] itself. This is true also for the alternative solution.

The approach we chose here was to allow width subtyping on complete objects only. It is possible to have width subtyping on incomplete objects as well, if hidden method names are carried along: (*i*) in the type of the object; (*ii*) in the object itself. Solution (*i*) would imply a more restrictive typing rule for object composition, to also check the possible conflicts among non-hidden and hidden methods, and rule out such conflicts completely. We think, though, that such a solution is too restrictive, as we think this kind of name clash is not an error. Hidden method name information in the object (solution (*ii*)) would solve all possible ambiguities at run-time, but it would be less standard, as the subsumption rule would act on the object expression, not only on its type. Nevertheless, we think this solution has the advantage of being quite general, even though it might be considered not elegant, and it will be presented as future work.

As a future work plan, we are also considering the integration of a form of object-based override with a form of depth subtyping on object types, and we will study solutions to deal with the conflicts arising. In particular, an incomplete object could redefine a method that is provided through method addition or object composition. This operation is the concept of method redefinition/overriding of class-based inheritance adapted to an object-based setting. This kind of method override will enhance dynamic compositionality and flexibility and will allow the programmer to implement rather easily a chain of method invocation established at run-time (see, e.g., *decorator* and *chain of responsibilities* patterns [9]). Furthermore, we will study a composition operation between two complete objects (e.g., no abstract methods).

Acknowledgements. The authors would like to thank the anonymous referees.

References

1. L. Bettini, V. Bono, and S. Likavec. A core calculus of mixin-based incomplete objects. In *Proc. FOOL 11*, pages 29–41, 2004.
2. L. Bettini, V. Bono, and S. Likavec. Safe and Flexible Objects. In *Proc. SAC '05, OOPS track*, pages 1258–1263. ACM Press, 2005.
3. V. Bono, A. Patel, and V. Shmatikov. A core calculus of classes and mixins. In *Proc. ECOOP '99*, pages 43–66. LNCS 1628, Springer-Verlag, 1999.
4. V. Bono, A. Patel, V. Shmatikov, and J. C. Mitchell. A core calculus of classes and objects. In *Proc. MFPS '99*, volume 220, 1999.
5. E. Crank and M. Felleisen. Parameter-passing and the lambda calculus. In *Proc. POPL '91*, pages 233–244, 1991.
6. M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.
7. K. Fisher, F. Honsell, and J. C. Mitchell. A lambda-calculus of objects and method specialization. *Nordic J. of Computing*, 1(1):3–37, 1994.
8. K. Fisher and J. C. Mitchell. A delegation-based object calculus with subtyping. In *Proc. 10th International Conference on Fundamentals of Computation Theory (FCT '95)*, pages 42–61. LNCS 965, Springer-Verlag, 1995.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
10. I. Mason and C. Talcott. Programming, transforming, and proving with function abstractions and memories. In *Proc. ICALP '89*, pages 574–588. LNCS 372, Springer-Verlag, 1989.
11. J. G. Riecke and C. A. Stone. Privacy via subsumption. *Information and Computation*, 172(1):2–28, 2002. A preliminary version appeared in FOOL5.
12. A. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.

Reachability Analysis in Boxed Ambients

Nadia Busi and Gianluigi Zavattaro

Department of Computer Science, University of Bologna,
Mura A.Zamboni 7, 40127 Bologna, Italy
{busi, zavattar}@cs.unibo.it

Abstract. The decidability of reachability for pure public Mobile Ambients (i.e. without communication and restriction) has been recently investigated in [5], where a characterization of a maximal deacidable fragment is provided. A peculiar feature of such a fragment is the absence of the `open` capability for ambient dissolution. In this paper we analyse reachability in Boxed Ambients [2], the most relevant variant of Mobile Ambients in which the `open` capability is dropped and replaced by a sophisticated parent/child form of communication.

The main novelties with respect to [5] are: (i) the definition of a more general notion of reachability (called *target reachability*); (ii) the proof of the decidability of target reachability for a richer calculus also comprising parent/child communication.

1 Introduction

Mobile Ambients (MA) [7] is a well known formalism exploited to describe distributed and mobile systems in terms of *ambients*. An ambient $n[P]$ is a collection named n of active processes and nested sub-ambients P. In the pure (i.e., without communication) version of MA only three mobility primitives are used to permit ambient and process interaction: `in` and `out` for ambient movement, and `open` to dissolve an ambient boundary. More precisely, a process performs an `in` m primitive to instruct its surrounding ambient to move inside a sibling ambient named m, `out` m to exit its parent ambient named m, and `open` m to dissolve the boundary of an ambient named m located at the same level of the process.

In a recent paper [5] we have investigated the decidability of reachability for public MA, i.e. the version of the calculus without name restriction. Reachability analysis consists in verifying, given two processes P and Q, whether there exists a computation that starts from P and leads to Q. The main contribution in [5] is the proof of decidability of a generalized form of reachability (that we have called *spatial reachability*) for a fragment of pure public Mobile Ambients.

Spatial reachability permits to specify a class of possible target processes characterized by a common structure of ambient nesting and a minimal number of processes that should be hosted inside those ambients. As an example of the use of spatial reachability consider the system

$$trojan[virus|P]|notebook[Q]$$

in which a *trojan* containing a *virus* program, and running program P, attacks a notebook running the program Q. One may be interested in checking whether the process

$$notebook[\ trojan[virus|P'] \mid Q']$$

can be reached for any possible P' and Q'. Observe that *virus* is a program for which it is necessary to check the actual presence inside the ambient *trojan* in the target process (a *trojan* that does not contain a *virus* is not dangerous).

The fragment for which we have proved the decidability of spatial reachability is obtained by removing the `open` capability and by limiting the use of replication to guarded processes only (e.g., $!n[\ ]$ is not a valid process for this fragment). This fragment is maximal because we prove that if one of these two restrictions is not taken into account reachability is no longer decidable. Clearly, the most significant restriction is the elimination of the `open` capability. This mobility primitive has been eliminated also in Boxed Ambients (BA) [2], one of the most relevant dialects of MA. For this reason, it is worth to continue our analysis of reachability moving to this more general calculus.

BA is a non-pure, i.e. with communication, dialect of MA. Communication is asynchronous: messages can be spawn and stored within ambients, and processes can subsequently consume them performing a message input operation. In BA, in order to cope with the impossibility to dissolve ambient boundaries, processes can read (beyond the messages which reside in the same ambient) also the messages which are inside the parent or a child ambient.

In [2] several examples are reported in order to justify the choice of the authors to substitute the `open` capability with parent/child communication. Most of these examples are concerned with access control systems. Here, we consider a slight variation of one of those examples. Let us consider two agents a and b and a resource r.

$$P \;=\; a[P_a] \mid b[P_b] \mid r[R \mid \langle M\rangle]$$

The processes P_a and P_b control the behaviour of agents a and b, respectively. Process R is a monitor controlling the access to the resource r which contains a message $\langle M\rangle$. Assume that agent a has the right to consume the messages inside the resource r, while b has not.

Formally, this means that it could happen that the configuration

$$Q' \;=\; b[P_b] \mid r[R' \mid a[P'_a]]$$

can be reached from P for any processes P'_a and R' such that R' does not contain any message $\langle M\rangle$. Observe that we assume that the agent b is not involved in the computation

Moreover, we want to avoid the possibility to reach, starting from P, a configuration

$$Q'' \;=\; a[P''_a] \mid r[R'' \mid b[P''_b]]$$

for any processes P''_a, P''_b, and R'' such that R'' does not contain any message $\langle M\rangle$.

Similarly to the example of the *trojan* ambient previously discussed, we have a universal quantification of the processes inside the ambients a and b. However, in this example we have a more constrained characterization for the processes R' and R'': we want that no message $\langle M \rangle$ is inside these two processes. Spatial reachability is not enough expressive to cope with these kind of constraints. Indeed, spatial reachability permits to specify only *lower bounds* about the processes or messages inside the ambients (i.e. processes and ambients that must be available), while the absence of a message is an *upper bound* (i.e. an indication of a maximal amount of instances of specific processes or messages).

In this paper we introduce a more general notion of reachability that permits to express also upper bounds. We call this new property *target reachability*. We prove that target reachability is decidable for the calculus BA$^-$ that corresponds to the public fragment of the Boxed Ambients [2] in which replication is guarded and in which variables cannot be used to compose more complex messages. In this way, we avoid the possibility to produce messages of unbounded length that can be obtained in standard BA simply by recursively consuming a message, and re-emitting a new message obtained extending the previously read one.

In order to prove the decidability of target reachability in BA$^-$ we proceed as in [5] by resorting to a Petri net semantics. Namely, in [5] the Petri net semantics reduces reachability on processes into coverability on Petri nets, which is a decidable property for the class of Petri nets we have exploited. Beyond having to adapt the Petri net semantics in order to deal with communication, the main technical problem is that it is not possible to use standard coverability. This because the upper bounds of target reachability requires to put also some upper bounds on the number of tokens in specific places of the corresponding net; these upper bounds are not compatible with the standard notion of coverability. In order to solve this problem, we first define a new decidable property for Petri nets which is a generalization of both coverability and reachability; then we show how to reduce target reachability on processes to this new property for Petri nets.

The paper is structured as follows. In Section 2 we report the syntax and semantics of BA$^-$, the fragment of BA that we consider, and we define formally the new notion of target reachability. In Section 3 we prove that target reachability is decidable in BA$^-$. Finally, in Section 4 we discuss the related literature and report some conclusive remark.

2 Public Boxed Ambients

In this section we introduce a fragment of Boxed Ambients, called BA$^-$, for which we prove the decidability of reachability.

Definition 1. – **BA** – *Let Name, ranged over by n, m, ..., be a denumerable set of ambient names and Var, ranged over by x, y, ..., be a denumerable set of variables, such that $Name \cap Var = \emptyset$. The set of sequences of capabilities is defined as follows:*

$$C ::= \texttt{in}\, n \mid \texttt{out}\, n \mid \texttt{in}\, x \mid \texttt{out}\, x \mid C.C$$

The set of expressions is defined by the following grammar:

$$e \ ::= \ n \mid x \mid \ C$$

The set of locations, ranged over by η*, is* $Name \cup Var \cup \{\uparrow, \star\}$*. The set of processes is defined by the following grammar:*

$$\begin{array}{l} P \ ::= \ \mathbf{0} \ \mid \ M.P \ \mid \ P|P \ \mid \ !M.P \ \mid \ n[P] \ \mid \ x[P] \\ M ::= \ C \ \mid \ x \ \mid \ (x)^\eta \ \mid \ \langle e \rangle^\eta \end{array}$$

We use $\prod_k P$ *to denote the parallel composition of* k *instances of the process* P*, while* $\prod_{i \in 1 \ldots k} P_k$ *denotes the parallel composition of the indexed processes* P_i*.*

Boxed Ambients considers two possibile capabilities: `in` n to enter a sibling ambient named n and `out` n to exit an outer ambient named n. The capabilities in a sequence are either on a fixed ambient name or on a variable, that will be subsequently instantiated by a communication. Expressions, representing the contents of messages, comprise ambient names, variables and sequences of capabilities.

The set of processes comprises the following terms. The term $\mathbf{0}$ represents the inactive process (and it is usually omitted). $M.P$ is a process guarded by a mobility or a communication primitive: after the execution of the primitive the process behaves like P. The processes $M.P$ are referred to as *guarded processes* in the following. A process may be also the parallel composition $P|P$ of two subprocesses. The guarded replication operator $!M.P$ is used to put in parallel an unbounded amount of instances of the process $M.P$. Finally, the term $n[P]$ denotes an ambient named n containing process P, while the term $x[P]$ denotes an ambient whose name will be instantiated by a communication.

In a guarded process $M.P$ the prefix M can be a sequence of capability, or a variable or a communication primitive. Communication primitives make use of locations: location n denotes communication with a process in a child ambient with name n, location x will be instantiated with an ambient name by a communication, location $\uparrow$ denotes communication with a process in the parent ambient, and location $\star$ (often omitted) denotes local communication. The input process $(x)^\eta.P$ and the output process $\langle e \rangle^\eta.P$ permit to model communication where η denotes the location. Both input and output processes are guarded processes, and $(x)^\eta$ acts as a binder for the occurrences of variable x in P. The notions of free and bound variables (denoted by $fv(P)$ and $bv(P)$), of alpha-conversion and of closed process are defined as usual. In the following we assume that processes are closed.

The calculus BA$^-$ corresponds to the public fragment of the Boxed Ambients [2] with the following restrictions:

- Replication is guarded, i.e. it can be applied only to prefixed processes with the form $!M.P$.
- A constrained use of variables in sequences of capabilities. Namely, in Boxed Ambients it is possible to include variables in a sequence of capabilities. As

illustrated in the following example this permits to produce sequences of capabilities of unbounded length:

$$\langle \texttt{in}\, n\rangle \ \mid !(x).\langle \texttt{in}\, n.x\rangle$$

In BA$^-$ a variable cannot be a proper subsequence of a sequence of capabilities, thus excluding from the calculus the above process.

Moreover, for the sake of simplicity we consider a monadic version of the calculus.

The operational semantics is defined in terms of a structural congruence plus a reduction relation.

Definition 2. – **Structural congruence** – *The structural congruence* $\equiv$ *is the smallest congruence relation satisfying:*

$$\begin{array}{ll} P \mid \mathbf{0} \equiv P & P \mid Q \equiv Q \mid P \\ P \mid (Q \mid R) \equiv (P \mid Q) \mid R & !M.P \equiv M.P \mid !M.P \end{array}$$

Definition 3. – **Reduction relation** – *The reduction relation is the smallest relation* $\rightarrow$ *satisfying the following axioms and rules:*

$$\begin{array}{ll}
(1) & n[\texttt{in}\, m.P \mid Q] \mid m[R] \ \rightarrow\ m[n[P \mid Q] \mid R] \\
(2) & m[n[\texttt{out}\, m.P \mid Q] \mid R] \ \rightarrow\ n[P \mid Q] \mid m[R] \\
(3) & (x)P \mid \langle e\rangle Q \ \rightarrow\ P\{e/x\} \mid Q \\
(4) & (x)^n P \mid n[\langle e\rangle Q \mid R] \ \rightarrow\ P\{e/x\} \mid n[Q \mid R] \\
(5) & n[(x)^{\uparrow} P \mid Q] \mid \langle e\rangle R \ \rightarrow\ n[P\{e/x\} \mid Q] \mid R \\
(6) & n[(x)P \mid Q] \mid \langle e\rangle^n R \ \rightarrow\ n[P\{e/x\} \mid Q] \mid R \\
(7) & (x)P \mid n[\langle e\rangle^{\uparrow} Q \mid R] \ \rightarrow\ P\{e/x\} \mid n[Q \mid R] \\
(8) & \dfrac{P \ \rightarrow\ Q}{P \mid R \ \rightarrow\ Q \mid R} \\
(9) & \dfrac{P \ \rightarrow\ Q}{n[P] \ \rightarrow\ n[Q]} \\
(10) & \dfrac{P' \equiv P \quad P \ \rightarrow\ Q \quad Q' \equiv Q}{P' \ \rightarrow\ Q'}
\end{array}$$

With $P\{e/x\}$ *we denote the process obtained by substituting each free occurrence of* x *in* P *with* e *(as usual, alpha-conversion is used when necessary, to avoid name capture).*

As usual, we use $\rightarrow^+$ *to denote the transitive closure and* $\rightarrow^*$ *for the reflexive and transitive closure of* $\rightarrow$. *If* $P \rightarrow^* Q$ *we say that* Q *is a* derivative *of* P.

Axioms (1) and (2) describe the semantics of the movement primitives `in` and `out`. A process inside an ambient n can perform an $\mathtt{in}\,m$ operation in presence of a sibling ambient m; if the operation is executed then the ambient n moves inside m. If inside an ambient m there is an ambient n with a process performing an $\mathtt{out}\,m$ action, this results in moving the ambient n outside the ambient m.

Axiom (3) describes local communication. The input prefix $(x)^n P$ in axiom (4) represents a request to read a datum sent by a process located into one of the child ambients n. In axiom (5), $(x)^\uparrow P$ is a request to read a datum sent by a process located in the parent ambient. Dually, $\langle e\rangle^n P$ in axiom (6) (resp. $\langle e\rangle^\uparrow P$ in axiom (7)) is a request to send e to a process located into the child ambient n (resp. the parent ambient). Note that a direct remote communication between sibling ambients is not possible: either mobility or the intervention of the parent of the sibling ambients is required.

Rules (8) and (9) are the contextual rules that respectively indicate that a process can move also when it is in parallel with another process and when it is inside an ambient. Finally, rule (10) is used to ensure that two structurally congruent terms have the same reductions.

2.1 Target Reachability

Classical reachability analysis consists in checking if $P \to^* R$ for two given BA$^-$ processes P and R. In this paper we consider a more general notion of reachability. The main novelty is that we permit a partial description of the target process. More precisely, it is possible to impose constraints on the number of occurrences of guarded processes inside an ambient. Such constraints are both lower bounds (e.g. there must be at least one instance of the guarded process $M.P$ in a given ambient) and upper bounds (e.g. there can be at most two occurrences of message $\langle n\rangle$ in a given ambient).

We need to introduce some additional notation to denote the partial description of target processes.

We introduce a notion of normal form for process that forbids the presence of both the unreplicated and the replicated version of a guarded term in a parallel composition. Any process can be transformed in a structurally congruent process in normal form by using the monoidal axioms for parallel composition and by applying the axiom for replication from right to left (i.e., $M.P \mid !M.P \to !M.P$).

Definition 4. – **Normal form** – *A process P is in normal form if $P = \prod_i M_i.P_i \mid \prod_j !M'_j.P'_j \mid \prod_k n_k[P''_k]$ and the following conditions hold:*

– *P_i, P'_j, P''_k are in normal form for all i, j, k;*
– *if $M_i = M'_j$ then $P_i \neq P'_j$.*

Proposition 1. *Let P be a process. Then there exists a process Q in normal form such that $P \equiv Q$.*

Definition 5. – **Target** – *The set of* targets *is defined by the following grammar:*

$$\begin{array}{l} T \ ::= \ \mathbf{0} \ \mid \ \texttt{any} \ \mid \ q \leq M.P \leq q' \ \mid \ T|T \ \mid \ !M.P \ \mid \ n[T] \ \mid \ x[T] \\ M \ ::= \ C \ \mid \ x \ \mid \ (x)^{\eta} \ \mid \ \langle e \rangle^{\eta} \end{array}$$

where $q \in I\!N$ *and* $q' \in I\!N \cup \{\infty\}$ [1].

A target any requires the presence of zero or more occurrences of any process, while $q \leq M.P \leq q'$ requires the presence of k occurrences of process $M.P$, with $q \leq k \leq q'$ (if $q' = \infty$ there is no upper bound to the number of occurrences). A target $!M.P$ requires the presence of one or more occurrences of process $!M.P$. As the behaviour of processes $\prod_k !M.P$ is the same for any $k \geq 1$, we prefer to require just the presence – or the absence – of a replicated process instead of providing upper and lower bounds to the number of its occurrences. Targets can be composed in parallel, and can be nested in ambients.

As an example, consider the target $n[1 \leq \texttt{in}\, n.P \leq 2] \mid m[!(x).Q] \mid k[any \mid 3 \leq \langle m \rangle \leq \infty]$. This target requires that ambient n contains one or two occurrences of process $\texttt{in}\, n.P$, ambient m contains only occurrences of of process $!(x).Q$ (at least one occurrence is required), and ambient k contains at least three occurrences of message $\langle m \rangle$, as well as any other process. Moreover, this target also requires that there is no process at top level.

We consider only a proper subset of *well formed* targets defined as follows.

Basically, a target is well formed if the upper and lower bounds on guarded terms are satisfiable (i.e., target $3 \leq (x).\mathbf{0} \leq 2$ is not well formed) and if the presence of a replicated version of a guarded process prevents the occurrence of the nonreplicated version of the same process in a parallel composition (i.e., target $(x).\mathbf{0} \mid !(x).\mathbf{0}$ is not well formed). We also require that at most one occurrence of a replicated process is present in a parallel composition (i.e., target $!(x).\mathbf{0} \mid !(x).\mathbf{0}$ is not well formed).

Definition 6. – **Well formed target** – *A* target T *is well formed if there exists a target* $S = \prod_i q_i \leq M_i.P_i \leq q'_i \mid \prod_j !M'_j.P'_j \mid \prod_k n_k[T''_k]$ *such that the following conditions hold:*

- *processes* P_i, P'_j *are in normal form for all* i, j*;*
- *either* $T \equiv S$ *or* $T \equiv S \mid$ any*;*
- $q_i \leq q'_i$ *for all* i*;*
- *if* $M_i = M'_j$ *then* $P_i \neq P'_j$*;*
- *if* $M'_i = M'_j$ *and* $P'_i = P'_j$ *then* $i = j$*.*
- T''_k *is well formed for all* k*.*

We define the set of processes $set(T)$ that satisfy the constraints imposed by a target T. Basically, we require the presence of the required number of occurrences of a prefixed process in each ambient; if the upper bound is ∞, then also the presence of a replicated version of the process satisfies the target (i.e.,

[1] $I\!N$ denotes the set of natural numbers and we assume that $q \leq \infty$ for all $q \in I\!N$.

process $n[!\mathtt{in}\, n.\mathbf{0}]$ satisfies the target $n[3 \leq \mathtt{in}\, n.\mathbf{0} \leq \infty])$. If the target any is present, then further (different) processes may be present. As already discussed, with a replicated process in the target we just require the presence of at least one occurrence of such a replicated process.

Definition 7. – **set(T)** – *Let T be a well formed target. A process P is in $set(T)$ if $P \equiv \prod_h L_h.P_h \mid \prod_g !L'_g.P'_g \mid \prod_k n_k[P''_k]$ and there exists a target $S = \prod_i q_i \leq M_i.Q_i \geq q'_i \mid \prod_j !M'_j.Q'_j \mid \prod_k n_k[T''_k]$ such that the following conditions hold:*

– *either $T \equiv S$ or $T \equiv S \mid$ any;*
– *for all i, either $q_i \leq |\{h \mid L_h.P_h = M_i.Q_i\}| \leq q'_i$ or $q'_i = \infty$ and there exists g such that $L'_g.P'_g = M_i.Q_i$;*
– *for all j there exist g such that $L'_g.P'_g = M'_j.Q'_j$;*
– *if $T \equiv S$ then for any h there exists i such that either $L_h.P_h = M_i.Q_i$ or $L_h.P_h = M'_i.Q'_i$*
and for any g there exists j such that $L'_g.P'_g = M'_j.Q'_j$;
– *for any k, $P''_k \in set(T''_k)$.*

It is worth to note that $set(T)$ is compatible with the structural congruence relation as formalized by the following Proposition.

Proposition 2. *Let T be a target and P and Q two processes such that $P \equiv Q$. Then, $P \in set(T)$ if and only if $Q \in set(T)$.*

We are now ready to formalize the notion of *target reachability.*

Definition 8. *Let P be a process and T be a target. We say that T is a target reachable from P (denoted by $TReach(P,T)$) if there exists a process Q such that $P \rightarrow^* Q$ and $Q \in set(T)$.*

Note that target reachability is a generalization of both standard and spatial reachability. Standard reachability analysis is obtained using targets that do not contain any and such that each lower bound q_i is equal to the corresponding upper bound q'_i. Spatial reachability, on the other hand, is obtained by putting all the upper bounds to ∞, as well as any in each ambient.

We conclude this section showing how to use target reachability to analyse the access control system described in the Introduction. We simply recall that the system is initially modeled by the process

$$P \;=\; a[P_a] \mid b[P_b] \mid r[R \mid \langle M\rangle]$$

where a is an agent which has access rights to the messages stored in the resource r, while the agent b has not. Formally, we accept that $TReach(P,T')$ with

$$T' \;=\; b[P_b] \mid r[\mathtt{any} \mid 0 \leq \langle M\rangle \leq 0 \mid a[\mathtt{any}]]$$

while we assume that $TReach(P,T'')$ does not hold with

$$T'' \;=\; a[\mathtt{any}] \mid r[\mathtt{any} \mid 0 \leq \langle M\rangle \leq 0 \mid b[\mathtt{any}]]$$

Observe that we formalize the absence of message $\langle M \rangle$ using the upper bound $\langle M \rangle \leq 0$, and that we add `any` in the specification of the contents of those ambients for which we used a universal quantification in the corresponding specification reported in the Introduction. The unique ambient in which we do not make use of `any` is the ambient b in the target T', because we assume that the agent a should be able to access the resource r without any intervention from the agent b.

3 Deciding Target Reachability in BA$^-$

The target reachability problem for BA$^-$processes consists in checking if, given a target T and a process P, the target T is reachable from P. In this Section we show that target reachability is decidable for BA$^-$processes. The proof is basically an adaptation of the proof of decidability of reachability for the open free, pure and public fragment of MA presented in [5]. The main differences are due the presence of (father/child) communication in BA$^-$and to the new notion of target reachability introduced in the present work. In [5] we reduced reachability on the MA fragment to reachability on Place/Transition Petri nets. As reachability is decidable on such class of Petri nets [10], we got the decidability result for reachability on the MA fragment. To deal with target reachability, we introduce target marking reachability – a generalization of the notion of reachability on Place/Transition Petri nets – and we sketch the decidability of generalized reachability by reduction to standard reachability. Then, we show how to reduce target reachability on BA$^-$to target marking reachability on Petri nets.

We start recalling some basic definitions on Petri nets, then we define target marking reachability and we provide a sketch of the reduction result. Finally, we show how to construct the Petri net that can be used to solve the target reachability problem on BA$^-$.

3.1 P/T Nets

We recall Place/Transition nets with unweigthed flow arcs (see, e.g., [11]). Here we provide a characterization of this model which is convenient for our aims.

Definition 9. *Given a set S, a* finite multiset *over S is a function $m : S \rightarrow I\!N$ such that the set $dom(m) = \{s \in S \,|\, m(s) \neq 0\}$ is finite. The* multiplicity *of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called* empty*. The set of all finite sets over S is denoted by $\wp_{fin}(S)$.*

Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes* multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$ then $m(s) - m'(s)$ else 0. The* scalar product*, $j \cdot m$, of a number j with m is $(j \cdot m)(s) = j \cdot (m(s))$.*

To lighten the notation, we sometimes use the following abbreviation. If m is a multiset containing only one occurrence of an element s (i.e., $dom(m) = \{s\}$ and $m(s) = 1$) we denote m by only s. Multiset union is represented also by comma, i.e., $m, m' = m \oplus m'$. Let m be a multiset over S and m' a multiset over $S' \supseteq S$, such that $(m'(s') = 0)$ for each $s' \in S' \setminus S$; with abuse of notation, we sometimes use m in place of m', and vice versa.

Definition 10. *A P/T net is a pair (S, T) where S is the set of* places *and $T \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ is the set of* transitions.

Finite multisets over the set S of places are called markings*. Given a marking m and a place s, we say that the place s contains $m(s)$* tokens*.*

A P/T net is finite if both S and T are finite.

A P/T system is a triple $N = (S, T, m_0)$ where (S, T) is a P/T net and m_0 is the initial marking*.*

A transition $t = (c, p)$ is usually written in the form $c \rightarrow p$. The marking c, usually denoted by ${}^\bullet t$, is called the preset *of t and represents the tokens to be* consumed*; the marking p, usually denoted by $t^\bullet$, is called the* postset *of t and represents the tokens to be* produced*.*

A transition t is enabled *at m if ${}^\bullet t \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is written as $m \xrightarrow{t} m'$ or simply $m \rightarrow m'$ when the transition t is not relevant. We use σ, τ to range over sequences of transitions; the empty sequence is denoted by ε; let $\sigma = t_1, \ldots, t_n$, we write $m \xrightarrow{\sigma} m'$ to mean the* firing sequence *$m \xrightarrow{t_1} \cdots \xrightarrow{t_n} m'$.*

We say that m' is reachable *from m if there exists σ such that $m \xrightarrow{\sigma} m'$.*

We say that m' covers *m if $m \subseteq m'$.*

Definition 11. *Let $N = (S, T, m_0)$ be a P/T system.*

The reachability problem for marking m consists of checking if $m_0 \rightarrow^* m$.

The coverability problem for marking m consists of checking if there exists m' such that $m_0 \rightarrow^* m'$ and m' covers m.

3.2 Target Marking Reachability on P/T Nets

We introduce a generalization of both the notions of reachability and coverability on P/T nets. The idea essentially consists in providing a lower and an upper bound to the number of tokens in each place of the net, and in checking if it is possible to reach a marking that satisfies such constraints.

Definition 12. **– target marking –** *Let $N = (S, T)$ be a P/T net. A target marking of N is a pair of functions $(inf, sup) \in (S \rightarrow \mathbb{N}) \times (S \rightarrow \mathbb{N} \cup \infty)$ such that, for all $s \in S$, $inf(s) \leq sup(s)$.*

Definition 13. **– target marking satisfiablity –** *Let $N = (S, T)$ be a P/T net. A marking m of N satisfies a target marking (inf, sup) of N if, for all $s \in S$, $inf(s) \leq m(s) \leq sup(s)$.*

Definition 14. **– target marking reachability –** *Let $N = (S, T, m_0)$ be a P/T system. A target marking (inf, sup) is reachable if there exists a marking m such that $m_0 \rightarrow^* m$ and m satisfies (inf, sup).*

We note that reachability and coverability are special cases of target marking reachability. Checking reachability of marking m is equivalent to check reachability of the target marking (m, m), while checking coverability of m is equivalent to reachability of the target marking $(m, \{(s, \infty) \mid s \in S\})$.

Now we reduce the target marking reachability problem for a system N and a target marking $(\mathit{inf}, \mathit{sup})$ to standard reachability on the P/T system $\mathit{TMSys}(N, (\mathit{inf}, \mathit{sup}))$ defined below.

Definition 15. *Let $N = (S, T, m_0)$ be a P/T system and $(\mathit{inf}, \mathit{sup})$ be a target marking of N. The P/T system $\mathit{TMSys}(N, (\mathit{inf}, \mathit{sup})) = (S', T', m'_0)$ is defined as follows. Let $\mathit{normal}, \mathit{ending} \notin S$.*

$$\begin{aligned} S' = \;& S \cup \{\mathit{normal}, \mathit{ending}\} \\ T' = \;& \{(c \cup \mathit{normal}, p \cup \mathit{normal}) \mid (c, p) \in T\} \cup \\ & \{(\mathit{normal}, \mathit{ending})\} \cup \\ & \{(s \cup \mathit{ending}, \mathit{ending}) \mid \mathit{sup}(s) = \infty\} \\ m'_0 = \;& m_0 \oplus \mathit{normal} \end{aligned}$$

The set of markings $\mathit{TMMark}(N, (\mathit{inf}, \mathit{sup}))$ is defined as follows:

$$\begin{aligned} \mathit{TMMark}(N, (\mathit{inf}, \mathit{sup})) = \{m \mid \forall s \in S : \;& (\mathit{sup}(s) = \infty \Rightarrow m(s) = \mathit{inf}(s)) \wedge \\ & (\mathit{sup}(s) \neq \infty \Rightarrow \mathit{inf}(s) \leq m(s) \leq \mathit{sup}(s))\} \end{aligned}$$

Proposition 3. *Let $N = (S, T, m_0)$ be a P/T system and $(\mathit{inf}, \mathit{sup})$ be a target marking of N. The set of markings $\mathit{TMMark}(N, (\mathit{inf}, \mathit{sup}))$ is finite.*

Proposition 4. *Let $N = (S, T, m_0)$ be a P/T system and $(\mathit{inf}, \mathit{sup})$ be a target marking of N. The target marking $(\mathit{inf}, \mathit{sup})$ is reachable in N iff one of the markings in the set $\mathit{TMMark}(N, (\mathit{inf}, \mathit{sup}))$ is reachable in TMSys $(N, (\mathit{inf}, \mathit{sup}))$.*

As a consequence of the two propositions above and of the decidability of reachability on P/T systems, we get the following:

Corollary 1. *Target marking reachability is decidable for P/T systems.*

3.3 Reducing Target Reachability on Processes to Target Marking Reachability on P/T Nets

Now we show that target reachability on processes can be reduced to target marking reachability on Petri nets; by decidability of target marking reachability on Petri nets, we get the following:

Theorem 1. *Let P be a BA^- process and T be a target. The target reachability problem $\mathit{TReach}(P, T)$ is decidable.*

Given a process P and a target R, we outline the construction of a (finite) Petri system $\mathit{Sys}_{P,R}$ satisfying the following property: the check of $\mathit{TReach}(P, T)$

is equivalent to check target marking reachability of a (finite set of) target markings on $Sys_{P,R}$. The technical details concerning the construction of the net are quite similar to the ones for deciding reachability in the pure, public, open free MA fragment in [5,6], and thus omitted. Here we only recall the basic ideas.

The intuition behind this result relies on a monotonicity property of BA$^-$: because of the absence of the `open` capability, the number of "active" ambients in a process (i.e., ambients that are not guarded by any capability) cannot decrease during the computation. Moreover, as the applicability of replication is restricted to guarded processes, the number of "active" ambients in a set of structurally equivalent processes is finite (while this is not the case in , e.g., the MA process $!n[0]$). Thanks to the property explained above, in order to check target reachability it is sufficient to take into account a subset of the derivatives of P: namely, the P-derivatives whose number of active ambients is not greater than the number of active ambients in R.

Unfortunately, this subset of P-derivatives is, in general, not finite, as the processes inside an ambient can grow unlimitedly. Consider, e.g., the process $P = m[!\mathtt{in}\,n.\mathtt{out}\,n.Q] \mid n[]$: it is easy to see that, for any k, $m[\prod_k Q \mid !\mathtt{in}\,n.\mathtt{out}\,n.Q] \mid n[]$ is a derivative of P.

On the other hand, we note that the set of guarded and replicated terms that can occur as subprocesses of (the derivatives of) a process P (namely, the subterms of kind $M.P$ or $!M.P$) is finite. The idea is to borrow a technique used to map (the public fragment of) a process algebra on Petri nets. A process P is decomposed in the (finite) multiset of its guarded and replicated subprocesses that appear at top-level (i.e., occur unguarded in P); this multiset is then considered as the marking of a Place/Transition Petri net. The execution of a computational step in a process will correspond to the firing (execution) of a transition in the corresponding net. Thus, we reduce the target reachability problem for BA$^-$ processes to reachability of a finite set of target markings in a Place/Transition Petri net, which we have shown to be a decidable problem. However, differently from what happens in process algebras, where processes can be faithfully represented by a multiset of subprocesses, BA$^-$ processes have a tree-like structure that hardly fits in a flat model such as a multiset.

The solution is to consider BA$^-$ processes as composed of two kinds of components; the tree-like structure of ambients and the family of multisets of prefixed and replicated subterms contained at top level in each ambient. As an example, consider the process

$$\mathtt{in}\,n.P \mid m[\mathtt{in}\,n.P \mid \mathtt{out}\,n.Q \mid n[\mathbf{0}] \mid k[\mathbf{0}] \mid \mathtt{in}\,n.P] \mid n[\mathtt{in}\,n.P]$$

having the tree-like structure $m[n[] \mid k[]] \mid n[]$. Moreover, there is a multiset corresponding to each "node" of the tree: the multiset $\{\mathtt{in}\,n.P\}$ is associated to the root, the same multiset is associated to the n-labelled son of the root, the multiset $\{\mathtt{in}\,n.P, \mathtt{in}\,n.P, \mathtt{out}\,n.Q\}$ is associated to the n-labelled son of the m-labelled son of the root, and so on.

The Petri net we construct is composed of the following two parts: the first part is basically a finite state automaton, where the marked place represents the current tree-like structure of the process; the second part is a set of identical subnets: the marking of each subnet represents the multiset associated to a particular node of the tree. To keep the correspondence between the nodes of the tree and the multiset associated to that node, we make use of labels. A distinct label is associated to each subnet; this label will be used in the tree-like structure to label the node whose contents (i.e., the set of prefixed and replicated subprocesses contained in the ambient corresponding to the node) is represented by the subnet.

The set of possible tree-like structures we need to consider is finite, for the following reasons. First of all, the set of ambient names in a process is finite. Moreover, to verify target reachability we need to take into account only those processes whose number of active ambients is limited by the number of ambients in the process we want to reach.

The upper bound on the number of nodes in the tree-like structures also provides an upper bound to the number of identical subnets we need to decide target reachability (at most one for each active ambient). In general, the number of active ambients grows during the computation; hence, we need a mechanism to remember which subnets are currently in use and which ones are not used. When a new ambient is created, a correspondence between the node representing such a new ambient in the tree-like structure and a not yet used subnet is established, and the places of the "fresh" subnet are filled with the marking corresponding to the prefixed and replicated subprocesses contained in the newly created ambient. To this aim, each subnet is equipped with a place called *unused*, that contains a token as long as the subnet does not correspond to any node in the tree-like structure.

For example, consider the process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]]$. The relevant part of the net is depicted in Figure 1: a subset of the places, representing the tree-like structure, is depicted in the left-hand part of the figure, while the subnets are depicted in the right-hand part. We only report the subnets labelled with l_2 and l_3, and omit the two subnets labelled with l_0 (with empty marking) and with l_1 (whose marking consists of a token in place $l_1 : \texttt{out}\, m$). The computation step $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]] \rightarrow n[\texttt{out}\, m \mid m[k[!\texttt{out}\, k]]]$ corresponds to the firing of transition t in the net.

Now we are ready to describe the net that will be used to decide reachability of a target T starting from a process P.

The set of places of the net is constructed as follows. The part of the net representing the tree-like structure contains a place for each tree of size not greater than the number of active ambients in T. Each of the subnets contains a place for each prefix and replicated subprocess of process P, and a place named "unused", that remains filled as long as the subnet does not correspond to any node in the tree-like structure. Moreover, we associate a distinct label to each subnet, and all the places of the subnet will be decorated with such a label.

The net has three sets of transitions: the first set permits to model the execution of the `in` and `out` capabilities, the second set is used deal with communication, and the third set to cope with replication.

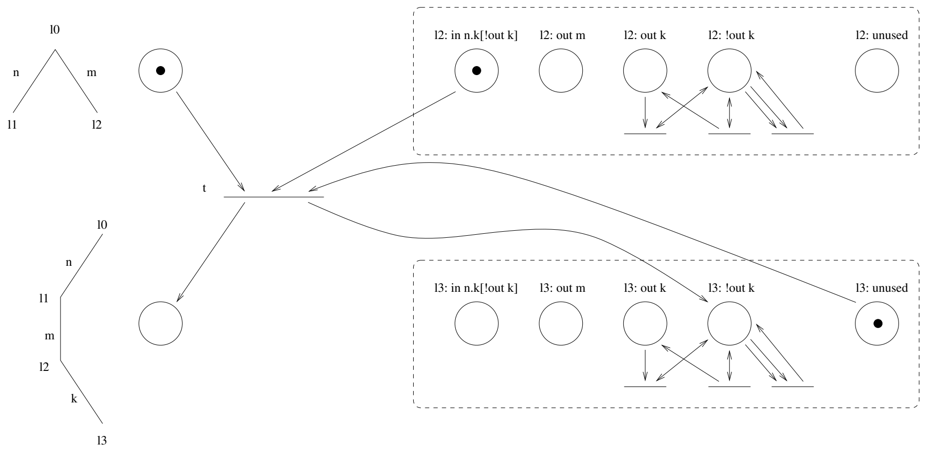


Fig. 1. A portion of the net corresponding to process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]]$

We concentrate on the first set of transitions. A capability, say, e.g., $\texttt{in}\, n$, can be executed when the following conditions are fulfilled: the tree-like structure must have a specific structure and a place corresponding to a prefixed subprocess $\texttt{in}\, n.Q$ is marked in a subnet whose label appears in the right position in the tree-like structure. Moreover, the number of active ambients created by the execution of the capability, added to the number of currently active ambients, must not exceed the number of active ambients in the target T. This condition is checked by requiring that there exist a sufficient number of "unused" places that are currently marked. The execution of the capability causes the following changes to the marking of the net: the place corresponding to the new tree-like structure is now filled and the marking of the subnet performing the $\texttt{in}\, n$ operation is updated (by adding the tokens in the places corresponding to the active prefixed and replicated subprocesses in the continuation Q). Moreover, a number of subnets equal to the number of active ambients in the continuation Q become active: their places will be filled with the tokens corresponding to the active prefixed and replicated subprocesses contained in the corresponding ambient, and the tree-like structure is updated accordingly.

The second set of transitions deal with communication and are quite similar to the rules for ambient movement. A local communication can be executed when a two places, corresponding to the prefixed processes $(x).Q_1$ and $\langle e \rangle.Q_2$, are marked in a subnet, and the number of active ambients created by the execution of the communication, added to the number of currently active ambients, must not exceed the number of active ambients in the target T. The execution of the local communication causes the following changes to the marking of the net: the place corresponding to the new tree-like structure is now filled and the marking of the subnet performing the communication is updated (by adding the tokens in the places corresponding to the active prefixed and replicated subprocesses in the continuations Q_1 and Q_2). Moreover, a number of subnets equal to the number of active ambients in the continuations Q_1 and Q_2 become active. Nonlocal communication is more involved, because the two communicating processes reside in two different ambients and it is necessary to check that such two ambients are located in the right place in the tree like structure.

The third set of transitions deals with replication. For all replicated processes $!M.Q$ occurring in P, we add to each subnet the transitions $!M.P \rightarrow !M.P, M.P$, $!M.P, M.P \rightarrow !M.P$ and $!M.P, !M.P \rightarrow !M.P$, respectively permitting to spawn a new copy of a replicated process, to absorbe a process that also appears in a replicated form in the marking, and to remove multiple occurrences of a replicated process in a marking. These transitions are used to reduce target reachability on BA$^-$to target marking reachability on the net system. An instance of such transitions is depicted in the subnet $l2$ of Figure 1.

The reachability of target T is reduced to reachability of a target marking (inf_T, sup_T) constructed as follows. We require that a token is contained in the place corresponding to the tree-like structure of T (and that the places corresponding to the other tree-like structures are empty). Moreover, for any active ambient in T,

– for any target $q \leq M.P \leq q'$ at top level in the active ambient, we require that $inf_T(l : M.P) = q$ and $sup_T(l : M.P) = q'$, where l is the label of the subnet corresponding to the active ambient;
– for any target $!M.P$ at top level in the active ambient, we require that $inf_T(l :!M.P) = sup_T(l :!M.P) = 1$;
– for any guarded or replicated process Q not occurring at top level in the active ambient, we require that $inf_T(l : Q) = 0$; if the target `any` occurs at top level in the active ambient, then we require $sup_T(l : Q) = \infty$, otherwise we impose $sup_T(l : Q) = 0$.

4 Related Work and Conclusion

Since its introduction, the calculus of Mobile Ambients [7] attracted widespread interest, and it has been used as a starting point for investigating the foundations of a great variety of mobile computing models. An interesting line of research on Mobile Ambients and its dialects is concerned with the analysis of Turing completeness and (un)decidability of properties such as divergence or termination (see, e.g., [8] and [4]). In [1] and [5] the decidability of *reachability* is investigated. In [1], Boneva and Talbot prove that reachability is undecidable even in a minimal fragment of pure Mobile Ambients in which both the restriction operator and the `open` capability are removed. A similar fragment (with the unique difference that replication can be applied to guarded processes only) is considered by Maffeis and Phillips in [8], where they show that in such a small fragment termination is undecidable while the decidability of reachability is left as on open problem. This open problem was closed in [5] where reachability is proved to be decidable.

In this paper we extend this line of research on reachability for Mobile Ambients following two intertwined directions. On the one hand we define a more general notion of reachability, called *target reachability*, which permits to specify lower and upper bounds on the possible processes within each ambient in the target process. On the other hand, we consider a richer calculus comprising also a sophisticated form of communication borrowed from Boxed Ambients [2]. As done in [5], we resort to a Petri semantics in order to prove that target reachability is decidable for a significant fragment of Boxed Ambients. With respect to the proof in [5], here we need to introduce an enhanced Petri net semantics that models also (parent/child) communication and, moreover, in order to deal with the upper bounds of target reachability we need to define a new notion of generalized reachability for Petri nets and prove that it is actually decidable for any finite Petri net.

As future work we plan to investigate the applicability of our results to recent dialects of Boxed Ambients. For instance, we intend to take under consideration the calculus in [3] where messages can be decorated with a label indicating that the message can be actually consumed only by processes residing in the same ambient, or only by processes inside the parent ambient, or only by processes stored in some child ambient.

We also plan to investigate the possibility to extend our results to BioAmbients – a calculus for the description of compartments biological systems inspired by Mobile Ambients and containing a form of parent/child interaction. We also plan to study the relationship among our results on reachability analysis and the static analysis techniques developed for Mobile Ambients and its variants.

References

1. I. Boneva and J.-M. Talbot. When Ambients Cannot be Opened. In *Proc. FOSSACS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 169-184. Springer-Verlag, Berlin, 2003. Full version to appear in Theoretical Computer Science, Elsevier.
2. M. Bugliesi, G. Castagna and S. Crafa Access Control for Mobile Agents: The Calculus of Boxed Ambients. ACM Transactions on Programming Languages and Systems, 26(1):57-124. ACM Press, 2004.
3. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication and Mobility Control in Boxed Ambients. To appear in the *Journal of Information & Computation*. Academic press.
4. N. Busi and G. Zavattaro. On the Expressive Power of Movement and Restriction in Pure Mobile Ambients. *in Theoretical Computer Science*, 322:477–515, Elsevier, 2004.
5. N. Busi and G. Zavattaro. Deciding Reachability in Mobile Ambients. In *Proc. ESOP'05*, volume 3444 of *Lecture Notes in Computer Science*, pages 248-262. Springer-Verlag, Berlin, 2005.
6. N. Busi and G. Zavattaro, Deciding Reachability in Mobile Ambients - Extended version. Available at `http://www.cs.unibo.it/~busi/papers/MA05.pdf`
7. L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
8. S. Maffeis and I. Phillips. On the Computational Strength of Pure Mobile Ambients. To appear in *Theoretical Computer Science*, Elsevier.

9. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325(1):141–167, Elsevier, 2004.
10. C. Reutenauer. *The Mathematics of Petri Nets.* Masson, 1988.
11. W. Reisig. *Petri nets: An Introduction.* EATCS Monographs in Computer Science, Springer, 1985.

Error Mining for Regular Expression Patterns

Giuseppe Castagna[1], Dario Colazzo[2], and Alain Frisch[3]

[1] CNRS, Ecole Normale Supérieure de Paris, France
[2] LRI, Université Paris Sud, Orsay, France
[3] INRIA, Rocquencourt, France

Abstract. In the design of type systems for XML programming languages based on regular expression types and patterns the focus has been over *result analysis*, with the main aim of statically checking that a transformation always yields data of an expected output type. While being crucial for correct program composition, result analysis is not sufficient to guarantee that patterns used in the transformation are correct. In this paper we motivate the need of static detection of incorrect patterns, and provide a formal characterization based on pattern matching operational semantics, together with locally exact type analysis techniques to statically detect them.

1 Introduction

Current type systems for query and transformation languages for XML data, such as those of XQuery [DFF$^+$05], ℂDuce [BCF03], C$_\omega$ [BMS05], XDuce [Hos00] mainly aim at result type analysis, that is at statically inferring the output type of a query or transformation function, starting from its structural requirements (XPath paths or ML-like patterns) and theinput type.

Result analysis has a crucial importance as by statically knowing the output type, we can check if it is included in the input type required by some other application. Hence, being the output type an upper bound for values resulted by the query/function (type soundness), result type analysis constitutes a powerful tool for sound system composition.

Unfortunately, while result analysis is often sufficient for programming languages that deal with simple data structures, this is no longer true for languages manipulating complex data structures as it is the case for XML.

Working on XML trees requires two different powerful language primitives: (i) iterator primitives in order to navigate XML trees and (ii) deconstructing primitives (usually called patterns or templates) in order to capture subparts of their structure. The result analysis is often sufficient to verify correctness of iterators, but it is useless to spot errors hidden inside the deconstructing primitives. In the context of XML processing languages two different classes of deconstructing primitives can be found: path expressions (usually XPath paths, but also the "dot" navigation of C$_\omega$) and regular expression patterns.

Path expressions are navigational primitives that finger where to capture data substructures. They closely resemble the homonymous primitives used by OQL in the contexts of OODB query languages with the difference that they return sets or sequences of elements, those that can be reached by the paths they define. These are at the basis of standard languages such as XSLT or XQuery.

More recently a new kind of deconstructing primitives was proposed, regular expression patterns [HP01], which extends by regular expressions the pattern matching as popularized by functional languages such as ML and Haskell. Regular expression patterns were first introduced in the XDuce [HP00] programming language and then adopted by other projects such as ℂDuce [BCF03] and its query language ℂQL [BCM05], Xtatic [GP03], Scala [OAC$^+$04], XHaskell [LS04] as well as the extension of Haskell proposed in [BFS04].

As we said result analysis is not sufficient to spot errors a programmer would have done in defining or writing paths or regular expressions patterns (from now on, "patterns" for short): indeed with the current technology a program containing errors in paths/patterns can (and in the absence of other errors, will) type check. In general, it is difficult to precisely characterize the class of wrong patterns or paths. An approximation is to consider as wrong those patterns/paths which contains subparts that are meaningless that is, roughly, that they are never be used whatever the input of the path/pattern is.

The problem of characterizing and detecting correctness of XPath expressions has been recently tackled by Colazzo et al. [Col04, CGMS04]. The authors provide quite a precise type analysis technique that, by checking the absence of matching between paths and input types, statically detects empty sub-queries of XQuery queries.

In this work we study the same problem for the other family of deconstructing primitives, that is regular expression patterns. In particular, we show how to formally define and statically detect patterns that contain subpatterns which are "never used". We develop our approach for the pattern algebra of the ℂDuce programming language since this algebra is the most general among those of the cited languages: the pattern algebras of the other languages are subsumed by the one of ℂDuce, therefore our technique can be straightforwardly adapted to them with few or no modifications.

To that end we study how such a kind of *local* errors can be (i) formally characterised in terms of operational semantics (hence, independently from a particular set of type rules) and (ii) statically detected by means of some improvements of the existing type systems. In particular, before defining the extended type system, we give several examples of practical and theoretical motivations of our study and, then, we give a formal characterization of the class of errors we want to mine. As we will see, the problem is not obvious to solve, due to possible high irregularities in types and patterns. However, the rich type algebra of ℂDuce will ensure a sound and complete analysis for a single pattern matching. The analysis reports a set of sub-patterns which are never used considering a given input type for the pattern. This analysis can be added to the ℂDuce type-checker. Of course, the analysis is then only locally exact (it is exact assuming that the type-checker gives the argument of the pattern matching a type which exactly denotes all the possible values of this argument at run-time), but globally sound (if it reports an unused sub-pattern, this sub-pattern is really useless and hence probably wrong).

Overview. The article is organized as follows. In the next section we provide some practical examples of the kind of errors we want to statically detect and that elude current type checkers technology. We also show the relevance of such errors and the importance to detect them when programming XML transformations. In Section 3 we formally define the class of errors we are interested in, together with a sound a complete analysis to statically detect them. In Section 4 we discuss the characteristic of our analysis and show how to embed it in existing type checkers.

2 Motivating Examples

In writing programs that process typed XML data, programmers are very likely to specify in their patterns, only the part of the schema that is strictly necessary to recover desired data. This is almost always the case when writing programs that query XML data, but even in the context of XML transformation programs, very often, only a sub part of the input structure must be matched and processed.

Partial specification of structural requirements can be specified in regular expression patterns by using the wildcard pattern "_" which matches every value. This is of crucial importance as it enormously simplifies coding of programs and makes them more robust to possible evolutions of the data schemas. However, at the same time, the extensive use of the wildcard patterns is an important (but not exclusive) source of the kind of errors that we target in this paper: the common practise of a massive use of wildcard patterns, thus, makes the presence of undetected errors very likely, whence the importance of our analysis.

As we will explain, the presence of incorrect patterns may strongly compromise quality of system behavior, as incorrect patterns never match data, and, as a consequence, some desired data may end up to not contribute to partial and/or final results, without having the possibility of becoming aware of this problem at compile time. So, negative effects of this problem may be visible only by careful observing the results of the programs. This makes error detection quite difficult and the subsequent debugging very hard.

Let us see all of this on a standard example, and use it to introduce ℂDuce patterns. Consider the following schema:

```
type Bib    = <bib>[Book*]
type Book   = <book year=String>[Title (Author+|Editor+) Price?]
type Author = <author>[Last First]
type Editor = <editor>[Last First]
type Title  = <title>[PCDATA]
type Last   = <last>[PCDATA]
type First  = <first>[PCDATA]
type Price  = <price>[PCDATA]
```

The declarations above should not pose any problem to the reader familiar with XML, DTD, and XML Schema. The type `Bib` classifies XML-trees rooted at tag `bib` that delimits a possibly empty list of books. These are elements with tag `book`, an attribute `year`, and containing a sequence formed exactly by one element title, followed by either a non empty list of author elements, or a non empty list of editor elements, and ended by an optional element price. Title elements are tagged by `title` and contain a sequence of characters, that is, a string (in XML terminology "parsed character data", i.e. PCDATA). The other declarations have similar explanations.

The declarations above give a rather complete presentation of ℂDuce types: there are XML types, that are formed by a tag part and a sequence type (denoted by square brackets). The content of a sequence type is described by a regular expression on types, that is, by the juxtaposition, the application of `*`, `+`, `?` operators, and the union `|` of types. Besides these types there also are: (i) values which are considered singleton types, so for instance `"Colazzo"` is the type that contains only the string `"Colazzo"`, (ii) intersection of types, denoted by *s&t* that contains all the values that have both type *s* and type *t*, (iii) difference "\" of types, so that the type

```
<book year=String"1999">[Title (Author+|Editor+) Price?]
```

is the type of all books *not* published in 1999, (iv) the `Any` type, which is the type of all values and which is often denoted as "`_`", especially in patterns, and its complement the `Empty` type.

Patterns are just types enriched with capture variables. For instance the pattern `<bib>[(x::Book)*]` captures in `x` the sequence of all books of a bibliography. Indeed, the `*` indicates that the pattern `x::Book` must be applied to every element of the sequence delimited by `<bib>`. When matched against an element, the pattern `x::Book` captures this element in the sequence `x`, provided that the element is of type `Book`. Patterns can then be used in match expressions:

```
match biblio with   <bib>[ (x::Book)* ] -> x
```

This expression matches `biblio` against our pattern and returns `x` as result, thus it makes nothing but stripping the `<bib>` tag from `biblio`. Note that if we knew that `biblio` has type `Bib`, then we could have used the pattern `<bib>[(x::Any)*]` (or its syntactic sugar `<bib>[(x::_)*]` since we statically know that all elements have type `Book`.

Besides capture variables there is just one further difference between patterns and types, namely the union operator `|` is commutative for types while it obeys a first match policy in patterns. So for instance the following expression returns the sequence of all the books published in 1999:

```
match biblio with  <bib>[ ( (x::<book year="1999">_) | _ )* ] -> x
```

Again, the pattern `((x::<book year="1999">_) | _ )` is applied to each element of the sequence. This pattern first checks whether the element has the tag `<book year="1999">` whatever its sequence of elements is, and if it is so it captures it in `x`; otherwise it matches the element against the pattern "`_`", which always succeeds without capturing anything (in this way it discards the element). Note that, if we had instead used `<bib>[ (x::<book year="1999">_)* ]` this pattern would have succeeded only for bibliographies composed only by books published in 1999.

After this brief introduction to regular expression patterns, let us show the pattern errors we target in this work. Suppose that, for each book, we need to extract all titles, together with relative authors or editors. In ℂDuce we can write the following function:[1]

[1] This is not the best way to write this function in ℂDuce but it serves to outline the problem.

```
let extract(x : [Book*]) : [(Title (Author+|Editor+))*] =
  transform x with
    <book>[z::<title>_ y::(<author>_ |<editor>_ )+ _*] -> z @ y
```

The function `extract` takes a possibly empty sequence of books and returns a possibly empty sequence where a title alternates with a non-empty uniform sequence of authors or editors. The expression `transform` applies the pattern to each element of the sequence `x` and returns the concatenation of all the results of the patterns that have succeeded. The pattern captures the title in the sequence variable `z`, the sequence of authors or editors in the sequence variable `y`, and returns the concatenation of `z` and `y`.

Imagine now that the programmer had put a typo in the pattern, writing instead:

```
    <book>[z::<tite>_ y::(<author>_ |<editor>_ )+ _*] -> z @ y
```

then the ℂDuce compiler would signal an error (actually, a warning), since no book starts with a <**tite**> element, so this pattern cannot ever match. But if the typo had been in the author (or in the editor) pattern:

```
    <book>[z::<title>_ y::(<autor>_ |<editor>_ )+ _*] -> z @ y
```

then no error would be signalled since the pattern can still match editors. However, all the books with authors would be filtered out from the result, which would then be of type `[(Title Editor+)*]`. If we had used a weaker pattern

```
    <book>[z::<title>_ y::(<autor>_ |<editor>_ )* _*] -> z @ y
```

in which we traded a + for a *, then the transform would return all the titles but only the editor lists (the author lists being matched by the final `_*` pattern), yielding a result of type `[(Title Editor*)*]`. In this case an error would be signalled but just because we used a very precise type for the function: had we specified a less precise type such as `[(Title (Author|Editor)*)*]`, then the error would have passed unnoticed again.

This kind of errors is very frequent when using patterns to code XPath-like expressions. For instance in ℂDuce it is possible to write a XPath-like expression of the form *e*/*t*, which is syntactic sugar for

```
   transform e with <_>[ (x::t|_)*] -> x
```

Thus for instance the following query extracts all titles from the database `biblio` of type `Bib`:

```
   [biblio]/<book>_/<title>_
```

If we replace `title` with `tite`, thus introducing an incorrect pattern, ℂDuce type system correctly rises a warning stating that the pattern never matches, as emptiness of a whole expression can be directly checked by result analysis. However, if we want to extract each title together with the relative price, we can write

```
   [bib]/<book>_/(<title>_ | <prize>_ )
```

which contains an error, as `prize` occurs instead of `price`. But since the result is not empty no warning is raised. Here, the error is hidden by the fact that the pattern is partially correct : it does find some match, even if, locally, `<prize>_` never matches, hence is incorrect. Note that, as price is optional, by looking at the query output, when seeing only titles, we do not know whether prices are not present in that database or something else went wrong .... This further motivates improvements of the type system in order to check at static time that each sub pattern will match in at least one evaluation. The subpattern `<prize>_` does not meet this property.

As previous examples showed, undetected wrong sub-patterns are mainly introduced by the pattern `_` (i.e. `Any`), which always matches and, thus, covers surrounding failures. However, this is far from being the only case. The error in the last version of `extract` function was covered by the final `_*` which captured all the authors, and possibly the price, of a book. However, the error would have been hidden even in the absence of `_*`. Indeed, if we had written

```
<book>[z::<title>_ y::(<autor>_ |<editor>_ )* Price?] -> z @ y
```

then all the books with authors would have been filtered out, yielding a result of type `[(Title Editor+)*]`. But again no type warning would be issued.

Finally, note that even if we used typos to introduce errors, other errors are possible, as well, of more conceptual nature. Imagine we want to select all books in which one author is either "Frisch" or "Colazzo", here is an example of hidden errors without any typo

```
let extract(x : [Book*]) : [Title*] =
  transform x with
    <book>[z::Title _*
           ( <author>[<last>"Colazzo"]
           | <author>[<last>"Frisch" _]) _*]  -> z
```

in this case no book with Colazzo as author will be selected since, contrary to the pattern for "Frisch", there is no pattern to match the `<first>_` element. But again no error is signalled.

The technique to detect these errors will be presented in next section. We will work on binary trees to stay as close as possible to the implementation level (as these are the structures actually used in XDuce and ℂDuce to encode regular expressions) but also because the presentation will result far simpler. The whole theory can be then easily extended to general cases. As we will see, thanks to powerful type combinators of ℂDuce (union, negation, and intersection) the type rules that we provide are quite intuitive and simple. Also, the efficient implementation of ℂDuce type system ensures good performance of the newly introduced analysis, which relies on the same basic operators.

3 Error Mining

Let us start by defining a simplified data model and type/pattern algebra. We are going to work with binary trees whose leaves are taken from a set of constants $\mathbb{C}$. We use the meta-variable c to range over constants. In ℂDuce, leaves can also be functions, and the trees have other kind of nodes (to deal with XML attributes and records).

Definition 1. *A* value *is a finite term produced by the following grammar:*

$$v \ ::= \ c \mid (v_1, v_2)$$

□

Now let us define the type and pattern algebra. For what concerns the contribution of this paper, namely the detection of useless sub-patterns, we do not need capture variables. This simplification allows us to give a common definition for types and patterns. However, we need an explicit way to localize sub-patterns. To do this, we annotate relevant sub-patterns with *marks* ranged over by the meta-variable ι. These marks can be thought as locations in the source code kept during the parsing phase and used to display error messages and warnings. Basic types are ranged over by the meta-variable b. A basic type denotes a set of constants. We write $(c : b)$ if the constant c belongs to the basic type b (the same constant can belong to many basic types).

Definition 2. *A* pattern *is a possibly infinite term produced by the following grammar:*

$$p \ ::= \ b \ \mid \ (p_1, p_2) \ \mid \ p_1 | p_2 \ \mid \ p_1 \,\&\, p_2 \ \mid \ \neg p \ \mid \ \mathbf{0} \ \mid \ \mathbf{1} \ \mid \ p^{\iota}$$

with two additional requirements:

1. *(regularity) the term must be a regular tree (only but a finite number of different sub-terms);*
2. *(contractivity) any infinite branch must contain an infinite number of pair nodes* (p_1, p_2). □

Where b ranges overs basic types and $\mathbf{0}$ and $\mathbf{1}$ respectively represent the `Empty` and `Any` types. The infiniteness of patterns accounts for recursive types. Of course these types must be machine representable, therefore we impose a condition of regularity. The contractivity instead rules out meaningless terms such as $p = \neg p$ (that is, an infinite unary tree where all nodes are labeled by $\neg$). Both conditions are standard when dealing with recursive types (e.g. see [AC93]).

Note that these patterns are more than enough to encode all the types we used in Section 2: sequences can be encoded à la Lisp by pairs, pairs can also be used to encode XML types, while regular expression types are encoded by recursive patterns . So for instance if we do not consider attributes, the type

```
    type Book = <book>[Title (Author+|Editor+) Price?]
```

can be encoded as $\mathit{Book} = (\text{‘}\mathit{book}, (\mathit{Title}, X|Y))$, $X = (\mathit{Author}, X|(\mathit{Price}, \text{‘}\mathit{nil})|\text{‘}\mathit{nil})$ and $Y = (\mathit{Editor}, Y|(\mathit{Price}, \text{‘}\mathit{nil})|\text{‘}\mathit{nil})$, where *‘book* and *‘nil* are singleton (basic) types. For more details about the encoding, also in the presence of attributes and capture variables, see [BCF03].

We can now give the semantics for patterns. Intuitively, a pattern (without capture variable) applied to a value can succeed or fail. Since we want to identify useless sub-patterns, we will directly introduce an instrumented semantics which keeps track of sub-patterns that have indeed be used. Given a value v and a pattern p, the result of matching v against p is a pair $v/p = (\varepsilon, I)$ where $\varepsilon = 0$ denotes failure and $\varepsilon = 1$ denotes success, and I collects all the used ι marks. The definition is given by the following equations:

$$\begin{array}{lll}
c/b & = (1,\varnothing) & \text{if } (c:b) \\
c/b & = (0,\varnothing) & \text{if } \neg(c:b) \\
(v_1,v_2)/b & = (0,\varnothing) & \\
c/(p_1,p_2) & = (0,\varnothing) & \\
(v_1,v_2)/(p_1,p_2) & = (\varepsilon,I_1 \cup I_2) & \text{if } v_1/p_1 = (1,I_1), v_2/p_2 = (\varepsilon,I_2) \\
(v_1,v_2)/(p_1,p_2) & = (0,I_1) & \text{if } v_1/p_1 = (0,I_1) \\
v/(p_1|p_2) & = (1,I_1) & \text{if } v/p_1 = (1,I_1) \\
v/(p_1|p_2) & = (\varepsilon,I_1 \cup I_2) & \text{if } v/p_1 = (0,I_1), v/p_2 = (\varepsilon,I_2) \\
v/(p_1\&p_2) & = (0,I_1) & \text{if } v/p_1 = (0,I_1) \\
v/(p_1\&p_2) & = (\varepsilon,I_1 \cup I_2) & \text{if } v/p_1 = (1,I_1), v/p_2 = (\varepsilon_2,I_2) \\
v/\neg p & = (1-\varepsilon,I) & \text{if } v/p = (\varepsilon,I) \\
v/\mathbf{0} & = (0,\varnothing) & \\
v/\mathbf{1} & = (1,\varnothing) & \\
v/p^{\iota} & = (0,I) & \text{if } v/p = (0,I) \\
v/p^{\iota} & = (1,I \cup \{\iota\}) & \text{if } v/p = (1,I)
\end{array}$$

There are no overlapping cases in this definition, and it is well-founded. Indeed, the values in the right-hand side are smaller than or equal to the value in the left-hand side; when they are equal (which happens for the patterns $p_1|p_2$, $p_1\&p_2$ and p^{ι}), the size of the patterns get strictly smaller, where the size of a pattern is defined by considering pair patterns as leaves (the size is finite because of the contractivity condition).

This instrumented semantics for pattern matching captures marks of sub-patterns which yield a successful match. A sequential traversal order has been chosen: the left sub-pattern in (p_1,p_2), $p_1\&p_2$, $p_1|p_2$ is first considered, and the right sub-pattern is considered only when needed. For the alternation $p_1|p_2$, this corresponds to a natural naive implementation of a first-match policy; for (p_1,p_2) and $p_1\&p_2$, this choice is arbitrary. In all cases, this sequential traversal order is just a way to formalize what are the used sub-patterns - and thus where to raise warnings for unused sub-patterns - and does not give any constraint on the actual run-time implementation of pattern matching.

Since patterns do not have capture variable in this presentation, they can be identified with types. We use the meta-variable t to range over types. The semantics of a type t is the set of values defined as:

$$[\![t]\!] = \{v \mid v/t = (1,I)\}$$

Note that the set of marks I is discarded in this definition. This semantics for types induces a natural equivalence relation: $t_1 \simeq t_2 \iff [\![t_1]\!] = [\![t_2]\!]$. From now on, we will identify types modulo this equivalence. Efficient algorithms have been developed to check inclusion between types; they obviously provide an effective and efficient way to check equivalence as well.

The pattern matching operation is intended to be used as a basic block in a programming language (such as $\mathbb{C}$Duce). The type system for the language provides a static type for the argument of the pattern matching, which is an upper bound for the set of values that can actually flow to the pattern. The question we are interested in is to determine whether some part of the pattern is left unused for any value in this type.

Definition 3. *Let t be a type and p a pattern. The set of used marks when matching t against p is defined as:*

$$I(t,p) = \bigcup_{v\in[\![t]\!],(\varepsilon,I)=v/p} I$$

In words, a marked subpattern of p is used with respect to t if there exists a value v of t for which the marked subpattern is used when matching v against p.

We will now give an algorithm to compute this set $I(t,p)$. First, we define a rewriting relation $\rightsquigarrow$ over type/pattern pairs:

$$\begin{array}{l}
(t,(p_1,p_2)) \rightsquigarrow (\pi_1(t),p_1)\\
(t,(p_1,p_2)) \rightsquigarrow (\pi_2(t\wedge(p_1,\mathbf{1})),p_2)\\
(t,p_1|p_2) \rightsquigarrow (t,p_1)\\
(t,p_1|p_2) \rightsquigarrow (t\wedge\neg p_1,p_2)\\
(t,p_1\&p_2) \rightsquigarrow (t,p_1)\\
(t,p_1\&p_2) \rightsquigarrow (t\wedge p_1,p_2)\\
(t,p^{\iota}) \rightsquigarrow (t,p)\\
(t,\neg p) \rightsquigarrow (t,p)
\end{array}$$

The type operators $\pi_i()$ are defined by the equation: $[\![\pi_i(t)]\!] = \{v_i \mid (v_1,v_2)\in[\![t]\!]\}$. It has been shown in previous work [Fri04] how to compute these operators effectively. The theory developed in this work also shows that, starting from a pair (t,p), the set of pairs (t',p') which are reachable under the reflexive and transitive closure of $\rightsquigarrow$ is finite. This comes from the regularity of types and patterns. This set is thus effectively computable. If we collect all the marks ι such that (t',p'^{ι}) is in this set, and such that some value in t' makes the pattern p' succeed, we obtain exactly the set $I(t,p)$.

Theorem 1. *Let t be a type and p a pattern. Then:*

$$I(t,p) = \{\iota \mid (t,p) \overset{\star}{\rightsquigarrow} (t',p'^{\iota}), t'\wedge p' \not\simeq \mathbf{0}\}$$

4 Discussion

4.1 Characteristics of the Analysis

In the previous section we defined the set $I(t,p)$ of all the pattern marks that are used when matching the pattern p against values in t. We also showed that it is possible to compute this set by saturating the pair (t,p) with a rewriting that is assured to terminate by the regularity of patterns. Actually, the saturated set can be computed quite efficiently, by using the very same algorithms implemented in the ℂDuce type checker.

The computation of this set allows us to detect *all* the unused subparts of a pattern. Indeed if we mark all the occurrences of p, then a mark ι of p is not in $I(t,p)$ if and only if for all values v of type t the sub-pattern marked by ι is not used when matching v against p. In other words, there is no value in t for which this sub-pattern is useful.

The "if and only if" states that our analysis is exact: we cannot refine it further. Of course, as usual, it is just "locally" exact since its global precision depends on the precision of the host type system in inferring the t at issue. For instance, consider the expression:

match e with p -> e'

to check whether p is correct the type system will mark all the occurrences of p, deduce the type t of e, and check whether all the marks of p are in $I(t,p)$. Thus the precision of the deduction of the correctness of p depends on the precision of the type system in inferring t: a more precise inference for the type of e might detect more errors in p, so the analysis is not globally complete, although globally sound (a pattern detected as wrong is indeed wrong).

Local soundness and completeness were not easy to obtain. Our first attempt to define correctness of sub-pattern was based on `Empty` substitutions. According to that attempt a sub-pattern of a pattern p was considered wrong with respect to an input type t if for every value v of t there was no difference between matching v against p, or matching v against the same p in which the sub-pattern is replaced by $\mathbf{0}$ (i.e. the `Empty` type). Now, such a characterization captures all the examples we gave in Section 2 but it is not sound with respect to all possible wrong patterns since it signals as wrong some sub-patterns that should not be considered as such. The most trivial example is the pattern `Int|3`: for an input type 3 it signals the pattern `Int` as wrong. But that is a trivial case in which the right hand side of `|` is contained in the left hand-side. A subtler example where the two branches of the "`|`" pattern are independent (no inclusion) is `(Even,_)|(Int,Bool)`. With input type `(Even,Bool)|(Odd,Int)` the subpattern `(Even,_)` is considered wrong according to `Empty` substitution characterization, while the analysis of Section 3 correctly fingers as wrong the sub-pattern `Bool`.

It is important to stress that our definition of "used mark" hardcodes the intuition we have about errors. We already stressed in the previous section that our definitions reflect a sequential transversal order for the tree. So for instance if the pattern `Int|Int` is used, our analysis fingers as wrong the rightmost occurrence of `Int`; an analysis signalling the leftmost occurrence as wrong would be equally correct but, in our opinion, less intuitive. The same left to right analysis is applied to intersections and pairs, as well.

Also we wondered whether to consider as wrong a pattern such as `Int & Int`. Indeed, in our left to right perspective the rightmost `Int` is useless. Note however that here it is not the matter of being not used, but of being redundant (for instance, the `Empty` substitution argument does not apply). Thus it was clear to us that redundancy *must not* considered as an error since it would go against a common programming practise: programmers prefer to use redundant patterns so as to reuse previous type definitions and make the code simpler and more readable rather than to write the exact pattern that ensures the absence of any redundancy: we must not force her/him to use this second option.

4.2 Extension to $\mathbb{C}$Duce and Other Languages

The analysis developed in Section 3 applies directly to the cited languages based on regular expression patterns. Xtatic and recent versions of XDuce, however, require a slight modification to the definition of used sub-patterns since they use a non-deterministic semantics for the `|` pattern. This is very simple as it suffices to replace the two cases for $v/(p_1|p_2)$ by

$$v/(p_1|p_2) = (\varepsilon_1 \,||\, \varepsilon_2, I_1 \cup I_2) \qquad\qquad \text{if } v/p_i = (\varepsilon_i, I_i)$$

where $||$ denotes the logical or. The algorithm to compute used sub-patterns is simple to adapt as well. The rewriting rules for the `|` pattern are changed to:

$$(t, p_1|p_2) \rightsquigarrow (t, p_1)$$
$$(t, p_1|p_2) \rightsquigarrow (t, p_2)$$

For what concerns the capture variables, we have to modify the definition of matching, since v/p must not only return a zero/one result but, in case of success, it also must return a substitution for the variables of the pattern. However, the analysis of Section 3 needs no change, since capture variables technically behaves the same as intersections with the type `Any`, as such they do not affect the analysis.

What it really remains to do in order to embed our analysis in the various languages at issue is to extend its definition to the other patterns present in these languages (for instance, in ℂDuce there also is a pattern for records), and to customize the typing rules of the languages so that they use the analysis. Let us discuss this last point for ℂDuce. We already hinted at how the typing rule for `match` expressions must be modified for taking into account the analysis. Formally this corresponds to having the following typing rule:

$$\frac{\begin{array}{l}(\text{for } t_i \equiv t \setminus \langle\!| p_1 |\!\rangle \setminus \dots \setminus \langle\!| p_{i-1} |\!\rangle) \\ \quad t \leq \langle\!| p_1 |\!\rangle \mid \dots \mid \langle\!| p_n |\!\rangle \qquad I(t_1, p_i) = (\varepsilon, I_i) \qquad \Delta'_i = marks(p_i) \setminus I_i \\ \qquad\qquad \Gamma \vdash e : t \rightsquigarrow \Delta \qquad\qquad \Gamma, (t_i/p_i) \vdash e_i : s_i \rightsquigarrow \Delta_i\end{array}}{\Gamma \vdash \text{match } e \text{ with } p_1 \rightarrow e_1 \mid \dots \mid p_n \rightarrow e_n : \bigcup_{\{i | t_i \not\simeq \text{Empty}\}} s_i \rightsquigarrow \bigcup_{i=1\dots n} \Delta_i \cup \Delta'_i \cup \Delta}$$

$\langle\!| p_i |\!\rangle$ denotes the exact type of all values that successfully match p_i (namely $\langle\!| p |\!\rangle = \{v \mid v/p \text{ succeeds}\}$), while $\Gamma \vdash e : t \rightsquigarrow \Delta$ means that, in the type environment Γ, e has type t and the labels in Δ denote unused sub-patterns in e; hence Δ is the error set computed by the type analysis (an expression is correct if the inferred error set is empty). The condition $t \leq \langle\!| p_1 |\!\rangle \mid \dots \mid \langle\!| p_n |\!\rangle$ ensures that patterns are exhaustive with respect to all possible values e may produce. This ensures that well-typed terms never get stuck at run-time (at least one branch matches). (t_i/p_i) denotes the set of type assignments of p_i variables, computed by matching the pattern against the type t_i (see [FCB02]). Each t_i is computed by taking into account the first-match policy, so the e_i is typed in an environment in which each p_i is matched over values that cannot be matched by previous branches. Incorrect sub-patterns in p_i's are computed by subtracting from all marks of each p_i, denoted by $marks(p_i)$, the set of used patterns $I(t_i, p_i)$.

Error mining for iterators is not so straightforward, due to typing based on case analysis over the argument type. For example, if e is proved to have type `[`$S \mid U$`]`, then the `with` part of

```
transform e with p -> e'
```

is typed twice, once under the assumption that the argument has type S and once under the assumption that the type for the argument is U, thus inferring two types T_S and T_U, together with two errors sets Δ_S and Δ_U. The final inferred type is `[`$T_S \mid T_U$`]`, while the final errors set is $\Delta_S \cap \Delta_U$. This is because a sub-pattern in p is incorrect (unused) if it is so for both alternatives S and U (or, equivalently, it is correct if it is correct (used) with respect to at least one alternative among S and U). This technique was introduced and proved to be correct in [Col04, CGMS04]. Thus the complete formalization of error mining rules for iterators such as `transform`, follows those established for the XQuery iterator `for` in the cited papers, relying on the technique of Section 3 to infer incorrect fragments of patterns.

A similar technique must be used for overloaded functions: in ℂDuce an overloaded function is a function whose type is an intersection of arrows and the body of the function is typed once for each type in the intersection; of course are wrong only those occurrences of patterns that result unused in all these type deductions; once more an intersection applies. To implement this just a slight modification of the original typing rule is required, since we only need to add error sets to judgements and opportunely combine them in a way that strictly resembles error mining for iterators:

$$\frac{\Gamma, (x : t_i), (f : \bigwedge_{i=1..n} t_i \to s_i) \vdash e : s_i \rightsquigarrow \Delta_i \qquad i = 1..n}{\Gamma \vdash \mathsf{fun} f(t_1 \to s_1; \ldots; t_n \to s_n)(x) = e \,:\, (\bigwedge_{i=1..n} t_i \to s_i) \rightsquigarrow \bigcap_{i=1..n} \Delta_i}$$

The extension of other rules is even simpler, and omitted for space reasons.

It is worth observing that the presented error mining technique preserves the typing discipline in the hosting language, since error-mining depends on type-inference but not viceversa. In other words, the technique we have described is not intrusive and can be seen as an add-on of the hosting type system. However, in order to make error-mining more precise, that is to increase the number of errors detected at static time, one may consider to change the type discipline of the language. This may be needed when the type system infers a not-empty sequence type for expressions that instead always evaluate to the empty sequence. This can raise from an interaction between `/` and `transform`: for an example see [Col04] where the problem has been solved for XQuery. At this stage, we did not investigate this problem in the context of languages based on regular expression patterns, and we postpone this to future work.

Acknowledgments. This work was partially supported by the RNTL project "GraphDuce" and by the ACI project "Transformation Languages for XML: Logics and Applications".

References

[AC93] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4), September 1993.

[BCF03] V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-friendly general purpose language. In *ICFP '03, 8th ACM International Conference on Functional Programming*, pages 51–63, Uppsala, Sweden, 2003. ACM Press.

[BCM05] V. Benzaken, G. Castagna, and C. Miachon. A full pattern-based paradigm for XML query processing. In *PADL 05, 7th International Symposium on Practical Aspects of Declarative Languages*, number 3350 in LNCS, pages 235–252. Springer, January 2005.

[BFS04] Niklas Broberg, Andreas Farre, and Josef Svenningsson. Regular expression patterns. In *ICFP '04: Proceedings of the ninth ACM SIGPLAN international conference on Functional programming*, pages 67–78, New York, NY, USA, 2004. ACM Press.

[BMS05] Gavin Bierman, Erik Meijer, and Wolfram Schulte. The essence of data access in Cω. In *ECOOP 2005*, LNCS, 2005. To appear.

[CGMS04] Dario Colazzo, Giorgio Ghelli, Paolo Manghi, and Carlo Sartiani. Types for Path Correctness for XML Queries. In *Proceedings of the ACM International Conference on Functional Programming (ICFP), Snowbird, Utah, USA*, 2004.

[Col04] Dario Colazzo. *Path Correctness for XML Queries: Characterization and Static Type Checking*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2004.

[DFF$^+$05] Denise Draper, Peter Fankhauser, Mary Fernandez, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics. Technical report, World Wide Web Consortium, February 2005. W3C Working Draft.

[FCB02] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic Subtyping. In *Proceedings, Seventeenth Annual IEEE Symposium on Logic in Computer Science*, pages 137–146. IEEE Computer Society Press, 2002.

[Fri04] Alain Frisch. *Théorie, conception et réalisation d'un langage de programmation fonctionnel adapté à XML.* PhD thesis, Université Paris 7, December 2004.

[GP03] Vladimir Gapeyev and Benjamin C. Pierce. Regular object types. In *European Conference on Object-Oriented Programming (ECOOP), Darmstadt, Germany*, 2003. A preliminary version was presented at FOOL '03.

[Hos00] Haruo Hosoya. *Regular Expression Types for XML.* PhD thesis, The University of Tokyo, Japan, December 2000.

[HP00] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. In *Proceedings of Third International Workshop on the Web and Databases (WebDB2000)*, 2000.

[HP01] Haruo Hosoya and Benjamin C. Pierce. Regular expression pattern matching for XML. In *The 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2001.

[LS04] K. Zhuo Ming Lu and M. Sulzmann. An implementation of subtyping among regular expression types. In *Proc. of APLAS'04*, volume 3302 of *LNCS*, pages 57–73. Springer-Verlag, 2004.

[OAC$^+$04] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger. An overview of the Scala programming language. Technical Report IC/2004/64, École Polytechnique Fédérale de Lausanne, 2004. Latest version at http://scala.epfl.ch.

Reconstructing an Alternate Periodical Binary Matrix from Its Orthogonal Projections

Marie-Christine Costa, Fethi Jarray, and Christophe Picouleau

Laboratoire CEDRIC, 292 rue Saint-Martin, 75003 Paris, France
{costa, fethi.jarray, chp}@cnam.fr

Abstract. This paper deals with the reconstruction of an alternate periodical binary matrix from its orthogonal projections. For a fixed vector (p, q), a binary matrix A is alternate periodical when $A_{i,j} + A_{i+p,j+q} = 1$. For vectors $(p = 1, q = 1)$, $(p, 0)$ and $(0, q)$ we propose polynomial time algorithms to reconstruct an alternate periodical binary matrix from both its vertical and horizontal projections if such a matrix exists.

Keywords: binary matrix, alternate periodicity, discrete tomography, polynomial time algorithm.

1 Introduction

Discrete tomography deals with the reconstruction of discrete homogenous objects regarded as binary matrices from their projections. The problem of reconstructing a $m \times n$ binary matrix from its orthogonal projections H and V is the following : given $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$ two nonnegative integer vectors find a binary matrix A such that the number of ones in every row i (resp. column j) equals h_i (resp. v_j). The reader can find results about this basic problem in the seminal work of Ryser [8] and in the paper of Brualdi [2]. The reader is also referred to the book of Hermann and Kuba [7] for an overview on discrete tomography.

In many applications such as image processing and electron microscopy, the only orthogonal projections are not sufficient to uniquely determine matrices. Fortunately, objects that occur in practical applications usually exhibit certain properties. Hence we seek to reconstruct binary matrices under additional constraints like connectivity or convexity for instance. Some special problems such as the reconstruction of polyominoes (connected sets) are NP-hard (see [1] and [9]). Other cases such as the reconstruction of hv-convex polyominoes are polynomial (see [1] and [4]).

The problem of reconstructing periodical objects arises especially on the reconstruction of crystalline structures. Del Lungo and al. [5] studied the (p, q)-periodical matrices reconstructing problem. A binary matrix A is said to be (p, q)-periodical if $A_{i,j} = A_{i+p,j+q}$ for $1 \leq i \leq m - p$ and $1 \leq j \leq n - q$. From the orthogonal projections and the (p, q)-periodicity, they define the "boxes" of the matrix and they establish some properties of these boxes. In the case of

$(1,1)$-periodical matrix, these boxes are reduced to single cells and it is possible to determine the value of some boxes and propagate these values into the matrix using the $(1,1)$-periodicity. Then the cells with an undermined value constitute "cycles" and the assignment of the values to these cycles is performed using the algorithm of Ryser [8] that reconstructs a binary matrix from its orthogonal projections. Hence the case $(1,1)$-periodicity is solved in polynomial time. In the sequel of their paper, the authors solve in polynomial time a particular case of the $(1,q)$-periodicity problem.

In this paper, we will focus on the binary alternate periodical matrices reconstruction problem because on the crystalline structures the atoms may be disposed in a periodical or an alternate periodical manner. Given two nonnegative integers p and q, a binary matrix A is a (p,q) alternate periodical matrix if $A_{i,j} + A_{i+p,j+q} = 1$ for $1 \leq i \leq m-p$ and $1 \leq j \leq n-q$.

The paper is organized as follows. Section 2 introduces the problem and develops general properties of alternate periodical matrices. In section 3, we solve the $(1,1)$-alternate periodical matrices reconstruction problem. Section 4 is devoted to the case of $(p,0)$ or $(0,q)$-alternate periodical matrices.

2 (p,q)-Alternate Periodical Matrix

Let A be a $m \times n$ binary matrix. The horizontal projection of A is the vector $H = (h_1, \ldots, h_m)$ such that $h_i = \sum_{j=1}^{n} A_{ij}$ is the sum of the elements lying on row i. The vertical projection of A is defined analogously as the vector $V = (v_1, \ldots, v_n)$ where $v_j = \sum_{i=1}^{m} A_{ij}$ is the sum of the elements on column j. Both projections H and V constitute the orthogonal projections of A.

Definition 1. *A is a (p,q)-alternate periodical matrix if $A_{i,j} + A_{i+p,j+q} = 1$ for $i = 1, \ldots, m-p$ and $j = 1, \ldots, n-q$, i.e. the entries (i,j) and $(i+p, j+q)$ of A have opposite values.*

Given vectors $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$, our problem consists in finding a (p,q)-alternate periodical matrix with H and V as orthogonal projections.

Now, we give a property of the orthogonal projections H and V of a (p,q)-alternate periodical matrix.

Proposition 1. *i) $|h_i + h_{i+p} - n| \leq q,\ i = 1, \ldots, m-p$;*
ii) $|v_j + v_{j+q} - m| \leq p,\ j = 1, \ldots, n-q$.

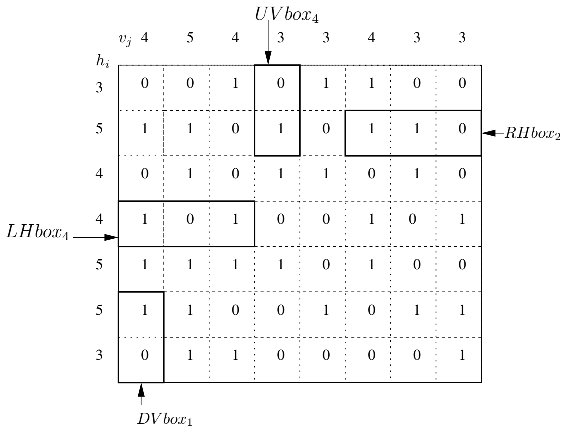
Proof. We have $h_i = \sum_{j=1}^{n} A_{i,j} = \sum_{j=1}^{n-q} A_{i,j} + \sum_{j=n-q+1}^{n} A_{i,j}$. So, taking into account the (p,q)-alternate periodicity, we get :
$h_i = \sum_{j=1+q}^{n}(1 - A_{i+p,j}) + \sum_{j=n-q+1}^{n} A_{i,j} = n - h_{i+p} - \sum_{j=1}^{q}(1 - A_{i+p,j}) + \sum_{j=n-q+1}^{n} A_{i,j}$ for $1 \leq i \leq m-p$. Hence $|h_i + h_{i+p} - n| = |\sum_{j=n-q+1}^{n} A_{i,j} - \sum_{j=1}^{q}(1 - A_{i+p,j})| \leq q$.

Proceeding in the same way for the vertical projection we obtain ii). □

It is convenient to introduce the definitions of boxes (see Figure 1):

Definition 2. *$RHbox_i$ is the set of entries $\{(i, n-q+1), \ldots, (i, n)\}$; its weight is $w_{RH_i} = \Sigma_{j=n-q+1}^{n} A_{i,j}$*
$LHbox_i$ is the set of entries $\{(i, 1), \ldots, (i, q)\}$; its weight is $w_{LH_i} = \Sigma_{j=1}^{q} A_{i,j}$
$UVbox_j$ is the set of entries $\{(1, j), \ldots, (p, j)\}$; its weight is $w_{UV_j} = \Sigma_{i=1}^{p} A_{i,j}$
$DVbox_j$ is the set of entries $\{(m-p+1, j), \ldots, (m, j)\}$; its weight is $w_{DV_j} = \Sigma_{i=m-p+1}^{m} A_{i,j}$

Remark 1. For an alternate periodical matrix A we have $w_{RH_i} + w_{LH_{i+p}} = h_i + h_{i+p} - n + q$ and $w_{DV_j} + w_{UV_{j+q}} = v_j + v_{j+q} - m + p$.



$$\begin{array}{c|cccccccc}
h_i \backslash v_j & 4 & 5 & 4 & 3 & 3 & 4 & 3 & 3 \\
\hline
3 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
5 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
4 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
5 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
5 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
3 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Fig. 1. A (2,3)-alternate periodical matrix and its boxes

3 (1, 1)-Alternate Periodical Matrix

In this section, we will study the case where $(p, q) = (1, 1)$.

Definition 3. *Let A be a binary matrix, $A_{i,j}$ is a border entry if $i = 1$ or $i = m$ or $j = 1$ or $j = n$. The four entries $A_{1,1}, A_{1,n}, A_{m,1}$ and $A_{m,n}$ are the corners.*

Here each box is reduced to a singleton, thus its weight is either one or zero. So the previous remark can be written as:
For $1 \leq i \leq m-1$:

- if $h_i + h_{i+1} = n + 1$ then $A_{i,n} = A_{i+1,1} = 1$,
- if $h_i + h_{i+1} = n - 1$ then $A_{i,n} = A_{i+1,1} = 0$,
- if $h_i + h_{i+1} = n$ then $A_{i,n} + A_{i+1,1} = 1$.

For $1 \leq j \leq n-1$:

- if $v_j + v_{j+1} = m + 1$ then $A_{m,j} = A_{1,j+1} = 1$,
- if $v_j + v_{j+1} = m - 1$ then $A_{m,j} = A_{1,j+1} = 0$,
- if $v_j + v_{j+1} = m$ then $A_{m,j} + A_{1,j+1} = 1$.

Thus any border entry $A_{i,j}$ that is not a corner is "matched" with another border entry and the corner $A_{1,n}$ (resp. $A_{m,1}$) is matched with both $A_{2,1}$ and $A_{m,n-1}$ (resp. $A_{1,2}$ and $A_{m-1,n}$). Moreover, for any couple, the knowledge of the value of one entry is sufficient to determine the value of the second one.

As in [5], we introduce the function mod$[n]$ which slightly differs from the well known mod$n$ function.
$\text{mod}[n] : \mathbb{N} \rightarrow \mathbb{N}$
$$x \bmod [n] = \begin{cases} x \bmod n & \text{if x mod n} \neq 0 \\ n & \text{else} \end{cases}$$

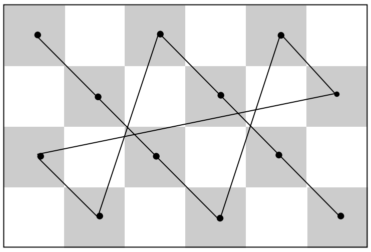
Definition 4. *Two entries $A_{i,j}$ and $A_{i',j'}$ are two neighbors if $i' = i+1 \bmod [m]$ and $j' = j+1 \bmod [n]$.*

This definition of neighborhood induces a graph $G(X, E)$ where the nodes of X correspond to the entries of A and the edge set E represents the neighborhood relation : $[A_{i,j}, A_{i',j'}] \in E$ iff $A_{i,j}$ and $A_{i',j'}$ are two neighbors. Since each entry has two neighbors exactly, G is a collection of cycles. The nodes of a cycle are $\{A_{(i+k) \bmod [m],(j+k) \bmod [n]}, k \in \mathbb{Z}\}$. By convention, we say that the cycle c_j *begins* on its upper left most entry $A_{1,j}$ (see Figure 2). Note that from the $(1,1)$-alternate periodicity of A and the remark above, the value of every entry of a cycle is determined by the knowledge of the value of one of its entries.

Proposition 2. *The length of the cycles is $LCM(m, n)$.*

Proof. We have $(i+k) \bmod [m] = i$ and $(j+k) \bmod [n] = j$ only if k is a multiple of both m and n. So $i = (i + k)\, mod\, m$ and $j = (j + k) \bmod [n]$ only if k is a multiple of $LCM(m, n)$ and the length of every cycle is $LCM(m, n)$. □

In matrix A, a set of positions/entries c is said to be an $(1,1)$-alternate cycle if $A_{i,j} + A_{(i+1) \bmod [m],(j+1) \bmod [n]} = 1$ for each $A_{i,j}$ of c except for the entries $A_{1,1}$ and $A_{m,n}$. Despite the cycle containing the entries $A_{1,1}$ and $A_{m,n}$ is not properly a cycle according to the terminology of graph theory (from our definition $A_{1,1}$ and $A_{m,n}$ and are not adjacent), we make no distinction between this $(1,1)$-alternate cycle and the others.



▨ Cycle c_1 beginning on $A_{1,1}$

Fig. 2. The cycles of an $(1,1)$-alternate periodical matrix

3.1 Reconstruction of an $(1,1)$-Alternate Periodical Matrix

We will design a polynomial time algorithm that reconstruct an $(1,1)$-alternate periodical matrix from vectors H and V, if such a matrix exists.

In a first step, using the values of $h_i + h_{i+1}, 1 \leq i \leq m-1$, and $v_j + v_{j+1}, 1 \leq j \leq n-1$, we determine the border entries for which we can assign the value 0 or 1. Then for every entry of a cycle containing such an entry we can also assign the value 0 or 1. At this step, we verify the consistency, i.e. there is no entry taking both values 0 and 1. If an inconsistency emerges then there is no $(1,1)$-alternate periodical matrix with projections H and V. If the values of all the entries of A are determined then the problem is solved.

In a second step, we suppose that there are some entries of A for which the value is not fixed. From the matching between the border entries, all these entries are covered by some $(1,1)$-alternate cycles.

We denote by x_j the value of the entry $A_{1,j}$ where the $(1,1)$-alternate cycle c_j begins. For an $(1,1)$-alternate cycle beginning on $A_{1,j}$, we can determine the number of 1's per row and column of A according to the value of x_j (see Table 1). The values a, a', b and b' may be considered as the projections of c_j.

Table 1. Projections of c_j and values of a, a', b and b'

	$x_j = 1$	$x_j = 0$
Odd row	a	a'
Even row	a'	a
Column $j' : j' = j \bmod [2]$	b	b'
Column $j' : j' = 1 + j \bmod [2]$	b'	b

	m even	m odd	n even	n odd
a	$\frac{LCM(m,n)}{m}$	$\lceil \frac{LCM(m,n)}{2m} \rceil$	-	-
a'	0	$\lfloor \frac{LCM(m,n)}{2m} \rfloor$	-	-
b	-	-	$\frac{LCM(m,n)}{n}$	$\lceil \frac{LCM(m,n)}{2n} \rceil$
b'	-	-	0	$\lfloor \frac{LCM(m,n)}{2n} \rfloor$

We denote by h'_i (resp. v'_j) the number of 1's that remains to be placed on row $i,\ i = 1, \ldots, m$, (resp. column $j,\ j = 1, \ldots, n$) after the first step. The $(1,1)$-alternate cycles with undetermined values should have H' and V' as projections.

Since the orthogonal projections of a cycle c_j depend only on x_j and on the parity of j, we introduce the variables $x = \sum_{j,\, even} x_j$ and $y = \sum_{j,\, odd} x_j$. I_e and I_o count the number of cycles starting on even columns and odd columns respectively and which should be placed after the first step. Then we have to consider the following linear system:

$$\mathcal{S} \begin{cases} ax + ay + a'(I_e - x) + a'(I_o - y) = h'_1 & (1) \\ a'x + a'y + a(I_e - x) + a(I_o - y) = h'_2 & (2) \\ b'x + by + b(I_e - x) + b'(I_o - y) = v'_1 & (3) \\ bx + b'y + b'(I_e - x) + b(I_o - y) = v'_2 & (4) \\ x \leq I_e, y \leq I_o \end{cases}$$

The constraint (1) ensures the satisfaction of the horizontal projections of the odd rows and the constraint (2) ensures those of the even rows. The constraints (3) and (4) ensure the satisfaction of the projections of the columns.

We deduce from $\mathcal{S}$ the following necessary conditions for an $(1,1)$-alternate periodical matrix to exist :

$$h'_1 + h'_2 = (a+a')I_e + (a+a')I_o = \frac{LCM(m,n)}{m}(I_e + I_o)$$

$$v'_1 + v'_2 = (b+b')I_e + (b+b')I_o = \frac{LCM(m,n)}{n}(I_e + I_o).$$

Moreover, an $(1,1)$-alternate cycle has $\frac{LCM(m,n)}{m}$ 1's on the first two rows and $\frac{LCM(m,n)}{n}$ 1's on the the first two columns. We suppose that these conditions hold. Then $\mathcal{S}$ can be simplified by eliminating (2) and (4):

$$\mathcal{S}' \quad \begin{cases} (a-a')(x+y) = h'_1 - a'I_e - a'I_o & (1') \\ (b'-b)(x-y) = v'_1 - bI_e - b'I_o & (2') \\ x \le I_e, y \le I_o \end{cases}$$

Three cases can be distinguished according to the values of a, a', b and b':

- if $a \neq a'$ and $b \neq b'$ then $\mathcal{S}'$ admits the unique solution

$$x = \frac{(h'_1 - a'I_e - a'I_o)(b-b') - (v'_1 - bI_e - b'I_o)(a-a')}{2(a-a')(b-b')}$$

$$y = \frac{(v'_1 - bI_e - b'I_o)(a-a') + (h'_1 - a'I_e - a'I_o)(b-b')}{2(a-a')(b-b')}$$

  whenever x and y are integer and $x \le I_e, y \le I_o$.
- if $a = a'$ then (see Table 1) m is odd and $a = a' = \lceil \frac{LCM(m,n)}{2m} \rceil = \lfloor \frac{LCM(m,n)}{2m} \rfloor$, and since $LCM(m,n)$ is even, n is even and then $b \neq b'$ and $b' = 0$. Equation (1') becomes $h'_1 = a(I_e + I_o)$ which is satisfied by the definitions of I_e, I_o and a. Thus the system $\mathcal{S}'$ is : $y - x = \frac{v_1}{b} - I_e$ with $x \le I_e$ and $y \le I_o$ and has solutions:

$$\{(x = I_e - k, y = \frac{v'_1}{b} - k) : k \le min(I_e, \frac{v'_1}{b}), k \in \mathbb{N}\} \text{ if } \mathrm{I_o} > \frac{\mathrm{v'_1}}{\mathrm{b}}$$

$$\{(x = I_e + I_o - \frac{v'_1}{b} - k, y = I_o - k) : k \le min(I_o, I_e + I_o - \frac{v'_1}{b}), k \in \mathbb{N}\} \text{ else.}$$

- if $b = b'$ then by the same way, $\mathcal{S}'$ has the following solutions :

$$\{(x = \frac{h'_1}{a} - k, y = k) : k \le min(I_o, \frac{h'_1}{a}), k \in \mathbb{N}\} \text{ if } \frac{\mathrm{h'_1}}{\mathrm{a}} \le \mathrm{I_e}$$

$$\{(x = I_e - k, y = \frac{h'_1}{a} - I_e + k) : k \le min(I_e, I_e + I_o - \frac{h'_1}{a}), k \in \mathbb{N}\} \text{ else.}$$

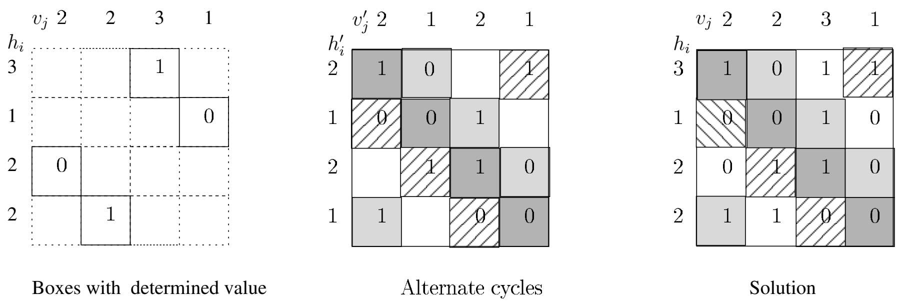


Fig. 3. Reconstruction of an $(1, 1)$-alternate periodical matrix

Finally, when $\mathcal{S}'$ has a solution, x_j's are obtained by setting the value 1 to x variables x_j such that j is even and to y variables x_j such that j is odd. The other x_j's are set to 0.

Example 1. We will illustrate our algorithm by an example: the goal is to reconstruct an $(1, 1)$-binary alternate periodical matrix with projections $H = (3, 1, 2, 2)$ and $V = (2, 2, 3, 1)$ (see Figure 3). The entries (boxes) $A_{2,4}$ and $A_{3,1}$ receive the value 0 because $h_2 + h_3 - n + q = 0$. Propagating along the direction $(1, 1)$, the entries $A_{1,3}$ and $A_{4,2}$ are set to 1. Thus we obtain $H' = (2, 1, 2, 1)$ and $V' = (2, 1, 2, 1)$ and we are concerned with three $(1, 1)$-alternate cycles c_1, c_2 and c_4. The length of these c_i's is $LCM(4, 4) = 4$. We have $I_e = 2$, $I_o = 1$, $a = b = 1$ and $a' = b' = 0$.

A solution to $\mathcal{S}'$ is $(x, y) = (1, 1)$ (in our example $\mathcal{S}'$ has an unique solution) and we have $x_1 = 1$ and $x_2 + x_4 = 1$. Choosing $x_2 = 0$ and $x_4 = 1$ we have that c_2 begins with an entry with the value 0 and c_1 and c_4 begin with an entry with value 1 (see Figure 3). One can remark that for $x_2 = 1$ and $x_4 = 0$, we obtain a second $(1, 1)$-alternate periodical matrix with the same orthogonal projections.

Proposition 3. *The time complexity of the reconstruction of an* $(1, 1)$*-alternate periodical binary matrix is* $O(mn)$.

Proof. It takes at most $O(m + n)$ operations to determine the matching of the border entries and the weight of any box. The determination of the c_i's and the propagation along the $(1, 1)$ direction take $O(mn)$. The system $\mathcal{S}'$ is solved in constant time. Thus, the total computational complexity of the proposed algorithm is $O(mn)$. □

4 $(p, 0)$-Alternate Periodical Matrix

In this section we consider the case where the matrices are $(p, 0)$-alternate periodical, i.e. the rows have a p-alternate periodicity and the columns don't have any periodicity. Using symmetry the case of $(0, q)$-alternate periodical matrices is solved as well.

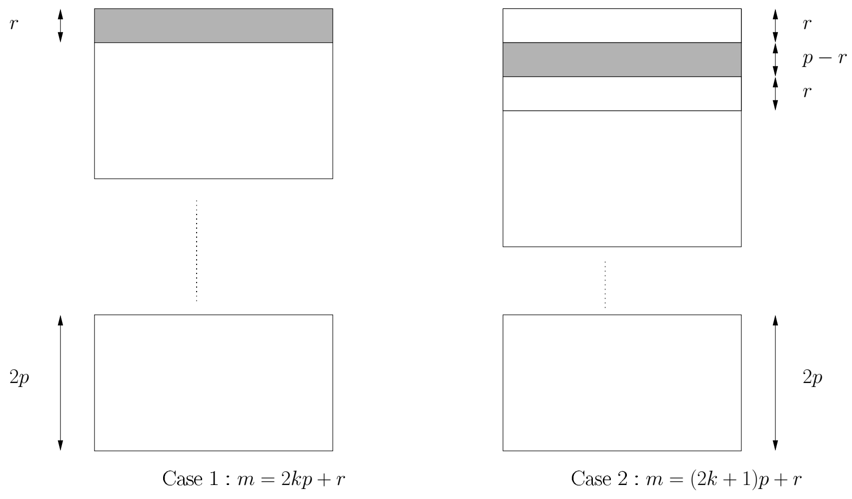


Fig. 4. The $(p, 0)$ case: Partition of rows

Remark 2. It's important to note that due to the $(p, 0)$-alternate periodicity, on each column, the sum of $2p$ consecutive elements is exactly p.

We distiguish two cases according to m and p (see Figure 4) :

4.1 Case 1 $m = 2kp + r, \quad r \leq p$

We make the following partition of the rows : the first block contains the rows $\{1, \ldots, r\}$, the k following blocks have $2p$ rows $\{2pi + j + r, 1 \leq j \leq 2p\}, 0 \leq i \leq k - 1$. We denote by $v'_j,\ j = 1, \ldots, n$, the vertical projection of the first block. By the remark above, we have $v'_j = v_j - kp,\ j = 1, \ldots, n$. This gives us a way to reconstruct a solution :

1. Using the algorithm of Ryser [8], reconstruct a $r \times n$ binary matrix respecting the horizontal projection $h_1, \ldots, h_r$ and the vertical projection $v'_1, \ldots, v'_n$, for the first block
2. For each row $i = r + 1, \ldots, p$, set h_i entries to 1
3. Propagate with $(p, 0)$-alternate periodicity the values of the first p rows.

4.2 Case 2 $m = 2kp + p + r, \quad r \leq p$

We use the following partition of the rows : the first block has rows $\{1, \ldots, r\}$, the second block contains the rows $\{r + 1, \ldots, p\}$, the third block has r rows $\{p + 1, \ldots, p + r\}$; the k following blocks have $2p$ rows $\{(2i + 1)p + j + r, 1 \leq j \leq 2p\}, 0 \leq i \leq k - 1$. We denote by $v'_j,\ j = 1, \ldots, n$, the vertical projection of the second block. We note that there are r elements equal to 1 and r elements equal to 0 per column on the first and third blocks (rows $\{1, \ldots, r\}$ and $\{p+1, \ldots, p+r\}$). So we get $v'_j = v_j - kp - r,\ j = 1, \ldots, n$. The algorithm proceeds according to the following steps :

1. Using the algorithm of Ryser [8], reconstruct a $(p-r) \times n$ binary matrix respecting the horizontal projection $h_{r+1}, \ldots, h_p$ and the vertical projection $v'_1, \ldots, v'_n$, for the second block
2. For each row $i = 1, \ldots, r$, set h_i entries to 1
3. Propagate with $(p, 0)$-alternate periodicity the values of the first p rows.

5 Conclusion

In this paper we have established general properties of (p, q)-alternate periodical matrices and we have proposed polynomial time algorithms to reconstruct such matrices from their orthogonal projections in the cases where $(p, q) = (1, 1)$ and $p = 0$ or $q = 0$. As far as the authors know, this paper is the first study of reconstructing binary matrices under alternate periodicity constraints. Several future research extensions of this research could be considered. Examples of these extensions include the following considerations, for a given (p, q), each line with direction (p, q) is either (p, q)-periodical or (p, q)-alternate periodical.

References

1. Barcucci, E., Del Lungo, A.: Reconstructing convex polyominoes from their horizontal and vertical projections. Theoretical computer science **155(1)** (1996) 321-347.
2. Brualdi, R.A.: Matrices of zeros and ones with fixed row and column sum. Linear algebra and its applications **3** (1980) 159-231.
3. Chrobak, M., Dürr, C.: Reconstructing Polyatomic Structures from X-Rays: NP Completness proof for three Atoms. Theoretical computer science, **259(1)** (2001) 81-98.
4. Chrobak, M., Dürr, C.: Reconstructing hv-convex polyominoes from orthogonal projections. Information Processing Letters, **69** (1999) 283-289.
5. Del Lungo, A., Frosini, A., Nivat, M., Vuillon, L.: *Reconstruction under Periodicity Constraints.* ICALP **1** (2002) 38-56.
6. Kuba, A., Hermann, G.T.: Discrete Tomography: a historical overview. Discrete Tomography: Foundations, Algorithms and Applications. Birkhauser (1999) 3-33.
7. Kuba, A., Hermann, G.T.: Discrete Tomography: Foundations, Algorithms and Applications. Birkhauser (1999).
8. Ryser, H.J.: Combinatorial Properties of Matrices of Zeros and Ones. Canad. J. Math **9** (1957) 371-377.
9. Woeginger, G.J.: The reconstruction of polyominoes from their orthogonal projections. Information Processing Letters **77(5-6)** (2001) 225-229.

Inapproximability Results for the Lateral Gene Transfer Problem*

Bhaskar DasGupta[1], Sergio Ferrarini[2], Uthra Gopalakrishnan[1], and Nisha Raj Paryani[1]

[1] Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053
{dasgupta, ugopalak, nparyani}@cs.uic.edu

[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32 20133, Milano, Italy
sferrarini@gmail.com

Abstract. This paper concerns the *Lateral Gene Transfer Problem*. This minimization problem, defined by Hallet and Lagergren [6], is that of finding the *most parsimonious* lateral gene transfer scenario for a given pair of gene and species trees. Our main results are the following:

(a) We show that it is not possible to approximate the problem in polynomial time within an approximation ratio of $1+\varepsilon$, for some constant $\varepsilon > 0$ unless P=NP. We also provide explicit values of ε for the above claim.
(b) We provide an upper bound on the cost of any 1-active scenario and prove the tightness of this bound.

1 Introduction

A fundamental problem in the field of evolutionary molecular biology is that of inferring information on the evolutionary relationships between taxa from a given set of gene trees (*i.e.*, an evolutionary model for a set of gene families). The underlying assumption is that gene families evolve in the same way as species; therefore a gene tree should determine the species tree. Unfortunately, there are a number of biological events, such as *gene duplications*, *gene losses* and *lateral gene transfers* (also called horizontal gene transfers) (*e.g.*, see [5, 9]) that may occur during evolution and that generate "differences" between a gene and a species tree. For these reasons, a single gene tree is usually not sufficient to reliably build the species trees, but it is necessary to consider a set a gene families to perform the construction. Since the gene trees may contain contradictory information, a natural problem that arises is that of *reconciling* the different gene trees into a single species tree. Such a reconciliation process can be naturally formulated as an optimization problem where the goal is to *minimize* the number

* This research was supported by NSF grants CCR-0296041, CCR-0206795, CCR-0208749 and IIS-0346973.

of biological events necessary to explain the "disagreements" between the gene trees and the species tree.

Several models have been proposed to solve the reconciliation problem. Each of these models is based on the assumption that only a restricted class of genomic events may occur. Here we focus on the so-called *lateral gene transfer model* defined by Hallett and Lagergren [6]. According to this model, all differences between the gene and the species trees are explained in terms of lateral gene transfer events. A lateral gene transfer is an event that causes some portion of the evolution represented by an arc in the gene tree to occur along one arc in the species tree, and the remaining portion of evolution to occur along another arc of the species tree. We say that the lateral transfer occurs between these two arcs of the species tree and involves the arc from the gene tree. Given a gene tree T and a species tree S (which we assume to be correct), an interesting optimization problem is that of identifying a *scenario* that is able to explain the differences between the two trees with the *minimum* number of lateral transfers. We refer to this problem as the LATERAL GENE TRANSFER PROBLEM, in short as the LGT PROBLEM.

In this paper we investigate efficient approximability issues of the LGT PROBLEM. We consider both the special case of activity level one and the general case of activity level some $\alpha \geq 1$, and establish hardness of efficient approximation for both cases. More specifically, we will prove that, unless P=NP, no algorithm can achieve an approximation ratio smaller than $1 + \varepsilon$ for some constant $\varepsilon > 0$. By easy calculations we also provide explicit values of ε for the above claim. We also show that an upper bound on the cost of *any* 1-active lateral transfer scenario is given by $n - 2$ where n is the number of leaves in the gene tree, and show that this upper bound is tight by explicitly giving a pair of species and gene trees with this cost.

1.1 Basic Definitions and Notations

In the remaining sections we consider just rooted binary trees, *i.e.*, trees where all vertices have out-degree at most two and all arcs are directed from the root to the leaves. Given a rooted tree T, we denote with $V(T)$ the set of vertices and with $A(T)$ the set of arcs. The leaves of T are denoted by $L(T)$ and the root by $r(T)$. We say that two distinct vertices v, v' are children of u in T if $\langle u, v\rangle$, $\langle u, v'\rangle \in A(T)$. We denote the left son of a vertex $u \in V(T)$ as $ls_T(u)$, the right son as $rs_T(u)$, and the parent of u as $p_T(u)$.

Let F be a *rooted forest*, that is, a union of disjoint rooted trees. If two vertices $u, v \in V(F)$ are connected by a directed path from u to v, then v is a *descendent* of u in F, and we write $v \leq_F u$ (note that every node is a descendent of itself). If $u \neq v$, then v is a *proper descendent* of u in F ($<_F$). Similarly, we can define *ancestors* ($\geq_F$) and *proper ancestors* ($>_F$). Moreover, let T be a rooted tree and $X \subset V(T)$. Then $T[X]$ denotes the forest of subtrees induced by X.

Let $\{t_i : 1 \leq i \leq n\}$ be a forest of non-empty rooted directed trees over a label set L. We use the notation $T = \prec t_1 \cdot t_2 \cdot \ldots \cdot t_n \succ$ to represent the tree built

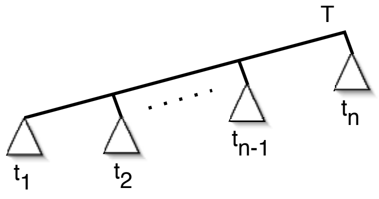


Fig. 1. The tree $T = \prec t_1 \cdot t_2 \cdot \ldots \cdot t_n \succ$

by connecting the subtrees t_i as shown in Figure 1. As a shorthand, we allow the notation $\prec \prod_{i=1}^{n} t_i \succ$ to mean $\prec t_1 \cdot t_2 \cdot \ldots \cdot t_n \succ$.

A *mixed graph* G is a graph where arcs may be both directed and undirected. We denote the set of directed arcs as $A(G)$ and the undirected arcs, or *edges*, as $E(G)$. $\varepsilon(A)$ indicates the set of edges underlying $A(G)$. Given a set of arcs A and a mixed graph G, we denote by $G \cup A$ the mixed graph with arcs $A(G) \cup A$, egdes $E(G)$, and vertices $V(G)$. Similarly is defined $G \cup E$, where E is a set of edges. A *directed mixed cycle* is a cycle on a mixed graph that may contain both arcs and edges, and where the cycle can be traversed so that the direction of the arcs that are part of the cycle is respected. Given a graph G and an arc $\langle x, y \rangle \in A(G)$, we say that we *subdivide* arc $\langle x, y \rangle$ if we replace it with a path from x to y that doesn't traverse any vertex in $V(G) \setminus \{x, y\}$. We say that a graph H is a *subdivision* of a graph G, if H is obtained from G by subdividing some of the arcs in $A(G)$ and adding new arcs between vertices in $V(H) \setminus V(G)$.

Finally, a $(1+\varepsilon)$-*approximate solution* (or simply an $(1+\varepsilon)$-approximation) of a minimization problem is a solution with an objective value no larger than $1+\varepsilon$ times the value of the optimum, and an algorithm achieving such a solution is said to have an *approximation ratio* of at most $1+\varepsilon$.

1.2 Basic Concepts of the Evolutionary Model

In this section we define some basic concepts for the evolutionary model we consider. We will first briefly introduce the concepts of a *gene tree* and a *species tree*. We will then present the concept of *least common ancestor mapping* and define a reconciliation model based on *lateral gene transfer events.*

Consider a set I of N biological taxa. The model for their evolutionary history is a rooted full binary tree S, where each of the N leaves is uniquely labeled by one element from I, and each internal node is unlabeled. Such tree S is called a *species tree*. An internal node in a species tree is equivalently treated as a subset (or *cluster*), which contains the labels of all leaves of the subtree rooted at that node. Thus, we can express the relation "m is a descendant of n" in set theory notation by $m \subset n$.

A *gene tree* T is a model for the evolution of a gene family. It is a rooted full binary tree where the internal nodes are unlabeled and the leaves are labeled by elements from I. As opposed to a species tree, labels in a gene tree may not be unique. In this case, internal nodes are represented by a multiset

$\{x_1^{i_1}, x_2^{i_2}, \ldots, x_m^{i_m}\}$, where i_j is the number of leaves labeled with x_j, among those reachable from the node. The *cluster* of an internal node is defined as the set $\{x_1, x_2, \ldots, x_m\}$.

Let Y be a rooted tree and $L(Y)$ the set of its leaf labels. The *least common ancestor (LCA)* of $X \subseteq L(Y)$, denoted by $lca_Y(X)$, is defined as the node $y \in Y$ such that $X \subseteq y$ and $X \not\subseteq w$ for every proper descendent w of y. Given a gene tree T and a species tree S such that $L(T) \subseteq L(S)$, we define $\lambda_{T,S} : V(T) \longrightarrow V(S)$ as a correspondence between nodes of the gene tree T and nodes of the species trees S. For any node $t \in T$, $\lambda_{T,S}(t)$ is the least common ancestor of t in S, *i.e.*, $\lambda_{T,S}(t) = lca_S(t)$. The function $\lambda_{T,S}$ is known as the *LCA mapping* from T to S.

1.2.1 The Lateral Gene Transfer Model

We are now ready to introduce the evolutionary model based on the concept of *lateral gene transfers.* This model was developed by Hallet and Lagergren [6]. It assumes a simplified evolutionary process where the only biological events that can occur are the so-called *lateral gene transfers.* In this framework, a natural problem is that of finding the most parsimonious *scenario* that explains in a biologically meaningful way how, via these events, the differences between the gene tree and the species tree arose. The definition of lateral transfer scenario is based on the concept of *lateral transfer scheme.*

Definition 1. *A* lateral transfer scheme *for a species tree S is a pair (S', A') where S' is a subdivision of S and $A' \subseteq \{\langle x, y\rangle : x, y \in V(S') \setminus V(S), x \neq y\}$ such that:*

1. *the mixed graph $S' \cup \varepsilon(A')$ does not contain a directed mixed cycle.*
2. *the tail of each arc in A' has in-degree 1 and out-degree 2 in $S' \cup A'$.*
3. *the head of each arc in A' has in-degree 2 and out-degree 1 in $S' \cup A'$.*

A lateral transfer scheme shows where the lateral transfers have occurred during evolution. The arcs in A' represent the *set of lateral transfers.* Note that the first condition in Definition 1 ensures that the scheme for a species tree S respects the partial order of evolution implied by S. Clearly, this is a required property for the model.

A lateral transfer scheme is meaningful when combined to the notion of *scenario.* A scenario is a mapping of a gene tree into a subdivision of a species tree. This mapping describes how the gene tree has evolved by showing at which point of evolution lateral gene transfers have occurred. In order for a scenario to be biologically meaningful, it must satisfy the conditions stated in the definition below.

An important parameter for this model is the *activity level* α. The parameter α measures the number of genes that are allowed to be simultaneously active in the genome of a taxa. Roughly speaking, an α-active scenario permits at most α copies of a gene to be mapped to the same ancestral taxon. In previous models, the presence of multiple copies of a gene was always assumed to be caused by a *gene duplication* event. The notion of acitivity level in a lateral transfer scenario overcomes this restriction, by postulating that this multiplicity may be generated by lateral transfer events alone.

We will first give the definition for the special case of 1-activity, and then state the definition for the general α-active case where $\alpha \geq 1$.

Definition 2. *A* 1-active lateral transfer scenario *(or* 1-active scenario*) for a species tree S and a gene tree T is a triple (S', A', g) where (S', A') is a lateral transfer scheme for S and $g : V(S') \to V(T)$ is a function such that:*

1. *$g(r(S')) = r(T)$.*
2. *if v_1 and v_2 are distinct children of v_0 in T, then there exists x_0 with distinct children x_1 and x_2 in $S' \cup A'$ such that $v_i \in g(x_i)$ for $i \in \{0, 1, 2\}$, and x_i is the $\leq_s$ -maximal vertex such that $v_i \in g(x_i)$ for $i \in \{1, 2\}$.*
3. *for each $v \in V(T)$, the vertices $\{x \in V(S') : v \in g(x)\}$ induce a directed path in S'.*
4. *$g(l) = l$, for all $l \in L(S)$.*

The cost *of a 1-active scenario (S', A', g) w.r.t. T is given by $|A'|$.*

Definition 3. *A* lateral transfer scenario *(or* scenario*) for a species tree S and a gene tree T is a triple (S', A', g) where (S', A') is a lateral transfer scheme for S and $g : V(S') \to 2^{V(T)}$ is a function such that:*

1. *$T[g(r(S'))]$ is connected and $r(T) \in g(r(S'))$.*
2. *if v_1 and v_2 are distinct children of v_0 in T and $v_1, v_2 \notin g(r(S'))$, then there exists x_0 with distinct children x_1 and x_2 in $S' \cup A'$ such that $v_i \in g(x_i)$ for $i \in \{0, 1, 2\}$, and x_i is the $\leq_S$ -maximal vertex such that $v_i \in g(x_i)$ for $i \in \{1, 2\}$.*
3. *if v_1 and v_2 are children of v_0 in T, $v_1 \in g(r(S'))$ and $v_2 \notin g(r(S'))$, then there exists a child x of $r(S')$ in S' such that $v_2 \in g(x)$.*
4. *for each $v \in V(T)$, the vertices $\{x \in V(S') : v \in g(x)\}$ induce a directed path in S'.*
5. *$g(x)$ is a $\leq_T$-antichain for each $x \in V(S') \setminus \{r(S')\}$.*
6. *$g(l) = \{l\}$, for all $l \in L(S)$.*

*A scenario (S', A', g) is α-*active *iff $max_{x \in S} |g(x)| = \alpha$. The* cost *of a α-active scenario (S', A', g) w.r.t. T is given by:*

$$\sum_{\langle x,y \rangle \in A} |\{\langle u, v\rangle \in A(T) : u \in g(x), v \in g(y)\}| + |V(T[g(r(S'))]) \setminus L(T[g(r(S'))])|$$

Let (S', A', g) be a scenario for S and T. We say that an arc of T is *invloved* into a lateral transfer if it belongs to the following set:

$$F = \{\langle u, v\rangle \in A(T) : u \in g(x), v \in g(y), \text{ where } \langle x, y\rangle \in A'\}$$

1.3 Problem Definitions

In this paper we investigate the following optimization problems:

1-active LGT Problem
Instance: A species tree S and a gene tree T, such that $L(T) \subseteq L(S)$.
Goal: Find a 1-active lateral transfer scenario for S and T with minimum cost.

α-active LGT Problem
Instance: A species tree S and a gene tree T such that $L(T) \subseteq L(S)$, a *constant* $\alpha \geq 1$.
Goal: Find an α-active lateral transfer scenario for S and T with minimum cost.

Note that one can easily convert the above optimization problems into their *decision version* by having an extra integer τ as input and requiring the minimum cost to be $\leq \tau$. We call the decision versions of these problems the 1-active τ-LGT Problem and α-active τ-LGT Problem respectively. It was shown in [7] that the α-active τ-LGT Problem is NP-complete.

1.4 Inapproximability Reductions: Key Concepts and Results

In [10] Papadimitriou and Yannakakis defined the class of *MAX-SNP* optimization problems and a special approximation-preserving reduction, the so-called *L-reduction*, that can be used to show MAX-SNP-hardness of an optimization problem. The version of the L-reduction that we provide below is a slightly modified but equivalent version that appeared in [4].

Definition 4. [4, 10] *Given two optimization problems Π and Π', we say that Π* L-reduces *to Π' if there are three polynomial-time procedures T_1,T_2, T_3 and two constants a and $b > 0$ such that the following two conditions are satisfied:*

1. *For any instance I of Π, algorithm T_1 produces an instance $I' = f(I)$ of Π' generated from T_1 such that the optima of I and I', $OPT(I)$ and $OPT(I')$, respectively, satisfy $OPT(I') \leq a \cdot OPT(I)$.*
2. *For any solution of I' with cost c', algorithm T_2 produces another solution with cost c'' that is no worse than c', and algorithm T_3 produces a solution of I of Π with cost c (possibly from the solution produced by T_2) satisfying $|c - OPT(I)| \leq b \cdot |c'' - OPT(I')|$.*

An optimization problem is MAX-SNP-*hard* if any problem in MAX-SNP L-reduces to that problem. If this problem is also in MAX-SNP, then it is MAX-SNP-*complete*. The importance of proving MAX-SNP-hardness results comes from a result proved by Arora et al. [1] which shows that, assuming P$\neq$NP, for every MAX-SNP-hard problem there exists a constant $\varepsilon > 0$ such that no polynomial time algorithm can achieve an approximation ratio better than $1+\varepsilon$.

1.5 Precise Statements of Our Results

Theorem 1
(a) For some constant $\varepsilon > 0$, it is not possible to approximate in polynomial time the 1-active LGT PROBLEM within an approximation ratio of $1+\varepsilon$ unless P=NP.
(b) The constant ε in (a) is at least $(3/370024) - \kappa$ for any $\kappa > 0$.

Theorem 2
(a) For some constant $\varepsilon > 0$, it is not possible to approximate in polynomial time the α-active LGT PROBLEM within an approximation ratio of $1+\varepsilon$, where $\alpha \geq 1$, unless P=NP.
(b) The constant ε in (a) is at least $(3/378068) - \kappa$ for any $\kappa > 0$.

Lemma 1. *The minimum number of lateral transfers necessary to build a 1-active lateral transfer scenario for any pair of gene and species trees, uniquely labeled over the same set of labels L, is precisely $n - 2$ where $n = |L|$. That is, there is a procedure to build a 1-active scenario of cost $n - 2$ and there exists a pair of a gene tree and a species tree that require at least $n - 2$ lateral transfers for any 1-active scenario.*

2 Hardness of Approximation of 1-Active LGT PROBLEM (Proof of Theorem 1(a))

In the following we will show that MAX-2SAT-B L-reduces to the 1-active LGT PROBLEM. MAX-2SAT-B is the variation of MAX-2SAT where the number of occurrences of each variable is bounded by a constant B. It is known from [2] that MAX-2SAT-B is MAX-SNP-complete for $B \geq 3$; thus the existence of an L-reduction will imply the result.

Let $X = \{X_1, \ldots, X_n\}$ be a set of n variables and let $\Phi = (C_1, \ldots, C_m)$ be a formula in 2-CNF, where each clause C_i is on two variables from X and where the number of occurrences of each variable is bounded by a constant B. We will refer to the j^{th} variable in the i^{th} clause as literal $C_{i,j}$. The goal of MAX-2SAT-B is to find a truth assignment on X that maximizes the number of satisfied clauses. Given an instance of MAX-2SAT-B, we will now exhibit how to build an instance of the 1-active LGT PROBLEM such that Conditions 1 and 2 of Definition 4 are satisfied. In other words, we will construct a gene tree T and a species tree S from Φ and prove that this transformation is an L-reduction.

Our construction of T and S from Φ is taken from the NP-completeness proof given in [7]. The only difference is that in our case the index j in $C_{i,j}$ ranges between 1 and 2, rather than 1 and 3 (their reduction is from 3SAT). A detailed description of this procedure is here omitted.

An important parameter used in the following proof is τ, defined as $\tau = 9m + 6k$, where

$$k = \left|\{\langle i,j,i',j'\rangle \mid C_{i,j} = C_{i\ ,j} \text{ or } C_{i,j} = \overline{C_{i\ ,j}}\,, 1 \leq i < i' \leq m, 1 \leq j,j' \leq 2\}\right|.$$

Notice that $k \leq \frac{nB(B-1)}{2}$, since the maximum number of occurrences is bounded by B for each variable in Φ; hence, $\tau \leq \gamma m$, where $\gamma = 9 + 6B(B-1)$. Also, let $\tau^{+} = \tau + m + 1$. To ease our presentation, we will adopt the same notation used in [7] throughout the rest of the proof.

Let Ψ be a truth assignment on the variable set X, i.e. $\Psi : X_i \longmapsto \{\text{true, false}\}$, for every $i = 1, \ldots, n$. We show that there is a correspondence between truth assignments on Φ and scenarios for T and S. The proof of the following claim is omitted due to page limits.

Claim 1. *Given a truth assignment Ψ on Φ that satisfies ρ clauses, $\rho \leq m$, it is always possible to build in polynomial time a 1-active lateral transfer scenario for T and S with cost $\tau + (m - \rho)$.*

It is now easy to show that the first condition of Definition 4 is satisfied. Starting from an optimal truth assignment Ψ_{OPT} that satisfies OPT_{Φ} clauses from Φ, by Claim 1 we can build a 1-active scenario of cost $\tau + (m - OPT_{\Phi})$. If we denote by $OPT_{T,S}$ the cost of the optimal scenario on T and S, then $OPT_{T,S} \leq \tau + (m - OPT_{\Phi}) \leq (\gamma + 1)m$. Moreover, it is not hard to see that a random truth assignment satisfies each clause with probability 3/4, and hence it is not hard to find (even deterministically) an assignment OPT_{Φ} that satisfies $3m/4$ clause (*e.g.*, see [8]). Thus, without loss of generality we may assume that $OPT_{\Phi} \geq 3m/4$. By combining the two inequalities we have $OPT_{T,S} \leq a \cdot OPT_{\Phi}$, where $a = \frac{4(\gamma+1)}{3}$. This completes the first part of the proof.

Lets now verify the second condition. Suppose we are given a 1-active lateral transfer scenario for T and S of cost c'. We can assume without loss of generality that $c' \leq \tau + m$, otherwise we could choose any scenario built from an arbitrary assignment to replace the given one. Observe that Claims 1-8 in [7] are true for the given scenario, while Claim 9 in [7] must be slightly modified to fit our construction. This is the modified result:

Claim 2. *[7] In any 1-active scenario for T and S, at least one element of $X_i = \{r(T_{i,j}) : 1 \leq j \leq 2\}$ is the tail of an arc involved in a lateral transfer, for every i. This requires $\geq m$ lateral transfers.*

Proof. Follows from the observation that, by the 1-activity conditions, only one element from X_i may be mapped to $r(B_{S_i})$. ❑

Note that Claims 1-8 in [7] together with Claim 2 imply that a lower bound on the cost of any lateral transfer scenario for T and S is equal to τ. We now show that, starting from the given scenario, it is always possible to build in polynomial time a new scenario of cost $\leq c'$, which induces a consistent truth assignment on Φ. *This new scenario and its induced truth assignment will satisfy Condition 2 of Definition 4 with* $b = 1$.

Let X_v be a variable from the variable set X and $\Omega_v = \{C_{i,j} \mid X_v$ appears in $C_{i,j}\}$, $\omega_v = |\Omega_v|$. Let $\tilde{T}_{i,j} = \prec\prec a_{i,j} \cdot c_{i,j} \succ \cdot \prec b_{i,j} \cdot d_{i,j} \succ\succ$ and F_v be a forest of subtrees of T defined as $F_v = \{\tilde{T}_{i,j} \mid C_{i,j} \in \Omega_v\} \cup \{\varepsilon'_{i,j,i\ ,j} \mid C_{i,j}, C_{i\ ,j} \in \Omega_v\}$. Moreover, let $k_v = |\{\langle i,j,i',j'\rangle \mid C_{i,j}, C_{i\ ,j} \in \Omega_v$ and $i < i'\}|$ and define $\tilde{\tau}_v = 2\omega_v + 4k_v$.

We say that a literal $C_{i,j}$ is *well-assigned* if the corresponding gene subtree $\tilde{T}_{i,j}$ has exactly two arcs involved in lateral transfers. This implies that either $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(c_{i,j}, d_{i,j}))$ or $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(a_{i,j}, b_{i,j}))$.

We also say that literal $C_{i,j}$ is *inconsistent* with respect to $C_{i\ ,j\ }$ if one of the two following conditions holds:

– $C_{i,j} = C_{i\ ,j\ }$, and i) $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(c_{i,j}, d_{i,j})$ and $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \notin g(lca_S(c_{i\ ,j\ }, d_{i\ ,j\ })$ or ii) $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(a_{i,j}, b_{i,j})$ and $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \notin g(lca_S(a_{i\ ,j\ }, b_{i\ ,j\ })$.
– $C_{i,j} = \overline{C_{i\ ,j\ }}$, and i) $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(c_{i,j}, d_{i,j})$ and $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \notin g(lca_S(a_{i\ ,j\ }, b_{i\ ,j\ })$ or ii) $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(a_{i,j}, b_{i,j})$ and $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \notin g(lca_S(c_{i\ ,j\ }, d_{i\ ,j\ })$.

Let $I_{i,j}$ be the set of all literals that are inconsistent w.r.t. $C_{i,j}$ and $i_{i,j} = |I_{I,J}|$.

Claim 3. *If no $C_{i,j} \in \Omega_v$ is well-assigned, then at least $\tilde{\tau}_v + \omega_v$ arcs in F_v are involved in lateral transfers. If there exists a well-assigned $C_{i,j} \in \Omega_v$, then at least $\tilde{\tau}_v + i_{i,j}$ arcs in F_v are involved in transfers.*

Proof. By Claims 7 and 8 in [7], $\tilde{\tau}$ is a lower bound on the number of arcs in F_v that are involved in lateral transfers. The first part of the claim follows immediately from the fact that for each non well-assigned literal $C_{i,j}$ at least three transfers are required for $\tilde{T}_{i,j}$, that is an additional transfer for every literal w.r.t. the minimum scenario.

Now suppose that $C_{i,j} \in \Omega_v$ is well-assigned, and assume that $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(c_{i,j}, d_{i,j}))$, i.e. $C_{i,j}$ is true. Consider a second literal $C_{i\ ,j\ } \in \Omega_v$ that is inconsistent w.r.t. $C_{i,j}$, and assume for example that $C_{i,j} = C_{i\ ,j\ }$. If $C_{i\ ,j\ }$ is not well-assigned, then $\tilde{T}_{i\ ,j\ }$ has at least three arcs involved in lateral transfers. Conversely, if $C_{i\ ,j\ }$ is inconsistent and well-assigned, it is straightforward to verify that $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \in g(lca_S(a_{i\ ,j\ }, b_{i\ ,j\ }))$, since any different mapping would require more than two arcs from $\tilde{T}_{i,j}$ involved in transfers. Moreover, the path from $p_T(b_{i\ ,j\ })$ to $a_{i\ ,j\ }$ in T blocks the path from $p_S(\beta_{i\ ,j\ ,i,j})$ to $\alpha_{i\ ,j\ ,i,j}$ in S, where by *blocks* we mean that at least one vertex belonging to the path on T is mapped to a vertex from the path on S. Equally, the path from $p_T(d_{i,j})$ to $c_{i,j}$ in T blocks the path from $p_S(\delta_{i,j,i\ ,j\ })$ to $\gamma_{i,j,i\ ,j\ }$ in S. Now, since both paths are blocked, for any valid scenario built under these assumptions at least three arcs in the subtree $\varepsilon'_{i\ ,j\ ,i,j}$ must be involved in lateral transfers. An example of this case is given in figure 2.

For the symmetric case where $lca_T(c_{i,j}, d_{i,j}) \in g(lca_S(a_{i,j}, b_{i,j}))$ and $lca_T(c_{i\ ,j\ }, d_{i\ ,j\ }) \in g(lca_S(c_{i\ ,j\ }, d_{i\ ,j\ }))$, i.e. $C_{i,j}$ is false and $C_{i\ ,j\ }$ is true, a similar argument shows that $\tilde{T}_{i\ ,j\ }$ or $\varepsilon'_{i,j,i\ ,j\ }$ require at least three lateral transfers.

We can therefore conclude that in both cases at least seven arcs from the subtrees $\tilde{T}_{i\ ,j\ }$, $\varepsilon'_{i,j,i\ ,j\ }$ and $\varepsilon'_{i\ ,j\ ,i,j}$ are involved in lateral transfers; that is, the given scenario requires on these subtrees at least one transfer more than the minimum scenario, which reaches the lower bound of six implied by Claims 7-8 in [7]. A similar reasoning establishes the same result for the cases where $C_{i,j} = \overline{C_{i\ ,j\ }}$.

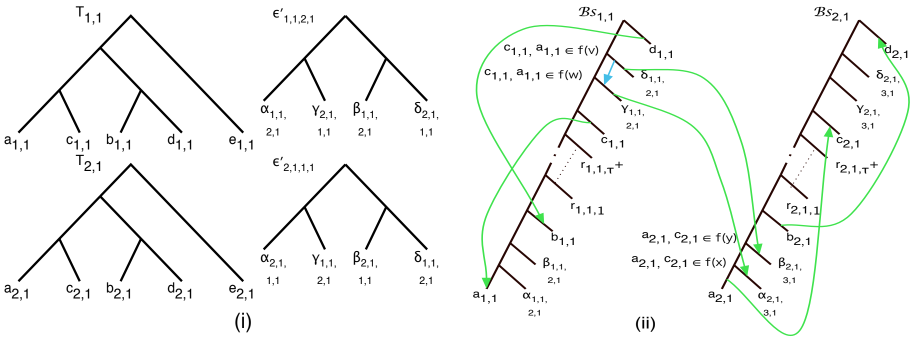


Fig. 2. Consider the formula $\Phi = (X_1 \vee \overline{X_2})(X_1 \vee X_2)$. (i) depicts the twister gadgets $(T_{1,1}, T_{2,1})$ and enforcement gadgets $(\varepsilon'_{1,1,2,1}, \varepsilon'_{2,1,1,1})$ corresponding to the literals $(C_{1,1}, C_{2,1})$ which are instances of $X_1 \in X$. (ii) represents a partial scenario where literals $C_{1,1}$ and $C_{2,1}$ are well-assigned and inconsistent. The dashed lines are the lateral transfers. Notice that both paths from $p_S(\delta_{1,1,2,1})$ to $\gamma_{1,1,2,1}$ and from $p_S(\beta_{2,1,1,1})$ to $\alpha_{2,1,1,1}$ are blocked by $\{c_{1,1}, a_{1,1}\}$,$\{a_{2,1}, c_{2,1}\} \in T$ respectively. Thus, an additional transfer from arc $\langle p_S(\delta_{1,1,2,1}), \delta_{1,1,2,1}\rangle$ to $\langle p_S(\gamma_{1,1,2,1}), \gamma_{1,1,2,1}\rangle$ is necessary.

Hence, for every literal not consistently assigned w.r.t $C_{i,j}$, at least one additional lateral transfer is required. Thus, $\geq \tilde{\tau_v} + i_{i,j}$ arcs of F_v are involved in transfers. ❑

We will now describe a simple procedure to build a new scenario of cost $\leq c'$ that is based on the given scenario as a starting point. Construct a truth assignment $\Psi : X \longrightarrow \{\text{true, false}\}$ on Φ in the following way. For each variable X_v, $v = 1, \ldots, n$, check if there exists (can be done in linear time) some literal $C_{i,j} \in \Omega_v$ which is well-assigned. If this is the case, assign to X_v the truth value read from $C_{i,j}$, i.e. $\Psi(X_v) = \text{true}$ if $lca_T(c_{i,j}, d_{i,j}) \in lca_S(c_{i,j}, d_{i,j})$ and $\Psi(X_v) = \text{false}$ if $lca_T(c_{i,j}, d_{i,j}) \in lca_S(a_{i,j}, b_{i,j})$. If no literal in Ω_v is well-assigned, then assign to X_v an arbitrary truth value. Now follow the procedure described in Claim 1 and build a scenario from Ψ. As a shorthand, call LTS_I the given scenario and LTS_F the new scenario built from Ψ.

We say that a clause C_i is *satisfied* by a lateral transfer scenario on T and S if exactly one element from the set $\{r(T_{i,j}) \mid 1 \leq j \leq 2\}$ is the tail of an arc involved in a lateral transfer. Clearly, this is true only if $\exists j$ s.t. $lca_T(c_{i,j}, d_{i,j}) \in lca_S(c_{i,j}, d_{i,j})$.

LTS_F has cost $c_a = \tau + (m - \rho)$, where ρ is the number of clauses that Ψ satisfies. In other words, LTS_F has a minimum number of transfers on all subtrees of S, except on the subtrees B_{S_i} corresponding to those clauses C_i that are not satisfied by Ψ, where an additional lateral transfer (w.r.t. the minimum cost scenario on this tree) is required. Claims 1-8 in [7] and Claim 2 establish that τ is a lower bound for any valid scenario, hence $c' \geq \tau$. Moreover, a clause that is not satisfied in LTS_F can be satisfied by LTS_I only by a non well-assigned literal $C_{i,j}$ (in the case where $C_{i,j} \in \Omega_v$ and X_v is arbitrarily assigned) or by a literal which is inconsistent w.r.t. the chosen assignment $\Psi(X_v)$. By Claim 3, this implies that LTS_I has at least one lateral transfer more than the minimum scenario for each clause that is true in LTS_I and false in LTS_F. Hence, $c' \geq \tau + (m - \rho)$.

Therefore, given any scenario on S and T, we are able to build in polynomial time a new scenario of cost $\tau + (m - \rho) \leq c'$ which corresponds to a valid truth assignment on Φ that satisfies ρ clauses. The theorem follows.

3 Hardness of Approximation of the α-Active LGT PROBLEM (Proof of Theorem 2(a))

Once again, we L-reduce from Max-2Sat-B. Let T^* and S^* respectively be the gene and species tree of the α-active LGT Problem instance. Build T^* and S^* by following the procedure given in [7]. The details on this construction are here omitted. Starting from an optimal truth assignment Ψ_{OPT} that satisfies OPT_Φ clauses from Φ, first create a 1-active scenario on T and S by applying the construction described in Claim 1. Use this scenario to construct an α-active scenario for T^* and S^* as shown in [7]. The cost of this scenario is $\tau^* + (m - OPT_\Phi)$, where $\tau^* = \tau + (\alpha - 1)$, and hence $OPT_{T^*,S^*} \leq \tau^* + (m - OPT_\Phi)$. From Theorem 1, we know that $\tau \leq \gamma m$, where $\gamma = 9 + 6B(B-1)$. *For all sufficiently large values* of m, we have $\tau^* \leq (\gamma + 1)m$ and $OPT_{T^*,S^*} \leq (\gamma + 2)m$. Thus, the the first condition from Definition 4 is satisfied, with $a = \frac{4(\gamma+2)}{3}$.

Consider now an α-active scenario for T^* and S^* of cost c^*. We can assume w.l.o.g. that $c^* \leq \tau^* + m$; if this were not the case, any scenario built from an arbitrary truth assignment would do better, and we could use this scenario as a starting point. Notice that any α-active scenario requires at least $\alpha - 1$ lateral transfers for subtrees $T^1, \ldots, T^{\alpha-1}$ and S^*. Therefore, at most $\tau + m$ transfers are involved in the partial scenario for T and S^*. By Claim 11 of [7], the α-active scenario for T and S^* induces a 1-active scenario for T and S of cost $\leq \tau + m$. It has been shown that any 1-active scenario for T and S of cost c' induces a truth assignment on Φ that satisfies ρ clauses, where ρ is s.t. $c' \geq \tau + (m - \rho)$. It follows that $c^* = c' + (\alpha - 1) \geq \tau^* + (m - \rho)$. Thus, the the second condition of Definition 4 is satisfied with $b = 1$. This concludes our proof.

4 Hard Inapproximability Bounds (Proofs of Theorem 1(b) and Theorem 2(b))

Berman and Karpinski [3] proved that it is NP-hard to approximate Max-2Sat-3 to within a factor $2012/2011 - \kappa$, for every $\kappa > 0$. The following result from [10] allows us to compute approximation ratios that are NP-hard to achieve for the 1-active and α-active LGT Problem from that of Max-2Sat-3.

Proposition 1. [10] *Let Π and Π' be two optimization problems. If Π L-reduces to Π' , and there is a polynomial time approximation algorithm for Π' with* *worst-case error ε, then there is a polynomial time approximation algorithm for Π with worst-case error $ab\varepsilon$, where a and b are the constants of the L-reduction.*

Proposition 1 can be stated equivalently as follows: if approximating Π to within an approximation ratio smaller than $1+\varepsilon$ is NP-hard, then achieving an approximation ratio for Π' smaller than $1+\varepsilon/(ab)$ is also NP-hard.

Consider the L-reduction for the 1-active case. We have $a = \frac{40+24B(B-1)}{3}$, which implies $a = 184/3$ for $B = 3$, and $b = 1$. It follows that, for the 1-active LGT Problem it is NP-hard to achieve an approximation ratio of $370027/370024 - \kappa$, for every $\kappa > 0$.

Similarly, for the α-active case, $a = \frac{44+24B(B-1)}{3}$, which implies $a = 188/3$ for $B = 3$, and $b = 1$. This shows that it is NP-hard to approximate the α-active LGT Problem to within a factor of $378071/378068 - \kappa$, for every $\kappa > 0$.

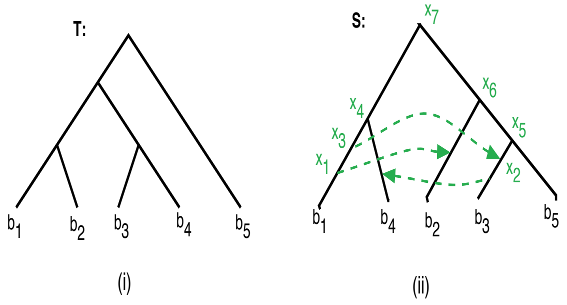


Fig. 3. An example of scenario built from the construction procedure illustrated below. (i) is the gene tree and (ii) the scenario built on the species tree. The dashed arcs are the lateral transfers. Here, $\{b_1b_2\} \in g(x_1)$, $\{b_3b_4\} \in g(x_2)$, $\{b_1b_2b_3b_4\} \in g(x_3)$ and $g(x_4)$, $\{b_5\} \in g(x_5)$ and $g(x_6)$, $r(T) \in g(r(S))$. Note that this isn't the minimum scenario.

5 Upper Bound of Cost of 1-Active Scenario (Proof of Lemma 1)

We first describe a procedure to build a 1-active scenario of cost $n-2$ for any given gene and species trees. Let T be a gene tree and S be a species tree that satisfy $L(T) = L(S)$. We order the *internal* vertices (i.e. all vertices except the leaves) of T, by imposing the ordering produced by a post-order traversal on the subtree of T containing the internal nodes only. Recall that a post-order traversal processes all vertices of a tree by recursively visiting all subtrees, then finally processing the root. Let $\{a_1, a_2, \ldots, a_{n-1}\}$ be the ordered sequence of internal vertices, where a_{n-1} is the root of T.

In the following description we will slightly abuse of notation, by applying the concepts of parent and children to nodes of S'. In this context, $p_S\,(v)$, where $v \in V(S')$ has in-degree one, denotes the tail of v's unique incoming arc; $ls_S\,(v)$ and $rs_S\,(v)$, where $v \in V(S') \cap V(S)$, respectively refer to the nodes of S' that

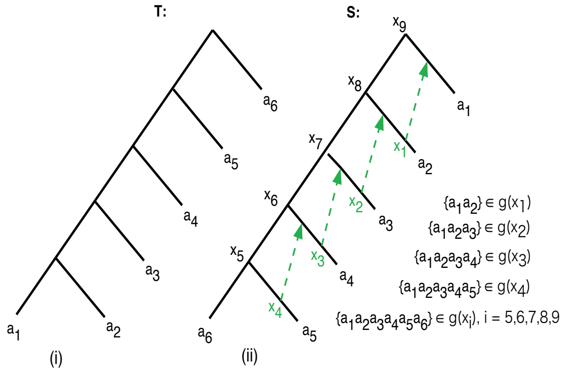


Fig. 4. An example of gene tree T (i) and the species tree S (ii) with $n = 6$. The dashed arcs in (ii) represent the lateral transfers. Note that 4 transfers are necessary for any scenario.

are first in the paths from v to $ls_S(v)$ and from v to $rs_S(v)$. We will now build a scenario (S', A', g) for T and S by creating the following lateral transfers:

– from arc $\langle p_S\ (g^{-1}(ls_T(a_i))), g^{-1}(ls_T(a_i))\rangle$ to arc $\langle p_S\ (g^{-1}(rs_T(a_i))), g^{-1}\ (rs_T(a_i))\rangle$, involving $\langle a_i, rs_T(a_i)\rangle$ from T,

for every i from 1 to $n-2$. Note that the post-ordering ensures that both children of a_i have been processed when vertex a_i is considered, hence $g^{-1}(rs_T(a_i))$ and $g^{-1}(rs_T(a_i))$ are defined. Also, observe that all lateral transfer in this scenario are incident on arcs of S that connect parents to leaves. Now, let $b = lca_S\ (g^{-1}(ls_T(r(T)), g^{-1}(ls_T(r(T))))$, and for all vertices v_i belonging to the path connecting $ls_S\ (b)$ to $g^{-1}(ls_T(r(T))$, place $ls_T(r(T) \in g(v_i)$. Similarly, for all vertices u_i belonging to the path connecting $rs_S\ (b)$ to $g^{-1}(rs_T(r(T))$, place $rs_T(r(T) \in g(u_i)$. Finally, map all vertices in the path from the root of S to b, to the root of T.

It is straightforward to verify that the resulting scenario does not contain mixed cycles, since all transfers are non-intersecting by construction. In addition, all conditions from Definition 2 are satisfied, hence the procedure yields a valid 1-active scenario. It is also easy to see that the running time of this algorithm is linear in the size of the trees.

We now show that this upper bound is tight by giving a simple example of a gene tree and a species tree that require at least $n-2$ lateral transfers for any 1-active scenario.

Let L be the set of labels $\{a_i : 1 \le i \le n\}$, where n is a positive constant. Consider the gene tree T and species tree S shown in figure 4, both over the same set of labels L:

$$T = \prec \prod_{i=1}^{n} a_i \succ$$

$$S = \prec \prod_{i=n}^{1} a_i \succ$$

We will show that any 1-active scenario for T and S requires at least $n-2$ lateral transfers. The result follows from the following Claim.

Claim 4. *Let $X = \{p_T(a_i) : 1 < i < n\}$, that is, X is the set all internal nodes of T except the root. In any 1-active lateral transfer scenario for T and S, every element of X is tail of an arc involved in a lateral transfer.*

Proof. Assume that a node $x \in X$ is not tail of an arc of T involved in a lateral transfer. This means that $x \notin g(v)$, for all nodes $v \in V(S') \setminus V(S)$, which implies that there exists a node $y \in S$ such that $x \in g(y)$, where $y \geq_S lca_S(x)$. But $lca_S(x) = r(S)$ for all $x \in X$, hence $y = r(S)$. This is a contradiction, since $r(T) \in g(r(S))$, by Condition 1 of Definition 2, and $x \neq r(T)$. It follows that x must be involved in a lateral transfer. ❑

Therefore, any 1-active scenario for T and S has at least $|X| = n - 2$ lateral transfers.

References

1. Arora, S., Lund, C., Motwani, R. Sudan, M. and Szegedy M. (1998), Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3): 501-555.
2. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti Spaccamela, A., and Protasi, M. (1999), *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*, Springer-Verlag, Berlin.
3. Berman, P., and Karpinski, M. (1998), On some tighter inapproximability results, further improvements, Technical Report TR98-065, *Electronic Colloquium on Computational Complexity* (ECCC); available online from `http://eccc.uni-trier.de/eccc-reports/1998/TR98-065/index.html`.
4. Berman, P. and Schnitger, G. (1992), On the complexity of approximating the independent set problem, *Information and Computation*, 96, 77-94.
5. Brown, J. R. (2003), Ancient horizontal gene transfer, *Nature Reviews, Genetics*, 4, 121-132.
6. Hallett, M. and Lagergren, J. (2001), Efficient algorithms for lateral gene transfer problems, Proc. 5^{th} Annual International Conference on Computational Molecular Biology (RECOMB), Montreal, Canada, 141-148.
7. Hallett, M. and Lagergren, J. (2004), Identifying lateral gene transfer events, submitted to *SIAM Journal of Computing* (available online from `http://www.mcb.mcgill.ca/~hallett/Lateral.pdf`)
8. Johnson, D.S. (1974), Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, 9, 256-278.
9. Page, R. D. M. and Charleston, M. A. (1997), From gene to organismal phylogeny: Reconciled tree and the gene tree/ species tree problem, *Molecular Phylogentics and Evolution*, 7, 231-240.
10. Papadimitriou, C. H. and Yannakakis, M. (1991), Optimization, approximation, and complexity classes, *Journal of Computer and System Sciences*, 43(3): 425-440.

Faster Deterministic Wakeup in Multiple Access Channels*

Gianluca De Marco[1], Marco Pellegrini[2], and Giovanni Sburlati[2]

[1] Dipartimento di Informatica e Applicazioni,
Università di Salerno, 84081 Baronissi (SA), Italy
demarco@dia.unisa.it

[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche,
via Moruzzi 1, 56124 Pisa, Italy
{marco.pellegrini, giovanni.sburlati}@iit.cnr.it

Abstract. We consider the fundamental problem of waking up n processors sharing a multiple access channel. We assume the weakest model of synchronization, the locally synchronous model, in which no global clock is available: processors have local clocks ticking at the same rate, but each clock starts counting the rounds in the round in which the correspondent processor wakes up. Moreover, the number n of processors is not known to the processors. We propose a new deterministic algorithm for this problem, which improves on the currently best upper bound.

Keywords: Algorithms, clock, synchrony, wakeup problem, multiple access channel.

1 Introduction

We consider the following model of multiple access channel, taken as the basis for theoretical studies on radio networks, satellite channels, serial bus communication networks. There are n processors (stations) sharing a common communication channel. The communication system is synchronous, in the sense that the processors send messages in rounds. Once a message is written on the channel, the message is broadcast to all other processors. Unfortunately, since the channel is shared by all processors, a collision among several processors attempting to write in the same round might occur. Precisely, the channel has the following property: a message is written successfully on the channel in a given round (and therefore heard by all processors) if and only if exactly *one* processor sends a message in that round.

Moreover we assume that the processors have no possibility of *collision detection*, that is, if more than one processor (or no processor at all) send in the same round, then nothing is heard by the other processors, so making it impossible to distinguish between multi-transmission and absence of transmission. Formally, if δ is the number of processors that transmit in a given round, three cases may happen:

* Work supported in part by the European RTN Project under contract HPRN-CT-2002-00278,COMBSTRU.

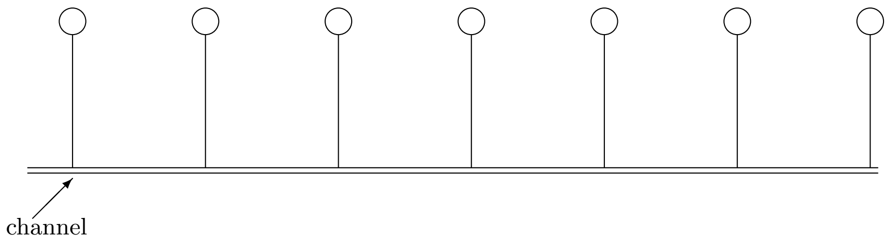


Fig. 1. A multiple access channel with $n = 7$ processors

1. $\delta = 0$: in this case, of course, no message is written on the channel;
2. $\delta = 1$: the message is written successfully on the channel and therefore heard by every processor;
3. $\delta > 1$: in this case, the δ simultaneous transmissions interfere with one another (a collision occurs) and therefore no message is written successfully on the channel.

It is clear that when no collision detection mechanism is available to the processors, cases 1. and 3. are completely undistinguishable.

A central issue is the measurement of time. Usually two extreme models are considered: the *globally synchronous* and the *locally synchronous* model. In the first model all the processors have access to a global clock. This implies that when a processor wakes up, it can see the current round number ticked by the clock. In the weaker *locally synchronous* model, each processor has its own local clock. This means that although the communication is synchronous (*i.e.* all the clocks tick with the same rate) there is no global round number. Each clock starts counting the rounds in the round in which the correspondent processor wakes up, without knowing anything about the other round numbers. In this paper, we adopt the weaker locally synchronous model.

Finally a crucial assumption is whether the processors using the shared channel are aware of the total number n of processors sharing the channel, or some polynomially related upper bound to such number. When such number n is known, much faster algorithms are possible. In keeping the line of using the weakest possible model, we assume such number to be unknown to the processors.

1.1 The Problem

One of the major problems in distributed computing is the wakeup problem that consists of having a subset of processors that spontaneously awakes and has to inform all the other processors about the beginning of the computation. This problem assumes particular importance in the locally synchronous model. Indeed, as already mentioned, once one of the processors manage to send successfully its message on the channel, the message is heard by all other processors. This characteristic can be exploited to allow the processors of a locally synchronous model to switch to the often more desirable globally synchronous one.

In fact, assume that at a given round all processors hear the message sent by one of them. Starting from this round, all processors can begin to count the successive rounds with the same numbers $1, 2, ...$ That means a global clock has been created. This allows the processors to use simple communication collision-free protocols such as *time division multiplexing* TDM, in which the principle is to assign each processor a round in which the processor can transmit (which is possible only if a global clock is available).

An algorithm for the wake up problem is a collection of n transmission schedules, one for each processor, such that one of the processors eventually transmits successfully to the channel, therefore waking up every processor. This will happen in the first round at which exactly one processor sends a message.

The problem is to design an algorithm that wakes up the system as fast as possible. The difficulty comes from the fact that the algorithm has no control on the processors that *spontaneously* can wake up at any moment during the execution, therefore disturbing the communication. In other words, we assume to be playing against an adversary who controls which processors wake up and when. The time complexity is measured by the time elapsed from the round in which the first processor woke up to the round in which a processor sends successfully on the channel, therefore waking up all the system.

The wakeup problem addressed in this paper, as will be discussed in the next subsection, has been introduced in [13] and can be stated formally as follows.

Definition 1 ((Wakeup Problem)). *We are given a multiple access channel with n processors, where the parameter n is* not *known by the processors. Assume there is no global clock (locally synchronous model) and no collision detection mechanism. Suppose that processors* spontaneously and independently *can wake up at any moment. Let τ_0 be the first time slot such that some processor is woken up. Assign n transmission schedules, one for each processor, such that there exists a time slot τ_1 such that exactly one processor (among the awaken processors at τ_1) sends a message at τ_1. The aim is to assign the transmission schedules in such a way to minimize $\tau_1 - \tau_0$.*

1.2 Related Work

Multiple access channels includes many network systems such as Aloha multi-access systems, local area Ethernet networks, satellite communication systems, packet radio networks that have been studied extensively in the literature [2,22]. In some of these models it is frequently assumed that a collision detection mechanism is available. As already mentioned, such a tool allows the transmitting processors to detect if its message has collided and therefore simplifies the wakeup problem. When collision detection is not available, the collisions must be *resolved* (*collision resolution*) by establishing a schedule of access to the shared communication channel that allows messages to be successively sent on the channel with as little delay as possible.

Collision detection and resolution, and access management algorithms were studied mainly assuming a known probability distribution on the arrival rate of messages at the different processors (see *e.g.* [11,12,16]). Moreover, in these contexts the wakeup problem, as defined in this paper, is not considered.

As already anticipated, when collision resolution is not available, but there is a global clock, one of the simplest schedule to resolve conflicts is the time division multiplexing protocol. This means that when there are n processors, then n time slots will be needed. This becomes very inefficient when the number of awaken processors is very small compared to n.

Komlós and Greenberg [20] were the first to considered a typical situation when a subset of k among n processors are awaken and have messages, and all of them need to be sent to the channel successfully as soon as possible. The fact that in their case *all of the messages* must be sent on the channel (contrasted with our wake up problem) does not mean a real difference with our situation, since their algorithm, stopped at the first successful message sent, is actually a wake up algorithm. On the other hand, there are strong differences with our case given by the fact that in [20] the number k of awaken processors is fixed and a global clock is available. They showed how to solve the problem deterministically in time $\Omega(k + k\log(n/k))$, where either n or k is known. A lower bound of $\Omega(k(\log n)/(\log k))$ was then proved by Greenberg and Winograd [14].

The work of [20] is, to the best of our knowledge, the first that shows how to exploit *deterministically* the fact that any processor that has already transmitted successfully its message on the channel, will not transmit in the subsequent time slots, therefore avoiding to interfere with the other processors that still have to transmit.

The issues discussed in [20] are also important for their relations with Coding Theory, in particular with combinatorial structures like superimposed codes [10,21] and its applications to combinatorial group testing. The reader interested can refer to the excellent book of Du and Hwang [8] (specially Chapter 4 and 5) for a more detailed study of the implications involved, and to the works of Indyk [17] and De Bonis and Vaccaro [6] for recent developments and generalizations.

Gąsieniec, Pelc and Peleg [13] were the first to consider the problem of waking up a multiple access channel with n processors in the synchronous setting when the number of awaken processors is not fixed, but can be any non-decreasing function of the time. In [13] many variations of synchrony and knowing assumptions are studied.

The authors of [13] considered both deterministic and randomized algorithms. For the deterministic case, which is the topic of the present paper, their result can be summarized as follows. In the globally synchronous model, when the size n is known to processors, they show an optimal deterministic algorithm that in time n solves the wakeup.

In case of unknown n, they construct a deterministic wakeup algorithm working in time $4n$ in the worst case.

Under the locally synchronous model with known n, they provide a deterministic $O(n^2 \log n)$ algorithm. They also show that even when n is known, every deterministic algorithm requires time at least $(1 + \epsilon)n$, for some $\epsilon > 0$, in the worst case. In the locally synchronous model when n is unknown, they propose a deterministic algorithm working in time $O(n^4 \log^5 n)$.

As for the randomized solutions (not discussed here), recent important improvements have been provided by Jurdiński and Stachowiak [19].

The wake up problem has been studied also in multi-hop radio networks, *i.e.* networks such that collisions can occur at any node in the communication graph (the multiple access channel models a single-hop radio network). Recent developments for the wakeup in multi-hop radio networks can be found in [3,4,5], where the authors consider the locally synchronous model when the nodes know the size n of the network (or a linear upper bound on it), but the topology is unknown. Again, knowing n is critical for the performance of such mechanisms in multi-hop networks.

1.3 Our Result

We consider the wakeup problem on multiple access channel in the weakest model for the deterministic setting: locally synchronous model with unknown n. We propose a new deterministic algorithm that completes the wakeup in time $O(n^4 \log^3 n)$ in the worst case. This is an improvement over the previous $O(n^4 \log^5 n)$ algorithm presented in [13] and it is based on a different approach. Since the bound $O(n^4 \log^3 n)$ depends on a result in Number Theory (Theorem 3 below) which probably is not tight (see our remark at the end of Section 4), actually the bound provided by our new algorithm might be improved.

2 Preliminaries

In this section we recall some well known results in Number Theory that will be used in the analysis of our algorithm. Throughout the paper the ith prime number will be denoted p_i. The prime counting function $\pi(n)$ is the function giving the number of primes $\leq n$.

The Prime Number Theorem gives an asymptotic form for the prime counting function.

Theorem 1 ((Prime Number Theorem)).

$$\pi(n) \sim \frac{n}{\ln n}.$$

(Equivalently, $p_n \sim n \ln n$). □

We have used the asymptotic notation $\sim$ as defined in [15]: in other words, the Prime Number Theorem tells us that the limit of the quotient of the two functions $\pi(n)$ and $n/\ln(n)$, as n approaches to infinity, is 1.

The theta function $\theta(n)$ is defined as follows:

$$\theta(n) = \sum_{i=1}^{\pi(n)} \ln p_i = \ln \left(\prod_{i=1}^{\pi(n)} p_i \right).$$

Chebyshev [15, p. 341] gave the following bound for $\theta(n)$.

Theorem 2 ((Chebyshev's bound on $\theta(n)$)).

$$\theta(n) < 2n \ln 2 \textit{ for all } n \geq 1. \qquad \square$$

Jacobsthal's problem is to estimate, for a given r, the maximum length of a sequence of consecutive integers, each divisible by one of r arbitrarily chosen primes. For references and history on this problem, see [9]. Iwaniec [18] proved the following important result.

Theorem 3 ((Iwaniec, 1978)). *For any positive integer r, let $C(r)$ denote the maximal length of a sequence of consecutive integers each divisible by at least one of r arbitrarily fixed primes. Then*

$$C(r) \in O(r^2 \ln^2 r) \quad (r \to +\infty). \qquad \square$$

We will also use the following well known result.

Theorem 4 ((Chinese Remainder Theorem)). *Let $m_1, \ldots, m_r$ be pairwise relatively prime positive integers. Let $M = m_1 \cdots m_r$ and let $a_1, \ldots, a_r, A$ be integers. Then there is exactly one integer a such that $A \leq a < A + M$ satisfying*

$$a \equiv a_k \pmod{m_k} \quad \forall\, k,\, 1 \leq k \leq r. \qquad \square$$

3 Wakeup in the Locally Synchronous Model with Unknown n

Here we consider the weakest model: there is no global clock available and the size n of the system is not known; each processor knows its own ID number, all ID numbers are distinct.

3.1 The New Upper Bound

Algorithm. FAST_WAKEUP: *For each $j \in \mathbb{N}$, let p_j be the j-th prime number. Each processor i transmits immediately when it is woken up, and successively it transmits exactly p_i time units after its last transmission.*

Theorem 5. *Algorithm* FAST_WAKEUP *wakes up a system of n processors in time $O(n^4 \ln^3 n)$.*

Proof. In order to measure the time necessary to wake up the system, we will often refer our calculation to a global time t to mean the tth time unit after the first processor has woken up. Of course, this time (unknown to the processors) does not have to be confused with the local times. For each integer i with $1 \leq i \leq n$, a_i is the time at which the processor labelled i wakes up ($a_i = +\infty$ if the i-th processor does not wake up during the whole process).

We call W the set of the labels of the processors that wake up in the process (*i.e.* $W = \{i : (1 \leq i \leq n) \wedge (a_i < +\infty)\}$).

Following the algorithm, it is easy to verify that any processor i transmits at a generic time $x \geq a_i$ if and only if $x \equiv a_i \pmod{p_i}$.

Let a_k (with $1 \leq k \leq n$) be a generic time unit at which some processor k wakes up; let m be the maximal label among the processors that are awake at time a_k, that is, $m = \max\{i : (1 \leq i \leq n) \wedge (a_i \leq a_k)\}$. We want to find a time unit $x \geq a_k$ at which processor k transmits and for any $i \in W$, $i \neq k$, $1 \leq i \leq m$ processor i does not transmit. A sufficient condition for this to happen is

$$\begin{cases} x \equiv a_k & \pmod{p_k}, \\ x \not\equiv a_i & \pmod{p_i} \ \ \forall\, i \in W,\, i \neq k,\, 1 \leq i \leq m. \end{cases}$$

Let us set $N = p_1 p_2 ... p_m$. From Theorem 2 it follows that $\sum_{1 \leq i \leq m} \ln p_i < p_m \ln 4$. Therefore, $N < 4^{p_m}$ and, since by Theorem 1 $p_m \in O(m \ln m)$, we must have $N \in 4^{O(m \ln m)}$. By Theorem 4 there exists at least a y in the range $1 \leq y \leq N$ satisfying

$$y \equiv a_k - a_i \pmod{p_i} \ \ \forall\, i \in W,\, 1 \leq i \leq m; \tag{1}$$

in particular for $i = k$ we obtain $y \equiv a_k - a_k \equiv 0 \pmod{p_k}$.

Consider the integer $y' = y/p_k$; by theorem 3 there exists a non-negative $\ell' \in O(m^2 \ln^2 m)$ such that $y' + \ell'$ is an integer, say r, which is co-prime to N. We can then write

$$y/p_k + \ell' = r, \text{ with } \ell' \in O(m^2 \ln^2 m) \text{ and } \gcd(r;\, N) = 1. \tag{2}$$

Let $\ell = p_k \ell'$, multiplying the two members of (2) by p_k gives $y + \ell = p_k r$. Since $p_k \leq p_m$, p_k lies in $O(m \ln m)$. Therefore,

$$\ell \in O(m \ln m) \cdot O(m^2 \ln^2 m) = O(m^3 \ln^3 m).$$

Since y satisfies all the congruences of system (1), from the equality $y + \ell = p_k r$ we can deduce that

$$\forall\, i \in W \text{ with } 1 \leq i \leq m,\ a_k - a_i + \ell \equiv p_k r \pmod{p_i},$$

i.e.

$$\forall\, i \in W \text{ with } 1 \leq i \leq m,\ a_k + \ell \equiv a_i + p_k r \pmod{p_i}. \tag{3}$$

Recalling that $\gcd(r;\, N) = 1$ (and then p_i does not divide r for any i with $1 \leq i \leq m$), the congruences in (3) imply

$$\begin{cases} a_k + \ell \equiv a_k & \pmod{p_k}, \\ a_k + \ell \not\equiv a_i & \pmod{p_i} \ \ \forall\, i \in W,\, i \neq k,\, 1 \leq i \leq m. \end{cases}$$

This means that, once processor k wakes up at time a_k, it transmits successfully within ℓ time units (where $0 \leq \ell \in O(m^3 \ln^3 m) \subseteq O(n^3 \ln^3 n)$), unless in the meantime another processor has been woken up after it. Since at most n processors can be woken up, it follows that a successful transmission will be definitely produced within $n \cdot O(n^3 \ln^3 n) = O(n^4 \ln^3 n)$ time units after any processor wakes up. $\square$

4 Conclusion and Further Research

In this paper, we have showed an upper bound of $O(n^4 \ln^3 n)$ on the time for waking up deterministically a locally synchronous multiple access channel of n processors, when n is unknown to the processors. The best lower bound for this problem is $(1+\epsilon)n$, for some $\epsilon > 0$, proved in [13].

Hence, the main question left open by the present paper is to narrow the (still huge) gap between upper and lower bound. To this aim it is useful to remark that our $O(n^4 \ln^3 n)$ upper bound given in Theorem 5 depends essentially on the $O(r^2 \ln^2 r)$ upper bound of Theorem 3. Such an upper bound probably is not optimal: in [18] Iwaniec cites the open question raised by Jacobsthal about the validity of the stronger upper bound $O(r^2)$. If this latter bound holds, the arguments used in the proof of Theorem 5 would lead to the result $O(n^4 \ln n)$ instead of $O(n^4 \ln^3 n)$.

Acknowledgements

We wish to thank Jean-Louis Nicolas for his guidance on the literature about Jacobstahl's problem.

References

1. R. Bar-Yehuda, O. Goldreich and A. Itai, *On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap between Determinism and Randomization.* Journal of Computer and System Sciences Vol. 45, 1992, pp. 104-126.
2. D. Bertsekas and R. Gallager, *Data Networks.* Prentice Hall, 1991.
3. M.Chrobak, L.Gasieniec and D.Kowalski *The Wake-Up problem in multi-hop radio networks*, In Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 992-1000.
4. B. Chlebus, L. Gasieniec, D. Kowalski and T. Radzik *On the Wake-up Problem in Radio Networks*, to appear in the proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005).
5. B. Chlebus and D. Kowalski *A better wake-up in radio networks*, Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC 2004), pp. 266-274.
6. A. De Bonis and U. Vaccaro, *Efficient Constructions of Generalized Superimposed Codes with Applications to Group Testing and Conflict Resolution in Multiple Access Channels*, Theoretical Computer Science, vol. 306, Issue 1-3, pp. 223-243, 2003.
7. G. De Marco and A. Pelc, *Faster broadcasting in unknown radio networks,* Information Processing Letters Vol 79, 2001, pp. 53-56.
8. D.Z. Du and F.K. Hwang, *Combinatorial Group Testing and its Applications*, World Scientific, 2000.
9. P. Erdös, *On the Integers Relatively Prime to n and on a Number-Theoretic Function Considered by Jacobsthal,* Math. Scand. 11 (1962), pp. 163-170.
10. P. Erdös, P. Frankl and Z. Füredi, *Families of Finite Sets in Which no Set Is Covered by the Union of r Others,* Israel J. Math., vol. 51, 1985, pp. 75-89.

11. R. Gallager, *A Perspective on Multiaccess Channels,* IEEE Trans. on Information Theory 31 (1985), pp. 124-142.
12. J. Goodman, A.G. Greenberg, N. Madras, P. March, *On the stability of Ethernet*, 17th ACM symposium on Theory of computing, STOC, 1985, pp. 379-387.
13. L. Gąsieniec, A. Pelc and D. Peleg, *The Wakeup Problem in Synchronous Broadcast Systems.* SIAM J. Discrete Math., Vol 14, No. 2, 2001, pp. 207-222.
14. A.G. Greenberg and S. Winograd, *A Lower Bound on the Time Needed in the Worst Case to Resolve Conflicts Deterministically in Multiple Access Channels.* J. ACM, 32 (1985), pp. 598-596.
15. G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers*, 4th ed. Oxford, England: Clarendon Press, 1975.
16. J. Hastad, T. Leighton, B. Rogoff, *Analisys of Backoff Protocols for Multiple Access Channels,* 19th ACM symposium on Theory of computing, STOC, 1987, pp. 241-253.
17. P. Indyk, *Deterministic Superimposed Coding with Application to Pattern Matching,* 38th Symposium on Foundations of Computer Science, FOCS, 1997, pp. 127-136.
18. H. Iwaniec, *On the problem of Jacobstahl*, Demonstratio Mathematica, **11** (1978), 225-231.
19. T. Jurdziński and Stachowiak *Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks*, ISAAC 2002, LNCS 2518, pp.535-549, 2002.
20. J. Komlós and A.G. Greenberg, *An Asymptotically Optimal Nonadaptive Algorithm for Conflict Resolution in Multiple-Access Channels*, IEEE Trans. on Information Theory, vol. 31, (2), (1985), pp. 302 - 306
21. W.H. Kautz and R.C. Singleton, *Nonrandom binary superimposed codes,* IEEE Trans. Inform. Theory, vol. 10, 1964, pp. 363 - 377.
22. A.Tannenbaum, *Computer Networks,* Prentice-Hall, Englewood Cliffs, NJ, 1981.

Weighted Coloring: Further Complexity and Approximability Results

Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos

LAMSADE, CNRS and Université Paris-Dauphine,
Place du Maréchal de Lattre de Tassigny,
75775, Paris Cedex 16, France
{escoffier, monnot, paschos}@lamsade.dauphine.fr

Abstract. Given a vertex-weighted graph $G = (V, E; w)$, $w(v) \geq 0$ for any $v \in V$, we consider a weighted version of the coloring problem which consists in finding a partition $\mathcal{S} = (S_1, \dots, S_k)$ of the vertex set V of G into stable sets and minimizing $\sum_{i=1}^{k} w(S_i)$ where the weight of S is defined as $\max\{w(v) : v \in S\}$. In this paper, we keep on with the investigation of the complexity and the approximability of this problem by mainly answering one of the questions raised by D. J. Guan and X. Zhu ("A Coloring Problem for Weighted Graphs", Inf. Process. Lett. 61(2):77-81 1997).

Keywords: Approximation algorithm; **NP**-complete problems; weighted coloring; interval graphs; line graph of bipartite graphs; partial k-tree.

1 Introduction

A k-coloring of $G = (V, E)$ is a partition $\mathcal{S} = (S_1, \dots, S_k)$ of the vertex set V of G into stable sets S_i. In the usual coloring problem, the objective is to determine a vertex coloring minimizing k. A natural generalization of this problem is obtained by assigning a strictly positive integer weight $w(v)$ for any vertex $v \in V$, and defining the weight of stable set S of G as $w(S) = \max\{w(v) : v \in S\}$. Then, the objective is to determine a vertex coloring $\mathcal{S} = (S_1, \dots, S_p)$ of G minimizing the quantity $\sum_{i=1}^{p} w(S_i)$. This problem has several applications. For instance in [8] this problem is motivated by a problem of transmission of real-time messages in a metropolitan network or a problem related to dynamic storage allocation. It is interesting to notice that in these two applications, graphs of a special kind are used: the interval graphs. Others examples of applications in different contexts can be found in [2,5].

In this paper, we continue the investigation of the complexity and the approximability of the Weighted Coloring problem by mainly answering one of the questions raised by Guan and Zhu [8].

Given an instance $I = (G, w)$, W denotes the set of different weights used in the instance, i.e., $W = \{w(v) : v \in V\}$, $opt(I)$ denotes the weight of an optimal weighted coloring of I and $\chi(I)$ denotes the minimum number of colors used among the optimal weighted colorings of I. As indicated in [4,8], this number may be very high, even in trees, although it is always bounded above by $\Delta(G)+1$.

Moreover, we can obtain a bound related to the quantities $\chi(G)$ and $|W|$ where $\chi(G)$ is the chromatic number of G and $|W|$ is the number of different weights used in I. Precisely, in [4] it is proved that we have $\chi(I) \leq 1+|W|(\chi(G)-1)$ and that this bound is tight for any $q = \chi(G)$ and $r = |W|$ for a family of instances $I_{q,r}$ where the graphs are chordal.

Let's recall some standard definitions about some class of graphs, see [9] for more details. A graph G is *chordal* iff any cycle C of G of length at least 4 has a chord. There are several characterizations of chordal graphs; one of them uses the notion of *perfect elimination order* (peo. in short) of $G = (V, E)$. An order $v_1, \ldots, v_n$ of the vertex set V is a peo. if the neighborhood of v_i in the subgraph induced by $\{v_1, \ldots, v_i\}$ is a clique; a graph is chordal iff it has a peo. When a graph G has a peo., we can easily find an optimal coloring of G by applying the greedy algorithm (take recursively a vertex not colored yet and color it with the smallest color) following the peo. of the vertices. A graph $G = (V, E)$ is a *split* graph iff one can partition V into V_1, V_2 such that V_1 is a stable and V_2 is a clique (there may be some edges linking V_1 to V_2). A split graph is in particular a chordal graph since it has a peo. A graph is a *k-tree* iff G has a peo. $v_1, \ldots, v_n$ such that $v_1, \ldots, v_k$ is a clique and the neighborhood of v_i in the subgraph induced by $\{v_1, \ldots, v_i\}$ has a size k for $i > k$ (a 1-tree is also called a tree). A graph G is a *partial k-tree* if G is a partial graph of a k-tree. A graph G is a comparability graph iff G has a direct orientation $\overrightarrow{G} = (V, \boldsymbol{E})$ such that $\overrightarrow{G}$ is acyclic (there is no circuit) and $\overrightarrow{G}$ is transitive (that is if $(x, y) \in \boldsymbol{E}$ and $(y, z) \in \boldsymbol{E}$, then $(x, z) \in \boldsymbol{E}$). A direct acyclic transitive orientation of G is also called a poset. As previously, we can prove that the coloring problem is polynomial in posets. A graph G is a co-comparability graph iff the complement of G is a comparability graph. A graph $G = (V, E)$ is an interval graph if it is the intersection graph of a family of open intervals. For graph-theoretical terms not defined here, the reader is referred to [1].

This paper is organized as follows. In Sect. 2, we answer a question raised in [8] by proving that Weighted Coloring is (strongly) **NP**-hard in interval graphs. In Sect. 3, we deal with polynomial approximation of Weighted Coloring, providing mainly approximation algorithms for graphs colorable with a few number of colors and for partial k-trees.

2 Interval Graphs

The interval graphs are a kind of graphs very used in practice, in particular when dealing with scheduling problems. A well known characterization of interval graphs is the following: G is an interval graph iff G is a chordal graph and G is a co-comparability graph. Although the coloring problem is polynomial in interval graphs (since an interval graph is a chordal graph), in [4], it is proved that the Weighted Coloring problem is strongly **NP**-hard in split graphs and thus strongly **NP**-hard in chordal graphs, since the split graphs is a subclass of chordal graphs. Moreover, it is shown in [5] that the Weighted Coloring problem is polynomial in complements of interval graphs. In this section, we prove that the Weighted Coloring problem is strongly **NP**-hard in interval graphs.

Theorem 1. *Weighted Coloring is strongly* **NP***-hard in interval graphs. Moreover, the problem of finding* $\chi(G,w)$ *is also* **NP***-hard in interval graphs.*

Proof. We reduce the Circular Arc Coloring problem to our problem. A circular arc graph is the intersection graph of arcs of a circle. Garey et al. [6] proved that the Circular Arc Coloring problem, i.e., the problem of finding a minimum size coloring in circular arc graphs, is **NP**-complete. Let G be the intersection graph of the n-tuple of circular arcs $\mathcal{A} = (A_j)_{j\in\{1,\cdots,n\}}$, and let $k \in \{1,\cdots,n\}$. Assume, wlog., that the intervals A_j are open. We transform this instance of Circular Arc Coloring in an instance $I' = (G', w)$ of Weighted Coloring as follows. Let a be any point on the circle, and $J_0 = \{j : a \in A_j\}$. For simplicity, assume wlog. that point a belongs to some arcs and that $J_0 = \{1,\cdots,j_0\}$, for some $j_0 \geq 1$. For any $j \leq j_0$, we split interval $A_j = (c_j, d_j)$ in $A'_j = (c_j, a)$ and $A''_j = (a, d_j)$. For $j > j_0$, we define $A'_j = A_j$. Set $\mathcal{A}'$ be the $(n + j_0)-$tuple of intervals $(A'_j)_{j\in\{1,\cdots,n\}}$ and $(A''_j)_{j\leq j_0}$. Let G' be the intersection graphs of $\mathcal{A}'$. We set the weights w of G' in the following way: $w(v'_j) = w(v''_j) = 2k(j_0 + 1 - j)$ if $j \leq j_0$ and $w(v'_j) = 1$ for $j > j_0$. The description of instance $I' = (G', w')$ of Weighted Coloring is now complete.

Note that $\{v_j, j \leq j_0\}$ is a clique in G. We can suppose $k \geq j_0$ (otherwise G is trivially not k-colorable). We claim that $\chi(G) \leq k$ if and only if $opt(G') \leq kj_0(j_0 + 1) + k - j_0 = B$.

Suppose that $\mathcal{S} = (S_1, \cdots, S_k)$ is a coloring of G. Then, set $\mathcal{S}' = (S'_1, \cdots, S'_k)$ where $S'_i = S_i \setminus \{v_j : j \leq j_0\} \cup \{v'_j, v''_j : v_j \in S_i, j \leq j_0\}$. One can easily see that $\mathcal{S}'$ is a coloring of G'. Furthermore, we have $opt(G') \leq w(\mathcal{S}') = 2k\sum_{j=1}^{j_0} j + (k - j_0) = B$.

Reciprocally, let $\mathcal{S}' = (S'_1, \cdots, S'_l)$ be a coloring of G' with $opt(I') = w(\mathcal{S}') \leq B$. Assume that $w(S_i) \geq w(S_j)$ for any $j \geq i$. Note that $\{v'_1, v''_1\} \in S'_1$, otherwise $opt(I') = w(\mathcal{S}') \geq 2kj_0 + 2kj_0 + 2k\sum_{j=1}^{j_0-2} j = kj_0(j_0 + 1) + 2kj_0 > B$. With an analogous argument, we can show that $\{v'_j, v''_j\} \in S'_j$ for any $j \leq j_0$. Consequently, $w(\mathcal{S}') = kj_0(j_0 + 1) + (l - j_0)$, and then $l \leq k$. Set $S_i = S'_i \setminus \{v'_i, v''_i\} \cup v_i$ for $i \leq j_0$ and $S_i = S'_i$ for $i > j_0$. $\mathcal{S} = (S_1; \cdots, S_l)$ is a l-coloring of G, and $\chi(G) \leq l \leq k$.

The **NP**-hardness of computing $\chi(I')$ follows easily from the previous proof. Indeed, if G' is not k-colorable, then obviously $\chi(I') > k$. Otherwise, let $\mathcal{S}'$ be an optimal coloring. Since $\chi(G) \leq k$ iff $opt(I') \leq B$, $w(\mathcal{S}') \leq B$ and, as we have seen above, $w(S') = kj_0(j_0+1) + |\mathcal{S}'| - j_0 = B + |\mathcal{S}'| - k$. Hence, $\chi(I') \leq |\mathcal{S}'| \leq k$. □

Using Theorem 1 and the characterization of interval graphs, we deduce that the Weighted Coloring problem is strongly **NP**-hard in co-comparability graphs.

3 Approximation Results

3.1 *k*-Colorable Graphs

We study in this section, the approximability of the Weighted Coloring problem in natural classes of graphs colorable with a few number of colors. We first focus ourselves on subfamilies of k-colorable graphs where the minimum coloring problem is polynomial. Our objective is to prove the following theorem.

Theorem 2. *Let $\mathcal{G}$ be a class of k-colorable graphs, where a k-coloring is computable in polynomial time. Then, in any $G \in \mathcal{G}$, Weighted Coloring is approximable within ratio $k^3/(3k^2-3k+1)$.*

Proof. Consider some graph $G=(V,E) \in \mathcal{G}$ of order n, and assume that any $v_i \in V$ has weight $w_i = w(v_i)$. Suppose that $w_1 \geq w_2 \geq \cdots \geq w_n$. Consider an optimal weighted coloring $\mathcal{S}^* = (S_1^*, \cdots, S_l^*)$, with $w(S_1^*) \geq \cdots \geq w(S_l^*)$ and denote by i_k^*, the index of the heaviest vertex in color S_k^* (hence, $w(S_k^*) = w_{i_k}$), by V_i the set of vertices $\{v_1, \cdots, v_i\}$ (hence, $V_n = V$) and by $G[V']$ the subgraph of G induced by $V' \subseteq V$.

We compute several colorings of G and choose as final solution the best one among the colorings computed. We first compute a k-coloring $\mathcal{S}^0$ of G. Clearly:

$$w\left(\mathcal{S}^0\right) \leq kw_1 = kw\left(S_1^*\right) \tag{1}$$

Then, for $j = 2, \cdots, n+1$, we do the following:

- if $G[V_{j-1}]$ is bipartite then:
  - consider the best weighted 2-coloring (S_1^j, S_2^j) among the 2-colorings of $G[V_{j-1}]$ (S_2^j may be empty);
  - color the remaining vertices $v_j, \cdots, v_n$ with k colors $(S_3^j, S_4^j, \cdots, S_{k+2}^j)$, thus obtaining a coloring $\mathcal{S}^j = (S_1^j, S_2^j, \cdots, S_{k+2}^j)$ of G.

Note that the first step is easily polynomially computable (merge optimally the unique 2-colorings of any connected component).

Consider now the iterations where $j = i_2^*$ and $j = i_3^*$. For $j = i_2^*$, V_{j-1} is an independent set; hence, $S_1^j = V_{j-1}$. We get in this case:

$$w(\mathcal{S}^j) \leq w_1 + kw_j = w\left(S_1^*\right) + kw\left(S_2^*\right) \tag{2}$$

On the other hand, for $j = i_3^*$, $G[V_{j-1}]$ is bipartite; hence, $w(S_1^j) + w(S_2^j) \leq w(S_1^*) + w(S_2^*)$. In this case:

$$w(\mathcal{S}^j) \leq w\left(S_1^*\right) + w\left(S_2^*\right) + kw\left(S_3^*\right) \tag{3}$$

Recall that the algorithm returns the best coloring among those computed. Note also that if the number l of colors in $\mathcal{S}^*$ is smaller than 2, then this algorithm computes an optimal coloring. Combination of equations (1), (2) and (3) with coefficients $(k-1)^2/k^3$, $k(k-1)/k^3$ and $k^2/k^3 = 1/k$, respectively, concludes that the output coloring $\mathcal{S}$ is such that: $w(\mathcal{S}) \leq (k^3/(3k^2-3k+1))w(\mathcal{S}^*)$ and the result follows. $\square$

Note that this improves a $(4-3/k)$-approximation algorithm given in [15] (see the note on related works at the end of the paper), for $k \leq 10$.

Corollary 1. *Weighted Coloring is approximable within ratio* $27/19 < 1.42$ *in polynomially 3-colorable graphs.*

It is well known that the coloring problem is polynomial in planar triangle-free graphs ([10]) and that the chromatic number in these graphs is bounded by 3. Moreover, it is proved in [3] that on the one hand the Weighted Coloring problem is strongly **NP**-hard and, on the other hand, the Weighted Coloring problem cannot be approximated with performance ratio better than $\frac{7}{6} - \varepsilon$ for any $\varepsilon > 0$ unless **P**$\neq$**NP**, in the planar triangle-free graphs, even if the maximum degree is bounded by 4. Using Theorem 2, we obtain:

Corollary 2. *Weighted Coloring is* $27/19$*-approximable in planar triangle-free graphs.*

As another corollary of Theorem 2, Weighted Coloring is approximable within ratio 64/37 in polynomially 4-colorable graphs. On the other hand, note that minimum coloring is not $(4/3 - \varepsilon)$-approximable in planar graphs, that these graphs are polynomially 4-colorable and that the Weighted Coloring problem is a generalization of the coloring problem. Putting all this together, we obtain:

Theorem 3. *Weighted Coloring is approximable within ratio* $64/37 < 1.73$ *in planar graphs, but it is not* $(4/3 - \varepsilon)$*-approximable in these graphs.*

Note that the result of Theorem 2 can be applied also to line graphs of bipartite graphs of degree at most Δ. A weighted coloring on $I = (L(G), w)$ where $L(G)$ is the line graph of G can be viewed as a weighted edge-coloring on (G, w). In fact, in [4], it is shown that the Weighted Coloring problem is strongly **NP**-complete in line graphs of regular bipartite graphs of degree Δ and that the Weighted Coloring problem is not $(\frac{2^\Delta}{2^\Delta - 1} - \varepsilon)$ approximable unless **P**=**NP**, for any $\Delta \geq 3$. Besides, the **NP**-completeness also holds for the line graphs of complete bipartite graphs. More recently, in [3] this lower bound is tightened up to $\frac{7}{6}$ when $\Delta = 3$. Furthermore, it is proved that this bound is the best possible since in [3] is also provided a 7/6-approximation algorithm.

Now, we generalize the technique used in Theorem 2 to get an approximation algorithm in line graphs of bipartite graphs of degree at most Δ, for any fixed $\Delta \geq 3$, since using König's theorem ([12]) we know that the coloring problem is polynomial in line-graphs of bipartite graphs. More precisely, we can show the following theorem:

Theorem 4. *For any* $\Delta \geq 3$*, Weighted Coloring in line graphs of bipartite graphs of maximum degree at most* Δ *is approximable within approximation ratio* ρ_Δ*, where* $\rho_1 = \rho_2 = 1$ *and:*

$$\rho_\Delta = \frac{\Delta}{\sum_{j=1}^{\Delta} \prod_{l=j}^{\Delta-1} (1 - \rho_l/\Delta)}$$

Proof. Let $\rho_1 = \rho_2 = 1$ and ρ_j be a ratio guaranteed by some polynomial algorithm for $3 \leq j \leq \Delta - 1$ on line graphs of bipartite graphs of maximum degree at most j. Then, on an instance $I = (G, w)$ where $G = L(H)$ and H is a bipartite graph of maximum degree Δ. Let us consider an optimal solution $\mathcal{S}^* = (S_1^*, \cdots, S_l^*)$. We have $l \geq \Delta$ by construction. As previously, set i_k^* the index of the heaviest vertex of S_k^*. Then, as in the proof of Theorem 2, we can compute three colorings of value at most $\Delta w(S_1^*)$, $w(S_1^*) + \Delta w(S_2^*)$ and $w(S_1^*)+w(S_2^*)+\Delta w(S_3^*)$, respectively. Moreover, for any $k \in \{4, \cdots, \Delta\}$, consider the graph $G[V_{i_k-1}]$ induced by the vertices $V_{i_k-1} = \{v_1, \cdots, v_{i_k-1}\}$. This graph is a line graph of a bipartite graph of maximum degree at most $k-1$ (since it is $k-1$ colorable). Hence, by applying our approximation algorithm with ratio ρ_{k-1} on $I_{i_k-1} = (G[V_{i_k-1}], w)$, we can get a coloring of I_{i_k-1} of value at most $\rho_{k-1} opt(I_{i_k-1}) \leq \rho_{k-1} \sum_{j=1}^{k-1} w(S_j^*)$ and then, a coloring of I of value at most $\rho_{k-1} \sum_{j=1}^{k-1} w(S_j^*) + \Delta \times w(S_k^*)$. If we take the best coloring $\mathcal{S}$ among all these colorings, we get:

$$\begin{aligned}
w(\mathcal{S}) &\leq \Delta \times w(S_1^*) \\
w(\mathcal{S}) &\leq \rho_1 w(S_1^*) + \Delta \times w(S_2^*) \\
&\cdots \\
w(\mathcal{S}) &\leq \rho_{\Delta-1} \sum_{m=1}^{\Delta-1} w(S_m^*) + \Delta \times w(S_\Delta^*)
\end{aligned}$$

Let $\beta = \displaystyle\sum_{j=1}^{\Delta} \prod_{l=j}^{\Delta-1} (1-\rho_l/\Delta)$ (note that by convention $\prod_{l=\Delta}^{\Delta-1}(1-\rho_l/\Delta) = 1$). Then, take the convex combination of these Δ inequalities with coefficients $\alpha_1, \cdots, \alpha_\Delta$, where $\alpha_i = \frac{\prod_{l=i}^{\Delta-1}(1-\rho_l/\Delta)}{\beta}$. We have $\alpha_i \in [0,1]$, $\alpha_\Delta = \frac{1}{\beta}$ and $\sum_{i=1}^{\Delta} \alpha_i = 1$.

We get $w(\mathcal{S}) \leq \rho_\Delta (\sum_{m=1}^{\Delta} w(S_m^*)) \leq \rho_\Delta w(S^*)$ where $\rho_\Delta = \Delta/\beta$. Indeed, the contribution of weight $w(S_i^*)$ in the convex combination is $\Delta\alpha_i + \sum_{j=i+1}^{\Delta} \alpha_j \times \rho_{j-1}$. If we denote $A_i = \sum_{j=i+1}^{\Delta} \alpha_j \times \rho_{j-1}$, then we can easily prove that $A_i = \Delta(\alpha_\Delta - \alpha_i)$. □

The following table gives the approximate value of ratio ρ_Δ for some Δ.

Δ	4	5	10	50	100	200	400	700	1000
ρ_Δ	1.61	1.75	2.16	2.97	3.25	3.51	3.73	3.89	3.99

Note that this improves a $(4 - 3/\Delta)$-approximation algorithm given in [15] (see the note on related works at the end of the paper), for $\Delta \leq 1025$.

We end this section by improving the lower bound obtained in [4] for the Weighted Coloring problem in line graphs of regular bipartite graphs of degree Δ.

Theorem 5. *For any $\Delta \geq 3$, $\varepsilon > 0$, the Weighted Coloring problem is not $(1+\frac{2}{\Delta} - \frac{2}{\Delta+1} - \varepsilon)$ approximable unless* ***P=NP****, in line graphs of regular bipartite graphs of degree Δ.*

Proof. For simplicity, we consider the edge model, i.e., we study the Weighted Edge-Coloring problem in regular bipartite graphs of degree Δ. We prove the following result by induction: *it is* ***NP****-complete to distinguish between* $opt(I) \leq \frac{\Delta(\Delta+1)}{2}$ *and* $opt(I) \geq \frac{\Delta(\Delta+1)+2}{2}$ *in regular bipartite graphs of degree* Δ*, where the weights used are in* $W = \{1, \ldots, \Delta\}$ *and there exists at least one vertex of degree* Δ *whose incident edges have different weights (i.e. are weighted by 1, 2, ... , and* Δ*).*

For $\Delta = 3$ the result is proved in [3]. Assume that the result holds for $\Delta = k - 1$ and let us prove the result for $\Delta' = k$.

Let $I = (G, w)$ be an instance with $\Delta = k-1$ (in other words, $G = (L, R; E)$ is a regular bipartite graphs of degree $k - 1$). We construct an instance $I' = (G', w')$ of the case $\Delta' = k$ as follows: we duplicate G as $G_1 = (L_1, R_1; E_1)$ and $G_2 = (L_2, R_2; E_2)$ and we add two matchings M_1 and M_2 between G_1 and G_2 such that M_i links the vertices of L_i to the vertices of R_{3-i}. The weights of the edges of G' are assigned as follows: if $e \in E_1 \cup E_2$, then $w'(e) = w(e)$; if $e \in M_1 \cup M_2$, then $w'(e) = \Delta'$.

It is clear that the instance $I' = (G', w')$ verifies the required hypothesis. We claim that $opt(I) \leq \frac{\Delta(\Delta+1)}{2}$ iff $opt(I') \leq \frac{\Delta' (\Delta' +1)}{2}$.

If $opt(I) \leq \frac{\Delta(\Delta+1)}{2}$, then by duplicating this edge coloring to G_i and by adding a new color to $M_1 \cup M_2$, we obtain an edge coloring of G' and $opt(I') \leq opt(I) + \Delta' \leq \frac{\Delta' (\Delta' +1)}{2}$.

Conversely, assume that $opt(I') \leq \frac{\Delta' (\Delta' +1)}{2}$. Then the edges of $M_1 \cup M_2$ have the same color. Otherwise, $opt(I') \geq \Delta' + \Delta' + \sum_{i=1}^{\Delta' -2} i > \frac{\Delta' (\Delta' +1)}{2}$ since in I' there are Δ' edges $e_1, \ldots, e_{\Delta'}$ with $w'(e_i) = i$ adjacent to the same vertex. Thus, the restriction of this solution to G_1 is an edge coloring verifying $opt(I) \leq opt(I') - \Delta'$. □

3.2 Partial *k*-Trees

A k-tree is a graph that can be reduced to a clique of size k by deleting iteratively some vertices the neighborhood of which is a clique of size k. A partial k-tree is a subgraph of a k-tree. There are several characterizations of partial k-trees. One of them is the following: G is a partial k-tree iff G is a subgraph of a chordal graph G' with a clique number equal to $k + 1$ (i.e., $\omega(G') = k + 1$). k-trees are (polynomially) $k + 1$-colorable, and we can get a ρ_{k+1}-approximation, but we can improve this result.

Let us define the List Coloring problem, where we want to answer the following question: given a graph $G = (V, E)$ with, for any $v \in V$, a set $L(v)$ of admissible colors, does there exist a (proper) coloring of G with colors from $L(V) = \cup_{v \in V} L(v)$ such that any vertex v is colored with a color from $L(v)$? The complexity of List Coloring has been studied in [13,11].

Theorem 6. *If* $\mathcal{G}$ *is a class of* t*-colorable graphs (where* t *is a constant) where List Coloring is polynomial, then Weighted Coloring admits a PTAS in* $\mathcal{G}$*.*

Proof. Let $\mathcal{G}$ such a class of graphs, a graph $G \in \mathcal{G}$ and $\varepsilon > 0$. Let $k = \lceil (t-1)(1 + \frac{1}{\varepsilon}) \rceil$.

Consider the following algorithm. For any $k' \leq k$:

- consider any k'-tuple $(x_1, \cdots, x_k\ ) \in W^k$;
- find a k'-coloring $(S_1, \cdots, S_k\ )$ of G such that $w(S_i) \leq x_i, i = 1, \cdots, k'$, if such a coloring exists;
- output the best coloring among those found in the previous step.

To achieve the second step, we use the fact that List Coloring is polynomial in $\mathcal{G}$. Indeed, given $(x_1, \cdots, x_k\ ) \in W^k$, we can define an instance of List Coloring on G: v_i can be colored with color S_j for all $j \in \{1, \cdots, k'\}$ such that $w(v_i) \leq x_j$. One can easily see that a coloring is valid for this instance of List Coloring, if and only if this coloring is such that $w(S_i) \leq x_i, i = 1, \cdots, k'$.

We claim that the solution computed by this algorithm is $1+\varepsilon$-approximate, for any $\varepsilon > 0$.

Indeed, consider an optimal solution $\mathcal{S}^* = (S_1^*, \cdots, S_l^*)$, with $w(S_1^*) \geq \cdots \geq w(S_l^*)$. If $l \leq k$, then we found, by our exhaustive search, a coloring $(S_1, \cdots, S_l)$ such that $w(S_i) \leq w(S_i^*)$ for all i, hence an optimal solution.

If $l > k$, then consider the k-tuple $(w_1, \cdots, w_k)$ where $w_i = w(S_i^*)$ for $i \leq k+1-t$ and $w_i = w(S_{k+1-t}^*)$ for $i \geq k+1-t$. If we consider the $k-t$ colors $S_i^*, i = 1, \cdots, k-t$, and any t-coloring $S_j, j = k+1-t, \cdots, k$ of the remaining vertices (the graph is polynomially t-colorable), then $w(S_j) \leq w(S_{k+1-t}^*)$ for $j = k+1-t, \cdots, k$. So, the algorithm finds a coloring for this particular tuple $(w_1, \cdots, w_k)$. Consequently, the solution $\mathcal{S}$ given by the algorithm is such that:

$$\begin{aligned}\frac{w(\mathcal{S})}{w(\mathcal{S}^*)} &\leq \frac{\sum_{i=1}^{k+1-t} w(S_i^*) + (t-1)w(S_{k+1-t}^*)}{\sum_{i=1}^{l} w(S_i^*)} \\ &\leq 1 + \frac{(t-1)w(S_{k+1-t}^*)}{\sum_{i=1}^{k+1-t} w(S_i^*)} \leq 1 + \frac{t-1}{k+1-t} \leq 1+\varepsilon\end{aligned}$$

Since we use less than $|W^{k+1}| \leq n^{k+1}$ times the algorithm for List Coloring as a subroutine, our algorithm is polynomial, hence we get the expected result. □

Since List Coloring is polynomial in partial k-trees ([11]), then we have the following corollary:

Corollary 3. *Weighted Coloring admits a PTAS in partial k-trees (hence, in particular, in trees).*

Although we have proposed an approximation scheme in partial k-trees, the complexity of the Weighted Coloring problem remains open for these graphs.

We now focus ourselves on the case where the input graph is a chain, or a collection of chains. Guan and Zhu proposed in [8] a polynomial time algorithm with complexity $O(n^4)$ to solve (optimally) Weighted Coloring in chains (as a particular case of a more general result). We can improve this result in the following way:

Theorem 7. *Weighted Coloring is polynomially solvable in $O(n|W|) \leq O(n^2)$ in chains.*

Proof. Consider a graph G which is a set of k disjoint chains $C_1, \cdots, C_k$. Note that $\Delta(G) \leq 2$ hence any optimal Weighted Coloring has at most 3 colors. As we have seen previously, the best 2-coloring is easily computable (in $O(n)$). To compute an optimal 3-coloring, we compute, for any $w \in W$, the smallest number $n_w \leq w$ for which there exists a 3-coloring (S_1, S_2, S_3) with $w(S_1) = w_{\max}$, $w(S_2) \leq w$ and $w(S_3) \leq n_w$ ($n_w = \infty$ if such a coloring does not exist). Remark first that every vertex v with weight $w(v) > w$ must receive color 1. Consider now two consecutive such vertices v_i, v_j in a chain C_l. If there exists an odd number of vertices between v_i and v_j, we can color these vertices with colors 1 and 2. Otherwise, one must use 3 colors, and one can do it by coloring with color 3 only the lightest vertex v_{ij} between v_i and v_j. Hence, n_w is the heaviest among these vertices v_{ij}. Given $w \in W$, we can find n_w (and the corresponding coloring) in $O(n)$, hence the result follows. □

3.3 Note on Related Works

During the time between submission and the current version, we have learnt that some results obtained here also appear in papers [14,15]. In these papers, the authors deal with the same problem that they call maximum coloring. In [14], the authors proved that this problem is **NP**-complete in interval graphs and gave constant polynomial approximation. However, the proof used for the **NP**-completeness is completely different. In [15], the authors proposed a PTAS for the case of trees. Moreover, they also provide a $(4 - 3/k)$-approximation algorithm in hereditary classes of k-colorable graphs where the usual coloring problem is polynomial. In particular, this gives a $(4 - 3/\Delta)$-approximation algorithm for line graphs of bipartite graphs with maximum degree Δ.

References

1. C. Berge. Graphs and Hypergraphs. *North Holland, Amsterdam*, 1973.
2. M. Boudhar and G. Finke. Scheduling on a batch machine with job compatibilities. *Special issue ORBEL-14: Emerging challenges in operations research (Mons, 2000). Belg. J. Oper. Res. Statist. Comput. Sci.*, 40(1-2):69–80, 2000.
3. D. de Werra, M. Demange, B. Escoffier, J. Monnot and V.Th. Paschos. Weighted coloring on planar, bipartite and split graphs: complexity and improved approximation. *Proc. ISAAC'04*, LNCS 3341, 896–907, 2004.
4. D. de Werra, M. Demange, J. Monnot and V.Th. Paschos. Weighted node coloring: when stable sets are expensive. *Proc. WG'02*, LNCS 2573:114–125, 2002.
5. G. Finke, V. Jost, M. Queyranne, A. Seb and X. Zhu. Batch Processing With Interval Graph Compatibilities Between Tasks. *Cahiers du laboratoire Leibniz*, 108, 2004, (available at http://www-leibniz.imag.fr/NEWLEIBNIZ/LesCahiers/index.xhtml.)
6. M.R. Garey, D.S. Johnson, G.L. Miller and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Dicrete Methods*, 1:216–222, 1980.
7. M. R. Garey and D. S. Johnson. Computers and intractability. a guide to the theory of NP-completeness. *CA, Freeman*, 1979.

8. D. J. Guan and X. Zhu. A Coloring Problem for Weighted Graphs. *Inf. Process. Lett.*, 61(2):77-81 1997.
9. M. C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. *Academic Press, New York*, 1980.
10. H. Grotzsch, *Ein dreifarbensatz fur dreikreisfreie netze auf der kugel*, Wiss. Z. Martin Luther Univ. Halle-Wittenberg, Math. Naturwiss Reihe **8** (1959), 109-120.
11. K. Jansen and P. Scheffler: Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75:135-155, 1997.
12. D. König. ber graphen und iher anwendung auf determinantentheorie und mengenlehre. *Math. Ann.*, 77:453-465, 1916.
13. J. Kratochvl and Z. Tuza. Algorithmic complexity of list colorings. *Discrete Applied Mathematics*, 50(3): 297-302, 1994.
14. S. V. Pemmaraju, R. Raman and K. R. Varadarajan. Buffer minimization using max-coloring. *SODA* 562-571, 2004.
15. S. V. Pemmaraju and R. Raman. Approximation algorithms for the max-coloring. *ICALP*, (to appear) 2005.

Quantum Algorithms for a Set of Group Theoretic Problems[⋆]

Stephen A. Fenner and Yong Zhang

University of South Carolina Columbia,
SC 29208, USA
{fenner, zhang29}@cse.sc.edu

Abstract. This work introduces two decision problems, STABILIZER$_D$ and ORBIT COSET$_D$, and gives quantum reductions from them to the problem ORBIT SUPERPOSITION (Friedl et al., 2003), as well as quantum reductions to them from two group theoretic problems GROUP INTERSECTION and DOUBLE COSET MEMBERSHIP. Based on these reductions, efficient quantum algorithms are obtained for GROUP INTERSECTION and DOUBLE COSET MEMBERSHIP in the setting of black-box groups. Specifically, for solvable groups, this gives efficient quantum algorithms for GROUP INTERSECTION if one of the underlying solvable groups has a smoothly solvable commutator subgroup, and for DOUBLE COSET MEMBERSHIP if one of the underlying solvable groups is smoothly solvable. Finally, it is shown that GROUP INTERSECTION and DOUBLE COSET MEMBERSHIP are in the complexity class **SZK**.

1 Introduction

This paper makes progress in finding connections between quantum computation and computational group theory. We give results about quantum algorithms and reductions for group theoretic problems, concentrating mostly on solvable groups.

Many problems that have quantum algorithms exponentially faster than the best known classical algorithms turn out to be special cases of the HIDDEN SUBGROUP problem for abelian groups, which can be solved using the Quantum Fourier Transform [1,2]. The non-abelian HIDDEN SUBGROUP problem remains a very interesting open problem since it has as its special case the GRAPH ISOMORPHISM problem. Recently Friedl et al. [3] made progress on the non-abelian case by introducing STABILIZER and ORBIT COSET, both of which generalize HIDDEN SUBGROUP, and showing that they can be solved efficiently on quantum computers for a family of smoothly solvable groups. They introduced in the same paper the problem ORBIT SUPERPOSITION as a useful tool. In this paper we further investigate the relationship among STABILIZER, ORBIT COSET, and

⋆ This work was supported in part by the National Security Agency (NSA) and Advanced Research and Development Activity (ARDA) under Army Research Office (ARO) contract number DAAD 190210048.

Orbit Superposition. We introduce two problems Stabilizer$_D$ and Orbit Coset$_D$, which are the decision versions of Stabilizer and Orbit Coset. We show that in bounded error quantum polynomial time Stabilizer$_D$ reduces to Orbit Superposition over solvable groups and Orbit Coset$_D$ reduces to Orbit Superposition over any finite groups.

These reductions to Orbit Superposition suggest that the difficulty in Stabilizer$_D$ and Orbit Coset$_D$ resides in the construction of uniform quantum superpositions over orbits (in a group action). This is in general not a surprise. Very often, solving a problem with a quantum algorithm can be reduced to preparing the right quantum superposition. For example, if one can prepare a uniform superposition over all graphs isomorphic to a given graph, then one can solve the Graph Isomorphism problem easily via a simple swap test [4,5]. What makes orbit superpositions interesting in our case, however, is their unexpected utility for solving a variety of different problems that may at first seem unrelated, including not only Stabilizer$_D$ and Orbit Coset$_D$ but also Group Intersection and Double Coset Membership, described below.[1]

Our results on Stabilizer$_D$ and Orbit Coset$_D$ help us to obtain efficient quantum algorithms for two well studied problems in computational group theory, Group Intersection and Double Coset Membership. No efficient classical algorithms are known for these two problems. Watrous [6] first used quantum computers to help solve problems in computational group theory. He constructed efficient quantum algorithms for several problems on solvable groups, such as Order Verification and Group Membership. Based on an algorithm of Beals and Babai [7], Ivanyos, Magniez, and Santha [8] obtained efficient quantum algorithms for Order Verification as well as several other group theoretic problems. Watrous asked in [6] whether there are efficient quantum algorithms for problems such as Group Intersection and Coset Intersection. Here we study Group Intersection and Double Coset Membership where Double Coset Membership generalizes Coset Intersection as well as Group Membership and Group Factorization. We show that for solvable groups, there are efficient quantum algorithms for Group Intersection and Double Coset Membership under certain conditions. We obtain these results by showing that these two problems reduce to Stabilizer$_D$ and Orbit Coset$_D$, respectively. Our results also imply that for *abelian* groups, Group Intersection and Double Coset Membership are in the complexity class **BQP**. Combined with Fortnow and Rogers' result [9] that any problem in **BQP** is low for the counting class **PP**, we obtain an alternative proof that they are low for the class **PP**. Arvind and Vinodchandran first proved this result [10].

Finally, motivated by a similar result in Aharonov and Ta-Shma [5], we show that Group Intersection and Double Coset Membership have honest-verifier zero knowledge proof systems, and thus are in **SZK**. This is an improvement of Babai's result [11] that Group Intersection and Double Coset

[1] Watrous's order-finding algorithm for solvable groups [6] also works by explicitly constructing a certain orbit superposition.

Membership are in **AM** ∩ co**AM**. While Watrous [12] showed that Group Nonmembership is in the complexity class **QMA**, another implication of our results is that Group Nonmembership is in **SZK**.

Our results and other known reducibility relationships between these and other various group theoretic problems are summarized in Figure 1.

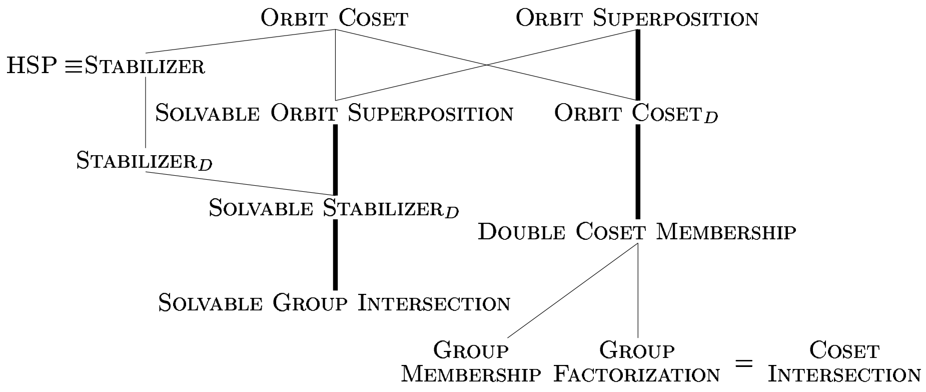


Fig. 1. Known reducibilities between various group theoretic problems. Thick lines represent nontrivial reducibilities shown in the current work.

2 Preliminaries

Background on general group theory and quantum computation can be found in the standard textbooks [13,14].

2.1 The Black-Box Group Model

All of the group theoretic problems discussed in this paper will be studied in the model of black-box groups. This model was first introduced by Babai and Szemerédi [15] as a general framework for studying algorithmic problems for finite groups. It has been extensively studied (see [6]). Here we will use descriptions similar to those in [10].

We fix the alphabet $\Sigma = \{0, 1\}$. A *group family* is a countable sequence $\mathcal{B} = \{B_m\}_{m\geq 1}$ of finite groups B_m, such that there exist polynomials p and q satisfying the following conditions. For each $m \geq 1$, elements of B_m are encoded as strings (not necessarily unique) in $\Sigma^{p(m)}$. The group operations (inverse, product and identity testing) of B_m are performed at unit cost by black-boxes (or group oracles). The order of B_m is computable in time bounded by $q(m)$, for each m. We refer to the groups B_m of a group family and their subgroups (presented by generator sets) as *black-box groups*. Common examples of black-box groups are $\{S_n\}_{n\geq 1}$ where S_n is the permutation group on n elements, and $\{GL_n(q)\}_{n\geq 1}$ where $GL_n(q)$ is the group of $n \times n$ invertible matrices over the finite field F_q. Depending on whether the group elements are uniquely encoded, we have the *unique encoding model* and *non-unique encoding model*, the latter of which enables us to deal with factor groups [15]. In the non-unique encoding model an additional group oracle has to be provided to test if two strings represent the same group element. Our results will apply only to the unique encoding model. In one of our proofs, however, we will use the non-unique encoding model to handle factor groups. For how to implement group oracles in the form of quantum circuits, please see [6].

Definition 1 ([10]). *Let $\mathcal{B} = \{B_m\}_{m\geq 1}$ be a group family. Let e denote the identity element of each B_m. Let $k \geq 2$ be any integer. Let $\langle S\rangle$ denote the group generated by a set S of elements of B_m. Below, g and h denote elements, and S_1 and S_2 subsets, of B_m.*

$$\begin{aligned}
\textsc{Group Intersection} &:= \{(0^m, S_1, S_2) \mid \langle S_1\rangle \cap \langle S_2\rangle = \langle e\rangle\},\\
\textsc{Multiple Group Intersection} &:= \{(0^m, S_1, \ldots, S_k) \mid \langle S_1\rangle \cap \ldots \cap \langle S_k\rangle = \langle e\rangle\},\\
\textsc{Group Membership} &:= \{(0^m, S_1, g) \mid g \in \langle S_1\rangle\},\\
\textsc{Group Factorization} &:= \{(0^m, S_1, S_2, g) \mid g \in \langle S_1\rangle\langle S_2\rangle\},\\
\textsc{Coset Intersection} &:= \{(0^m, S_1, S_2, g) \mid \langle S_1\rangle g \cap \langle S_2\rangle \neq \emptyset\},\\
\textsc{Double Coset Membership} &:= \{(0^m, S_1, S_2, g, h) \mid g \in \langle S_1\rangle h\langle S_2\rangle\}.
\end{aligned}$$

Multiple Group Intersection is a generalized version of Group Intersection. Also, it is easily seen that Double Coset Membership generalizes Group Membership, Group Factorization, and Coset Intersection. Therefore in this paper we will focus on Double Coset Membership. All our results about Double Coset Membership will also apply to Group Membership, Group Factorization, and Coset Intersection. (Actually, Coset Intersection and Group Factorization are easily seen to be the same problem.)

2.2 Solvable Groups

The *commutator subgroup* G' of a group G is the subgroup generated by elements $g^{-1}h^{-1}gh$ for all $g, h \in G$. We define $G^{(n)}$ such that

$$\begin{aligned}
G^{(0)} &= G,\\
G^{(n)} &= (G^{(n-1)})', \text{ for } n \geq 1.
\end{aligned}$$

G is *solvable* if $G^{(n)}$ is the trivial group $\{e\}$ for some n. We call $G = G^{(0)} \triangleright G^{(1)} \triangleright \cdots \triangleright G^{(n)} = \{e\}$ the *derived series* of G, of length n. Note that all the factor groups $G^{(i)}/G^{(i+1)}$ are abelian. There is a randomized procedure that computes the derived series of a given group G [16].

The term *smoothly solvable* is first introduced in [3]. We say that a family of abelian groups is *smoothly abelian* if each group in the family can be expressed as the direct product of a subgroup with bounded exponent and a subgroup of polylogarithmic size in the order of the group. A family of solvable groups is *smoothly solvable* if the length of each derived series is bounded by a constant and the family of all factor groups $G^{(i)}/G^{(i+1)}$ is smoothly abelian.

In designing efficient quantum algorithms for computing the order of a solvable group (Order Verification), Watrous [6] obtained as a byproduct a method to construct approximately uniform quantum superpositions over elements of a given solvable group.

Theorem 1 ([6]). *In the model of black-box groups with unique encoding, there is a quantum algorithm operating as follows (relative to an arbitrary group oracle). Given generators $g_1, \ldots, g_m$ such that $G = \langle g_1, \ldots, g_m \rangle$ is solvable, the algorithm outputs the order of G with probability of error bounded by ϵ in time polynomial in $mn + \log(1/\epsilon)$ (where n is the length of the strings representing the generators). Moreover, the algorithm produces a quantum state ρ that approximates the state $|G\rangle = |G|^{-1/2} \sum_{g \in G} |g\rangle$ with accuracy ϵ (in the trace norm metric).*

2.3 A Note on Quantum Reductions

In Sections 3 and 4 we describe quantum reductions to various problems. Quantum algorithms for these problems often require several identical copies of a quantum state or unitary gate to work to a desired accuracy. Therefore, we will implicitly assume that our reductions may be repeated t times, where t is some appropriate parameter polynomial in the input size and the logarithm of the desired error bound.

3 Stabilizer$_D$ and Orbit Coset$_D$

Friedl et al. [3] introduced several problems which are closely related to Hidden Subgroup. In particular, they introduced Stabilizer, Hidden Translation, Orbit Coset, and Orbit Superposition. Stabilizer generalizes Hidden Subgroup. In fact, the only difference between Stabilizer and Hidden Subgroup is that in the definition of Stabilizer the function f can be a *quantum function* that maps group elements to mutually orthogonal quantum states with unit norm. Orbit Coset generalizes Stabilizer and Hidden Translation. Orbit Superposition is a relevant problem, which is also of independent interest. The superpositions Watrous constructed in Theorem 1 can be considered as an instance of Orbit Superposition.

We would like to further characterize the relationship of these problems. First we define and study the decision versions of Stabilizer and Orbit Coset, denoted as Stabilizer$_D$ and Orbit Coset$_D$. The original definitions of Stabilizer and Orbit Coset concerns about finding generating sets of certain stabilizer subgroups. In the decision version, we simplify the problems by only asking whether the stabilizer subgroups are trivial. We also give the definition of the problem Orbit Superposition.

Let G be a finite group. Let Γ be a set of mutually orthogonal quantum states. Let $\alpha : G \times \Gamma \to \Gamma$ be a group action of G on Γ, i.e., for every $x \in G$ the function $\alpha_x : |\phi\rangle \to |\alpha(x, |\phi\rangle)\rangle$ is a permutation over Γ and the map h from G to the symmetric group over Γ defined by $h(x) = \alpha_x$ is a homomorphism. We use the notation $|x \cdot \phi\rangle$ instead of $|\alpha(x, |\phi\rangle)\rangle$, when α is clear from the context. We let $G(|\phi\rangle)$ denote the set $\{|x \cdot \phi\rangle : x \in G\}$, and we let $G_{|\phi\rangle}$ denote the stabilizer subgroup of $|\phi\rangle$ in G, i.e., $\{x \in G : |x \cdot \phi\rangle = |\phi\rangle\}$. Given any positive integer t, let α^t denote the group action of G on $\Gamma^t = \{|\phi\rangle^{\otimes t} : |\phi\rangle \in \Gamma\}$ defined by $\alpha^t(x, |\phi\rangle^{\otimes t}) = |x \cdot \phi\rangle^{\otimes t}$. We need α^t because the input superpositions cannot be cloned in general.

Definition 2. *Let G be a finite group and Γ be a set of pairwise orthogonal quantum states. Fix the group action $\alpha : G \times \Gamma \to \Gamma$.*

- *Given generators for G and a quantum state $|\phi\rangle \in \Gamma$,* Stabilizer$_D$ *is to check if the subgroup $G_{|\phi\rangle}$ is the trivial subgroup $\{e\}$.*
- *Given generators for G and two quantum states $|\phi_0\rangle, |\phi_1\rangle \in \Gamma$,* Orbit Coset$_D$ *is to either reject the input if $G(|\phi_0\rangle) \cap G(|\phi_1\rangle) = \emptyset$ or accept the input if $G(|\phi_0\rangle) = G(|\phi_1\rangle)$.*
- *Given generators for G and a quantum state $|\phi\rangle \in \Gamma$,* Orbit Superposition *is to construct the uniform superposition*

$$|G \cdot \phi\rangle = \frac{1}{\sqrt{|G(|\phi\rangle)|}} \sum_{|\phi'\rangle \in G(|\phi\rangle)} |\phi'\rangle.$$

Next we show that the difficulty of Stabilizer$_D$ and Orbit Coset$_D$ may reside in constructions of certain uniform quantum superpositions, which can be achieved by the problem Orbit Superposition.

We will use the following result which is easily derivable from Theorem 7 in Ivanyos, Magniez, and Santha [8]:

Theorem 2 ([8]). *Assume that G is a solvable black-box group given by generators with not necessarily unique encoding. Suppose that N is a normal subgroup given as a hidden subgroup of G via the function f. Then the order of the factor group G/N can be computed by quantum algorithms in time polynomial in n, where n is the input size.*

Please note that we can apply Theorem 2 to factor groups since it uses the non-unique encoding black-box groups model.

Theorem 3. *Over solvable groups,* Stabilizer$_D$ *reduces to* Orbit Superposition *in bounded-error quantum polynomial time.*

Proof. Let the solvable group G and quantum state $|\phi\rangle$ be the input for the problem Stabilizer$_D$. We can find in classical polynomial time generators for each element in the derived series of G [16], namely, $\{e\} = G_1 \lhd \cdots \lhd G_n = G$. For $1 \leq i \leq n$ let $S_i = (G_i)_{|\phi\rangle}$, the stabilizer of $|\phi\rangle$ in G_i. By Theorem 1 we can compute the orders of $G_1, \ldots, G_n$ and thus the order of G_{i+1}/G_i for any $1 \leq i < n$. We will proceed in steps. Suppose that before step $i+1$, we know that $S_i = \{e\}$. We want to find out if $S_{i+1} = \{e\}$ in the $(i+1)$st step. Since $G_i \lhd G_{i+1}$, by the Second Isomorphism Theorem, $G_iS_{i+1}/G_i \cong S_{i+1}$. Consider the factor group G_{i+1}/G_i, we will define a function f such that f is constant on G_iS_{i+1}/G_i and distinct on left cosets of G_iS_{i+1}/G_i in G_{i+1}/G_i. Then by Theorem 2 we can compute the order of the factor group G_{i+1}/G_i over G_iS_{i+1}/G_i. The group oracle needed in the non-unique encoding model to test if two strings s_1 and s_2 represent the same group elements can be implemented using the quantum algorithm for Group Membership, namely, testing if $s_1^{-1}s_2$ is a member of G_i. The order of this group is equal to the order of G_{i+1}/G_i if and only if S_{i+1} is trivial.

Here is how we define the function f. Using G_i and $|\phi\rangle$ as the input for Orbit Superposition, we can construct the uniform superposition $|G_i \cdot \phi\rangle$. Let Γ be the set $\{|gG_i \cdot \phi\rangle | g \in G_{i+1}\}$. We define $f : G_{i+1}/G_i \to \Gamma$ be such that $f(gG_i) = |gG_i \cdot \phi\rangle$. What is left is to verify that f hides the subgroup G_iS_{i+1}/G_i in the group G_{i+1}/G_i. For any $g \in G_iS_{i+1}$, it is straightforward to see that $|gG_i \cdot \phi\rangle = |G_i \cdot \phi\rangle$. If g_1 and g_2 are in the same left coset of G_iS_{i+1}, then $g_1 = g_2g$ for some $g \in G_iS_{i+1}$ and thus $|g_1G_i \cdot \phi\rangle = |g_2G_i \cdot \phi\rangle$. If g_1 and g_2 are not in the same left coset of G_iS_{i+1}, we will show that $|g_1G_i\phi\rangle$ and $|g_2G_i\phi\rangle$ are orthogonal quantum states. Suppose there exists $x_1, x_2 \in G_i$ such that $|g_1x_1 \cdot \phi\rangle = |g_2x_2 \cdot \phi\rangle$, then $x_1^{-1}g_1^{-1}g_2x_2 \in S_{i+1}$. But $x_1^{-1}g_1^{-1}g_2x_2 = x_1^{-1}x_2'g_1^{-1}g_2$ for some $x_2' \in G_i$. Thus $g_1^{-1}g_2 \in G_iS_{i+1}$. This contradicts the assumption that g_1 and g_2 are not in the same coset of G_iS_{i+1}.

We need to repeat the above procedure at most $\Theta(\log |G|)$ times. For each step the running time is polynomial in $\log |G| + \log(1/\epsilon)$, for error bound ϵ. So the total running time is still polynomial in the input size.

We can also easily reduce Orbit Coset$_D$ to Orbit Superposition in quantum polynomial time. In this reduction, we don't require the underlying groups to be solvable. The proof uses similar techniques that Watrous [12] and Buhrman et al. [4] used to differentiate two quantum states.

Theorem 4. Orbit Coset$_D$ *reduces to* Orbit Superposition *in bounded-error quantum polynomial time.*

Proof. Let the finite group G and two quantum states $|\phi_1\rangle$, $|\phi_2\rangle$ be the inputs of Orbit Coset$_D$. Notice that the orbit coset of $|\phi_1\rangle$ and $|\phi_2\rangle$ are either identical or disjoint, which implies the two quantum states $|G \cdot \phi_1\rangle$ and $|G \cdot \phi_2\rangle$ are either identical or orthogonal. We may then tell which is the case using a version of the swap test of Buhrman et al. [4].

4 Quantum Algorithms for Group Intersection and Double Coset Membership

In this section we use results in the previous section to make progress in finding quantum algorithms for Group Intersection and Double Coset Membership.

We will need the following results which are easily derivable from Friedl et al. [3].

Theorem 5 ([3]). *Let G be a finite solvable group having a smoothly solvable commutator subgroup. Let α be a group action of G.* Stabilizer$_D$ *can be solved in G for α^t in quantum time poly*$(\log |G|)\log(1/\epsilon)$ *with error ϵ when $t = (\log^{\Omega(1)} |G|)\log(1/\epsilon)$,*

Theorem 6 ([3]). *Let G be a smoothly solvable group and let α be a group action of G. When $t = (\log^{\Omega(1)} |G|)\log(1/\epsilon)$,* Orbit Coset$_D$ *can be solved in G for α^t in quantum time poly*$(\log |G|)\log(1/\epsilon)$ *with error ϵ.*

First we show that with the help of certain uniform quantum superpositions over group elements, Group Intersection can be reduced to Stabilizer$_D$.

Theorem 7. Group Intersection *reduces to* Stabilizer$_D$ *in bounded-error quantum polynomial time if one of the underlying groups is solvable.*

Proof. Given an input $(0^m, S_1, S_2)$ for Group Intersection, without loss of generality, suppose that $G = \langle S_1 \rangle$ is an arbitrary finite group and $H = \langle S_2 \rangle$ is solvable. By Theorem 1 we can construct an approximately uniform superposition $|H\rangle = |H|^{-1/2} \sum_{h \in H} |h\rangle$. For any $g \in G$, let $|gH\rangle$ denote the uniform superposition over left coset gH, i.e., $|gH\rangle = |H|^{-1/2} \sum_{h \in gH} |h\rangle$. Let $\Gamma = \{|gH\rangle | g \in G\}$. Note that the quantum states in Γ are (approximately) pairwise orthogonal. Define the group action $\alpha : G \times \Gamma \to \Gamma$ to be that for every $g \in G$ and every $|\phi\rangle \in \Gamma$, $\alpha(g, |\phi\rangle) = |g\phi\rangle$. Then the intersection of G and H is exactly the subgroup of G that stabilizes the quantum state $|H\rangle$.

Corollary 1. Group Intersection *over solvable groups can be solved within error ϵ by a quantum algorithm that runs in time polynomial in $m + \log(1/\epsilon)$, where m is the size of the input, provided one of the underlying solvable groups has a smoothly solvable commutator subgroup.*

Proof. Follows directly from Theorems 7 and 5.

We observe that a similar reduction to Stabilizer$_D$ holds for Multiple Group Intersection.

Proposition 1. Multiple Group Intersection *reduces to* Stabilizer$_D$ *in bounded-error quantum polynomial time if all but one of the underlying groups are solvable.*

Proof. Without loss of generality, we illustrate the proof for the case $k = 3$. Suppose we have three input groups G, H, and K, where H and K are solvable. We let Γ be the set $\{|gH\rangle \otimes |gK\rangle | g \in G\}$ and the group action $\alpha : G \times \Gamma \to \Gamma$ be that for every $g \in G$ and every $|\phi\rangle \otimes |\psi\rangle \in \Gamma$, $\alpha(g, |\phi\rangle \otimes |\psi\rangle) = |g\phi\rangle \otimes |g\psi\rangle$. Then $G \cap H \cap K$ is the stabilizer subgroup of G that stabilizes the quantum state $|H\rangle \otimes |K\rangle$.

It is not clear if a similar reduction to Stabilizer$_D$ exists for Double Coset Membership. However, Double Coset Membership can be nicely put into the framework of Orbit Coset$_D$.

Theorem 8. Double Coset Membership *over solvable groups reduces to* Orbit Coset$_D$ *in bounded-error quantum polynomial time.*

Proof. Given input for Double Coset Membership S_1, S_2, g and h, where $G = \langle S_1 \rangle$ and $H = \langle S_2 \rangle$ are solvable groups, we construct the input for Orbit Coset$_D$ as follows. Let $\Gamma = \{|xH\rangle | x \in \langle S_1, S_2, g, h\rangle\}$. Define group action $\alpha : G \times \Gamma \to \Gamma$ to be $\alpha(x, |\phi\rangle) = |x\phi\rangle$ for any $x \in G$ and $|\phi\rangle \in \Gamma$. Let two input quantum states $|\phi_0\rangle$ and $|\phi_1\rangle$ be $|gH\rangle$ and $|hH\rangle$, which can be constructed using Theorem 1. It is not hard to check that $G(|\phi_0\rangle) = G(|\phi_1\rangle)$ if and only if $g \in GhH$.

Corollary 2. Double Coset Membership *over solvable groups can be solved within error ϵ by a quantum algorithm that runs in time polynomial in $m + \log(1/\epsilon)$, where m is the size of the input, provided one of the underlying groups is smoothly solvable.*

Proof. Given input for Double Coset Membership S_1, S_2, g and h, suppose that $G = \langle S_1 \rangle$ is smoothly solvable and $H = \langle S_2 \rangle$ is solvable. Let $S_1, |gH\rangle, |hH\rangle$ be the input for Orbit Coset$_D$, the result follows from Theorem 6. If instead H is the one which is smoothly solvable, then we modify the input by swapping S_1 and S_2 and using g^{-1}, h^{-1} to replace g, h. Note that this modification will not change the final answer.

5 Statistical Zero Knowledge

A recent paper by Aharonov and Ta-Shma [5] proposed a new way to generate certain quantum states using Adiabatic quantum methods. In particular, they introduced the problem Circuit Quantum Sampling (CQS) and its connection to the complexity class Statistical Zero Knowledge (**SZK**). Informally speaking, CQS is to generate quantum states corresponding to classical probability distributions obtained from some classical circuits. Although CQS and Orbit Superposition are different problems, they bear a certain level of resemblance. Both problems are concerned about generation of non-trivial quantum states. In their paper they showed that any language in **SZK** can be reduced to a family of instances of CQS. Based on Theorem 3 and Theorem 4, we would like to ask if there are connections between **SZK** and the two group-theoretic problems discussed in section 4.

Our results are that Group Intersection and Double Coset Membership have honest-verifier zero knowledge proofs, and thus are in **SZK**. This is an improvement of Babai's result [11] that these two problems are in **AM**$\cap$co**AM**. One of our proofs shares the same flavor with Goldreich, Micali and Wigderson's proof that Graph Isomorphism is in **SZK** [17].

For standard notions of interactive proof systems and zero knowledge interactive proof systems, see Vadhan's Ph.D. thesis [18]. Here we only use honest-verifier zero knowledge proof systems. Let $\langle P, V\rangle$ be an interactive proof system for an language L. We say that $\langle P, V\rangle$ is *honest-verifier perfect zero knowledge*

(**HVPZK**) if there exists a probabilistic polynomial-time algorithm M (*simulator*) such that for every $x \in L$ the output probability distribution of V (after interacting with P) and M, denoted as $\langle P, V\rangle(x)$ and $M(x)$, are identical. Similarly, we say $\langle P, V\rangle$ is *honest-verifier statistical zero knowledge* (**HVSZK**) if $\langle P, V\rangle(x)$ and $M(x)$ are statistically indistinguishable. It is clear that **HVPZK** $\subseteq$ **HVSZK**. Goldreich, Sahai, and Vadhan showed that **HVSZK** and **SZK** are actually the same class [19]. Some complexity results concerning **SZK** (**HVSZK**) include that **BPP** $\subseteq$ **SZK** $\subseteq$ **AM** $\cap$ co**AM**, and **SZK** is closed under complement, and **SZK** does not contain any **NP**-complete language unless the polynomial hierarchy collapses (see [18]).

The following theorem due to Babai [20] will be used in our proof. Let G be a finite group. Let $g_1, \ldots, g_k \in G$ be a sequence of group elements. A *subproduct* of this sequence is an element of the form $g_1^{e_1} \ldots g_k^{e_k}$, where $e_i \in \{0, 1\}$. We call a sequence $h_1, \ldots, h_k \in G$ *a sequence of* ϵ*-uniform Erdős-Rényi generators* if every element of G is represented in $(2^k/|G|)(1 \pm \epsilon)$ ways as a subproduct of the h_i.

Theorem 9 ([20]). *Let $c, C > 0$ be given constants, and let $\epsilon = N^{-c}$ where N is a given upper bound on the order of the group G. There is a Monte Carlo algorithm which, given any set of generators of G, constructs a sequence of $O(\log N)$ ϵ-uniform Erdős-Rényi generators at a cost of $O((\log N)^5)$ group operations. The probability that the algorithm fails is $\leq N^{-C}$. If the algorithm succeeds, it permits the construction of ϵ-uniform distributed random elements of G at a cost of $O(\log N)$ group operations per random element.*

Basically what Theorem 9 says is that we can randomly sample elements from G and verify the membership of the random sample efficiently. Given a group G and a sequence of $O(\log N)$ ϵ-uniform Erdős-Rényi generators $h_1, \ldots, h_k$ for G, we say that $e_1 \ldots e_k$ where $e_i \in \{0, 1\}$ is a *witness* of $g \in G$ if $g = h_1^{e_1} \ldots h_k^{e_k}$.

Theorem 10. Group Intersection *has an honest-verifier perfect zero knowledge proof system.*

Proof. Given groups G and H, the prover wants to convince the verifier that the intersection of G and H is the trivial group $\{e\}$. The protocol is as follows.

(V1) The verifier randomly selects $x \in G$ and $y \in H$ and computes $z = xy$. He then sends z to the prover.
(P1) The prover sends two elements, denoted as x' and y', to the verifier.
(V2) The verifier verifies if x is equal to x', and y is equal to y'. The verifier stops and rejects if any of the verifications fails. Otherwise, he repeats steps from (V1) to (V2).

If the verifier has completed m iterations of the above steps, then he accepts.

If $G \cap H$ is trivial, z will be uniquely factorized into $x \in G$ and $y \in H$. Therefore the prover can always answer correctly. On the other hand, if the $G \cap H$ is nontrivial, the factorization of z is not unique, thus with probability at least one half the prover will fail to answer correctly. For a honest verifier V, clearly this protocol is perfect zero-knowledge.

We observe that the above zero knowledge proof does not apply to Multiple Group Intersection. If there are more than two input groups, the factorization of z will not be unique even if the intersection of input groups is trivial.

Theorem 11. Double Coset Membership *has a honest-prover statistical zero knowledge proof system.*

Proof (sketch). Given groups G, H and elements g, h, the prover wants to convince the verifier that $g = xhy$ for some $x \in G$ and $y \in H$. Fix a sufficiently small $\epsilon > 0$. The protocol is as follows.

(V0) The verifier computes ϵ-uniform Erdős-Rényi generators $g_1, \ldots, g_m$ and $h_1, \ldots, h_n$ for G and H. The verifier sends the generators to the prover.
(P1) The prover selects random elements $x \in G$ and $y \in H$ and computes $z = xgy$. He then sends z to the verifier .
(V1) The verifier chooses at random $\alpha \in_R \{0, 1\}$, and sends α to the prover.
(P2) If $\alpha = 0$, then the prover sends x and y to the verifier, together with witnesses that $x \in G$ and $y \in H$. If $\alpha = 1$, then the prover sends over two other elements, denoted as x' and y', together with witnesses that $x' \in G$ and $y' \in H$.
(V2) If $\alpha = 0$, then the verifier verifies that x and y are indeed elements of G and H and $z = xgy$. If $\alpha = 1$, then the verifier verifies that x' and y' are indeed elements of G and H and $z = x'hy'$. The verifier stops and rejects if any of the verifications fails. Otherwise, he repeats steps from (P1) to (V2).

If the verifier has completed m iterations of the above steps, then he accepts.

It is easily seen that the above protocol is an interactive proof system for Double Coset Membership. Note that z is in the double coset GhH if and only if g is in the double coset GhH. If $g \notin GhH$, then with probability at least a half the prover will fail to convince the verifier. If $g \in GhH$, let $g = ahb$ for some $a \in G$ and $b \in H$. Then it is clear that $x' = xa$ and $y' = by$ are also random elements of G and H, thus revealing no information to the verifier. Therefore, to simulate the output of the prover, the simulator simply chooses random elements $x \in G$ and $y \in H$ and outputs z to be xgy if $\alpha = 0$ and xhy if $\alpha = 1$ in the step P1. Given sufficiently small ϵ, the two probability distribution are easily seen to be statistically indistinguishable.

6 Future Research

A key component in our proofs is to construct uniform quantum superpositions over elements of a group, which is addressed by the problem Orbit Superposition. Watrous [6] showed how to construct such superpositions over elements of a solvable group. We would like to find new ways to construct such superpositions over a larger class of non-abelian groups. Aharonov and Ta-Shma [5] used adiabatic quantum computation to construct certain quantum superpositions such as the superposition over all perfect matchings in a given bipartite graph. An interesting question is whether adiabatic quantum computation can help to construct superpositions over group elements.

Besides the decision versions, we can also define the order versions of STABILIZER and ORBIT COSET, where we only care about the order of the stabilizer subgroups. In fact, the procedure described in the proof of Theorem 3 is also a reduction from the order version of STABILIZER to ORBIT SUPERPOSITION. An interesting question is to further characterize the relationship among the decision versions, the order versions, and the original versions of STABILIZER and ORBIT COSET.

Acknowledgments

We would like to thank George McNulty, Frédéric Magniez, John Watrous, Variyam Vinodchandran, Derek Robinson, Scott Aaronson, and Dorit Aharonov for many useful discussions.

References

1. Mosca, M.: Quantum Computer Algorithms. PhD thesis, University of Oxford (1999)
2. Jozsa, R.: Quantum factoring, discrete algorithm and the hidden subgroup problem (2000) Manuscript.
3. Friedl, K., Ivanyos, G., Magniez, F., Santha, M., Sen, P.: Hidden translation and orbit coset in quantum computing. In: Proceedings of the 35th ACM Symposium on the Theory of Computing. (2003) 1–9
4. Buhrman, H., Cleve, R., Watrous, J., de Wolf, R.: Quantum fingerprinting. Physical Review Letters **87** (2001) 167902
5. Aharonov, D., Ta-Shma, A.: Adiabatic quantum state generation and statistical zero knowledge. In: Proceedings of the 35th ACM Symposium on the Theory of Computing. (2003) 20–29
6. Watrous, J.: Quantum algorithms for solvable groups. In: Proceedings of the 33rd ACM Symposium on the Theory of Computing. (2001) 60–67
7. Beals, R., Babai, L.: Las Vegas algorithms for matrix groups. In: Proceedings of the 34th IEEE Symposium on Foundations of Computer Science. (1993) 427–436
8. Ivanyos, G., Magniez, F., Santha, M.: Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. In: Proceedings of 13th ACM Symposium on Parallelism in Algorithms and Architectures. (2001) 263–270
9. Fortnow, L., Rogers, J.: Complexity limitations on quantum computation. Journal of Computer and System Sciences **59** (1999) 240–252
10. Arvind, V., Vinodchandran, N.V.: Solvable black-box group problems are low for PP. Theoretical Computer Science **180** (1997) 17–45
11. Babai, L.: Bounded round interactive proofs in finite groups. SIAM Journal on Computing **5** (1992) 88–111
12. Watrous, J.: Succinct quantum proofs for properties of finite groups. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science. (2000)
13. Burnside, W.: Theory of Groups of Finite Order. Dover Publications, Inc (1955)

14. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
15. Babai, L., Szemerédi, E.: On the complexity of matrix group problems I. In: Proceedings of the 25th IEEE Symposium on Foundations of Computer Science. (1984) 229–240
16. Babai, L., Cooperman, G., Finkelstein, L., Luks, E., Seress, A.: Fast Monte Carlo algorithms for permutation groups. Journal of Computer and System Sciences **50** (1995) 296–307
17. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM **38** (1991) 691–729
18. Vadhan, S.: A study of statistical zero knowledge proofs. PhD thesis, M.I.T. (1999)
19. Goldreich, O., Sahai, A., Vadhan, S.: Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In: Proceedings of the 30th ACM Symposium on the Theory of Computing. (1998) 399–408
20. Babai, L.: Local expansion of vertex-transitive graphs and random generation in finite graphs. In: Proceedings of the 23rd ACM Symposium on the Theory of Computing. (1991) 164–174

On the Computational Complexity of the $L_{(2,1)}$-Labeling Problem for Regular Graphs[⋆]

Jiří Fiala and Jan Kratochvíl

Institute for Theoretical Computer Science[⋆⋆]
and Department of Applied Mathematics,
Charles University, Prague
{fiala, honza}@kam.mff.cuni.cz

Abstract. An $L_{(2,1)}$-labeling of a graph of span t is an assignment of integer labels from $\{0, 1, \dots, t\}$ to its vertices such that the labels of adjacent vertices differ by at least two, while vertices at distance two are assigned distinct labels.

We show that for all $k \geq 3$, the decision problem whether a k-regular graph admits an $L_{(2,1)}$-labeling of span $k+2$ is NP-complete. This answers an open problem of R. Laskar.

1 Introduction

Motivated by models of channel assignment in wireless communication [7,6], generalized graph coloring and in particular the concept of $L_{(2,1)}$-labeling have drawn significant attention in the graph-theory community in the past decade [1].

Besides the practical aspects, also purely theoretical questions became very intersting. Among other we shall highlight a long-lasting conjecture of Griggs and Yeh that the span of any optimal $L_{(2,1)}$-labeling is upperbounded by $\Delta(G)^2$, where $\Delta(G)$ is the maximum degree of the given graph G [6]. So far this conjecture is still open, though it has been verified for various classes of graphs (e.g., for chordal graphs [11,8] or for graphs of diameter at most two [6]).

We focus our attention on the computational complexity of the decision problem whether a given graph G allows an $L_{(2,1)}$-labeling of span at most λ. If λ is a part of the input, the problem becomes NP-complete by a reduction from the Hamiltonian path problem [6]. If λ is a fixed constant (i.e., the parameter of the problem), the computational complexity was settled in [3], by constructing a polynomial time algorithm for $\lambda \leq 3$ and by showing that the problem is NP-complete otherwise. The core argument of the NP-hardness proof is based on the fact that vertices of high degree may allow only extremal labels (i.e., 0 or λ) of the given spectrum.

In response to this fact, R. Laskar asked at the DIMACS/DIMATIA/Rényi Workshop on Graph Colorings and their Generalizations (Rutgers University,

[⋆] The authors acknowledge support of grant Kontakt 525 Dimacs-Dimatia-Renyi.

[⋆⋆] Supported by the Ministry of Education of the Czech Republic as project 1M0021620808.

2003) what is the computational complexity of the $L_{(2,1)}$-labeling problem when restricted to regular graphs, hoping that the restriction might provide new ideas for a general proof of hardness results on distance constrained labelings. In this note we settle the computational complexity of the $L_{(2,1)}$-labeling problem on regular graphs in the following sense:

Theorem 1. *For every integer $k \geq 3$, it is* NP*-complete to decide whether a k-regular graph admits an $L_{(2,1)}$-labeling of span (at most) $\lambda = k + 2$.*

The result is the best possible in terms of the span, since no k-regular graph (for $k \geq 2$) allows an $L_{(2,1)}$-labeling of span $k + 1$ (see e.g. a paper by Georges and Mauro [5] on labelings of regular graphs). Though our result is not totally unexpected, the reduction (namely the garbage collection) is surprisingly uneasy to design. It utilizes so called multicovers introduced in [10].

The paper is organized as follows: The next section provides necessary definitions and facts used latter. In Section 3 we prepare tools used in the construction and discuss their properties. The main result is then proven in Section 4.

2 Preliminaries

All graphs considered in this paper are finite and simple, i.e., with a finite vertex set and without loops or multiple edges. A graph G is denoted as a pair (V_G, E_G), where V_G stands for a finite set of vertices and E_G is a set of edges, i.e. unordered pairs of vertices. The distance $\mathrm{dist}_G(u, v)$ between two vertices u and v of a graph G is the length (the number of edges) of a shortest path connecting u and v. If two vertices belong to different components, we let their distance be unspecified.

The set of vertices adjacent to a vertex u is called *the neighborhood* of u and it is denoted by $N_G(u)$. The *degree* of a vertex u is the cardinality of its neighborhood, i.e., $\deg(u) = |N_G(u)|$. A graph is called *k-regular* if all its vertices are of degree k.

A vertex labeling by nonnegative integers $f : V_G \to \mathbb{Z}_0^+$ is called an *$L_{(2,1)}$-labeling* of G if $|f(u) - f(v)| \geq 2$ holds for any pair of adjacent vertices u and v, and the labels $f(u)$ and $f(v)$ are distinct whenever $\mathrm{dist}(u, v) = 2$.

The *span* of an $L_{(2,1)}$-labeling is the difference between the largest and the smallest label used. The parameter $\lambda_{(2,1)}(G)$ is the minimum possible span of an $L_{(2,1)}$-labeling of G. Such a labeling will be called *optimal*, and we may assume that it uses labels from the discrete interval $[0, \ldots, \lambda_{(2,1)}(G)]$.

With an optimal labeling f we associate its *symmetric* labeling f', defined by $f'(u) = \lambda_{(2,1)}(G) - f(u)$. Clearly the symmetric labeling is also optimal.

$L_{(2,1)}$-labelings are closely related to graph covers: A *full covering projection* from a graph G to a graph H is a graph homomorphism $h : V_G \to V_H$ such that the neighborhood $N_G(u)$ of any vertex $u \in V_G$ is mapped bijectively on the neighborhood $N_H(h(u))$ of $h(u)$.

Similarly, if the mapping is locally injective, i.e., if $N_G(u)$ is mapped injectively into $N_H(h(u))$, we call h a *partial covering projection.* Obviously every full covering projection is also a partial covering projection.

The relationship between $L_{(2,1)}$-labelings and (partial) covering projections was discussed in [2]:

Proposition 1. *Every $L_{(2,1)}$-labeling of a graph G of span λ corresponds to a partial covering projection $G \to \overline{P_{\lambda+1}}$, and vice versa.*

In particular, $\overline{C_{\lambda+1}} \subset \overline{P_{\lambda+1}}$, hence every partial covering projection to $\overline{C_{\lambda+1}}$ is also an $L_{(2,1)}$-labeling of span at most λ.

Kratochvíl, Proskurowski and Telle [10] gave an explicit construction of a special multicover graph allowing many extensions to full covering projections. We will use it in our gadgets.

Proposition 2 ([10]). *For any regular graph F, there exists a graph H (called a* multicover *of F) with a distinguished vertex $u \in V_H$ such that any locally injective homomorphism $h' : N_H(u) \cup u \to F$ can be extended to a locally bijective homomorphism $h : H \to F$.*

3 Gadgets

3.1 Polarity Gadget

Let $k \geq 3$ be a positive integer. Consider the graph F_p on $k+5$ vertices $v_1, \ldots, v_{k-1}, u_1, \ldots, u_4, x, y$, with edges defined as follows:

$$\begin{aligned} E(F_p) = {} & \{(v_i, v_j) \mid 1 \leq i, j \leq k-1, |i-j| \geq 2\} \\ & \cup \{(v_i, u_j) \mid 1 \leq i \leq k-1, j = 1, 2, 3\} \\ & \cup \{(v_i, u_4) \mid 2 \leq i \leq k-2, \} \\ & \cup \{(u_1, u_2), (u_3, u_4), (u_4, x), (u_4, y)\} \end{aligned}$$

See Fig. 1 for an example of such a graph. Observe also that each vertex except x and y is of degree k.

Lemma 1. *In the graph F_p, the pair of vertices x and y are labeled by 0 and λ (or vice versa) under any $L_{(2,1)}$-labeling f of span $\lambda = k+2$.*

Proof. The edge (u_1, u_2) participates in $k-1$ triangles. If both u_1 and u_2 were labeled by labels different from 0 and λ, then at most $\lambda - 4 < k-1$ labels would remain for $v_1, \ldots, v_{k-1}$, which is insufficient. So without loss of generality we may assume $f(u_1) = 0$, and then u_2 may get only two possible labels: 2 or λ. The latter would, however, exclude all possible choices for u_3.

Now up to a symmetry of labelings we have $f(u_1) = 0$, $f(u_2) = 2$ and for the vertices $v_1, \ldots, v_{k-1}$ remain the labels $4, 5, \ldots, \lambda$. Since F_p restricted onto these vertices is the complement of a path on $k-1$ vertices, only two possible labelings exist: either $f(v_i) = i+3$ or $f(v_i) = \lambda + 1 - i$. In both cases (they are equivalent under an automorphism of F_p) only one possible label remains for the vertex u_3, namely $f(u_3) = 1$. We deduce by similar arguments that $f(u_4) = 3$.

Finally, since u_4 is adjacent to vertices labeled by $1, 5, \ldots, \lambda - 1$, its remaining neighbors x and y must be labeled either one by 0 and the other one by λ as claimed.

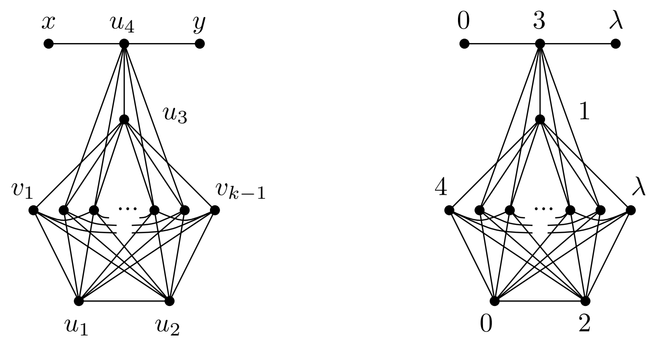


Fig. 1. The polarity gadget F_p and its $L_{(2,1)}$-labeling

3.2 Swallowing Gadget

In our construction we involve multicovers allowing two different $L_{(2,1)}$-labelings as follows. Let H, u be a multicover of the k-regular graph $\overline{C_{k+3}}$. For the swallowing gadget we take two copies H_1, H_2 of the graph H (with the notation that the copy of vertex v in H_i is denoted by v_i, $i = 1, 2$), insert two new vertices x, y and modify the edge set as follows:

$$E(F_s) = (E(H_1) \cup E(H_2) \setminus \{(u_1, v_1), (u_2, v_2)\}) \cup \{(x, v_1), (y, v_2), (u_1, u_2)\}$$

where v is an arbitrary neighbor of u in H. Observe that again all vertices except x and y are of degree k. The construction of the swallowing gadget is illustrated in Fig. 2.

Lemma 2. *The graph F_s allows two $L_{(2,1)}$-labelings f and f', such that $f(v_1) = f'(v_1) = \lambda - 1$, $f(v_2) = f'(v_2) = 1$, while $f(x) = 0$, $f'(x) = 1$ and $f(y) = \lambda$, $f'(y) = \lambda - 1$.*

Proof. We label the vertices of $\overline{C_{k+2}}$ by integers $[0, \lambda]$, in a usual sequential way. For the construction of f we choose the covering projection from H_1 to $[0, \lambda]$

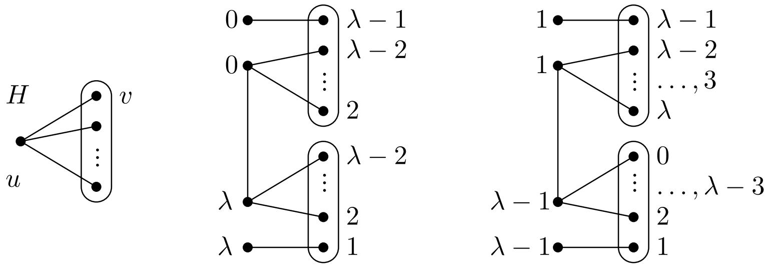


Fig. 2. The swallowing gadget F_s and the two of its $L_{(2,1)}$-labelings

where u is mapped on 0 and v on $\lambda - 1$. The remaining neighbors of u are mapped onto $2, 3, \dots, \lambda - 1$. On H_2 we use the symmetric labeling and get a valid $L_{(2,1)}$-labeling of F_s, since the "central" vertices u_1 and u_2 got labels 0 and λ which are sufficiently separated for the desired $L_{(2,1)}$-labeling f.

Similarly f' can be obtained in a similar way from an $L_{(2,1)}$-labeling of H where u is mapped on 1, the vertex v on $\lambda - 1$ and the remaining neighbors of u are mapped on the set $3, 4, \dots, \lambda - 2, \lambda$.

Both labelings are schematically depicted in Fig. 2.

3.3 Coupling Gadget

Let a be an integer in the range $[1, \lambda - 1]$. Set $T = \{1, 2, \dots, a-1, a+1, \dots, k\}$. We construct the following graph (called the *coupling gadget*) F_c^a on $k^2 + 2k + 1$ vertices $\{v_i^t : i = 0, 1, \dots, \lambda,\ t \in T\} \cup \{u_1, u_2, x, y\}$ by setting the edges as follows:

$$\begin{aligned} E(F_c^a) = {} & \{(v_i^t, v_j^t) \mid 0 \le i, j \le \lambda, |i - j| \ge 2, t \in T\} \\ & \setminus \{(v_t^t, v_\lambda^t), (v_{\lambda-t}^t, v_0^t) \mid t \in T\} \\ & \cup \{(u_1, v_t^t), (u_2, v_{\lambda-t}^t) \mid t \in T\} \\ & \cup \{(u_1, x), (u_2, y)\} \end{aligned}$$

An example of the coupling gadget for a particular choice $\lambda = 5, a = 2$ is depicted in Fig 3. Observe that similarly as above all vertices except x and y are of degree k.

Lemma 3. *The graph F_c^a allows an $L_{(2,1)}$-labeling f of span λ such that $f(x) = a$, $f(y) = \lambda - a$, $f(u_1) = \lambda$ and $f(u_2) = 0$.*

Proof. Set $f(v_i^t) = i$ for $i = 0, 1, \dots, \lambda,\ t \in T$. (See Fig. 3 for an example.) An easy check shows that it is a valid $L_{(2,1)}$-labeling.

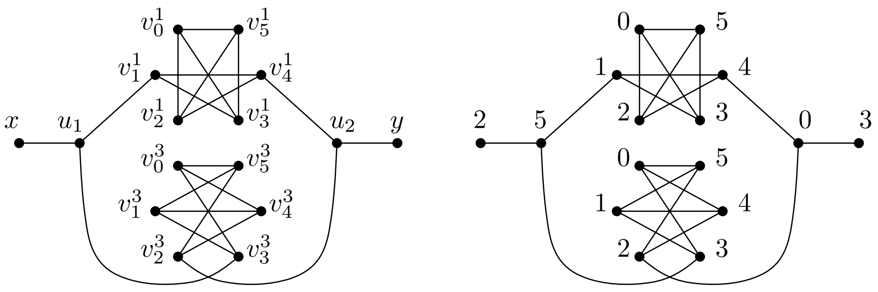


Fig. 3. The coupling gadget F_c^2 for $\lambda = 5$ and its $L_{(2,1)}$-labeling

4 Main Result

Theorem 2. *For every integer $k \geq 3$, it is* NP*-complete to decide whether a k-regular graph admits an $L_{(2,1)}$-labeling of span (at most) $\lambda = k + 2$.*

Proof. The problem is clearly in NP. Moreover, no k-regular graph admits an $L_{(2,1)}$-labeling of span less than $k + 2$ (as long as $k \geq 1$), so we can restrict our attention only to labelings of span exactly $k + 2$.

We prove the theorem by a reduction from the Not-All-Equal 3-Satisfiability problem. The input of this problem is a Boolean formula Φ in conjunctive normal form with exactly three literals in each clause and the question is whether it is NAE-satisfiable, i.e, if a truth assignment exists such that each clause contains at least positively and at least one negatively valued literal. Determining whether Φ is NAE-satisfiable has been shown NP-complete by Schaefer [12] (see also [4, Problem LO3]).

Without loss of generality we may assume that with each clause C the formula Φ contains also its complementary clause C' consisting of the complements of all literals in C. In particular, each variable has then the same number of positive and negative occurrences.

From such a formula Φ we construct a k-regular graph G such that G allows an $L_{(2,1)}$-labeling f of span $k + 2$ if and only if Φ is NAE-satisfiable. The graph G is constructed by local replacements of variables and clauses by variable and clause gadgets described below. (Consult Fig. 4 for details of the construction.)

Variable gadgets: Assume first that $k \neq 4$. For each variable with t positive and t negative occurrences, we insert in G $2t$ copies of the polarity gadget F_p, arranged in a circular manner, i.e., the vertex y_i of the i-th gadget will be identified with the vertex x_{i+1} of the consequent gadget. (The last and the first gadgets are joined accordingly as well.)

The vertices x_i with odd indices will represent positive occurrences of the associated variable, while even indices will be used as negated occurrences of the variable. For $k \geq 5$, we conclude the construction of each variable gadget by inserting $t(k + 1)$ new vertices v_i^s, $i = 1, \dots, k + 1$, $s = 1, \dots, t$, and t triples of coupling gadgets F_c^1, F_c^3 and F_c^3 linked by the following edges:

– (v_i^s, v_j^s) if $|i - j| \geq 2$, i.e., each $(k + 1)$-tuple with the same upper index induces the complement of a path on $k + 1$ vertices
– $(v_i^s, x_{2s-1}), (v_i^s, x_{2s})$ if $i \neq 1, 3, \lambda - 3, \lambda - 1$

Moreover, for each $s = 1, \dots, t$ the vertices v_1^s and $v_{\lambda-1}^s$ are identified with the x, y vertices of its uniquely associated coupling gadget F_c^1, and similarly v_3^s and $v_{\lambda-1}^s$ are merged with the x, y of a pair of F_c^3s.

When $k = 4$, we join polarity gadgets in a similar way: Use t copies of the polarity gadget, the x-vertices represent positive occurrences and the y-vertices negations. Now with a help of the $t(k + 1)$ new vertices $v_1^1, \dots, v_{k+1}^t$, we define the remaining edges as:

– (v_i^s, v_j^s) if $|i-j| \geq 2$, i.e., each $(k+1)$-tuple with the same upper index induces the complement of a path on $k+1$ vertices
– $(v_i^s, y_s), (v_i^s, x_{s+1})$ for $i = 2, 4$

As above, two coupling gadgets F_c^1, F_c^3 are joined to vertices v_1^s and v_5^s (gadget F_c^1) and to v_3^s (gadget F_c^3); both connections terminate in v_3^s). See Fig 4 (right) for a detail of this construction.

Observe that at this moment vertices of variable gadgets are of degree $k-1$ (the x_i's) or of degree k (all others).

Clause gadgets: Each clause gadget consists of $k+3$ vertices $z_1, z_2, z_3, w_1, \ldots, w_k$, and of the following edges:

– (w_i, w_j) if $|i-j| \geq 2$, inducing the complement of a path on k vertices
– (z_i, w_j) if $i = 1, 2, 3$ and $2 \leq j \leq k-2$

Clause gadgets of complementary clauses C and C' are joined by use of

– three swallowing gadgets F_s where for each $i = 1, 2, 3$, the vertices x, y are identified with z_i and z'_i. (Both z vertices must represent the same variable — one a positive occurrence, the other one a negated occurrence),
– two coupling gadgets F_c^k where both x's are merged with w_k and both y's with w'_k,
– a coupling gadget F_c^1 between w_1 and w'_1,
– the edge (w_1, w'_1).

Completing the construction: Finally, all variable and clause gadgets are composed together as follows: The x-vertices of variable gadgets are linked in one-by-one manner to the z-vertices of clause gadgets such that edges between gadgets represent the variable-clause incidence relation in Φ between the associated variables and clauses. As was already noted above, vertices x_i with odd i indicate positive occurrences of the associated variable, while those with an even i represent negations. (Formally, if a variable v occurs positively in a clause c, we pick a unique x_i, i odd, of the gadget representing v and a unique z_j of the clause gadget representing c and insert into G the edge (x_i, z_j). Similarly for a negated variable we choose x_i with an even index i.) This concludes the construction of the graph G, see Fig. 4 for an illustration.

Clearly G is k-regular. It remains to be shown that G admits an $L_{(2,1)}$-labeling of span $k+2$ if and only if Φ is NAE-satisfiable. In particular, we prove that the NAE-satisfying Boolean assignments ϕ are in one-to-one correspondence with valid labelings f of G via the equivalence:

(*) $\phi(v) = \text{TRUE} \Leftrightarrow f(x_1) = 0$ for x_1 of the variable gadget representing v.

Assume first that Φ is NAE-satisfied by an assignment ϕ. Then the partial of x_1's can be extended to all variable gadgets such that each x_i is inside the gadget incident with vertices $2, 3, \ldots, \lambda - 2$. (Just set $f(v_i^s) = i$ and extend it to the polarity and coupling gadgets.) Consider a clause C and its gadget. Without loss of generality we may assume that C contains one positively and two negatively valued literals, i.e., (up to an permutation of indices) z_1 is adjacent to a variable

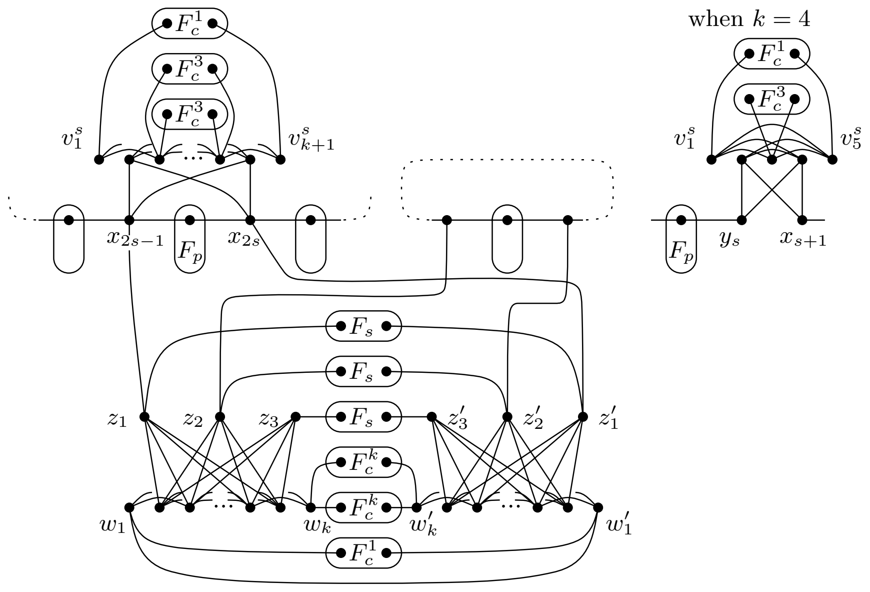


Fig. 4. Construction of G, variable gadgets in the upper part, two complementary clause gadgets at the bottom. The different construction of the variable gadget for $k = 4$ shown on the right side.

vertex labeled by λ, and z_2, z_3 to vertices labeled by 0. We extend f onto the clause gadget by letting $f(z_1) = 0$, $f(z_2) = \lambda - 1$, $f(z_3) = \lambda$ and $f(w_i) = i$. For the complementary clause C', we label its gadget symmetrically and extend f to the remaining swallowing and coupling gadgets to get a valid labeling of the entire graph G.

In the opposite direction, it is easy to observe that in a valid $L_{(2,1)}$-labeling f of G of span $k + 2$ the following arguments hold:

– Up to symmetry the polarity gadgets allow only one possible labeling, where in each variable gadget $f(x_i) \neq f(x_{i+1})$, $f(x_i) \in \{0, \lambda\}$.
– For $k \geq 5$, the $k - 3$ common neighbors $\{v_2^s, v_4^s, v_5^s, \ldots, v_{\lambda-4}^s, v_{\lambda-2}^s\}$ of x_{2s-1} and x_{2s} must be labeled by the set $\{2, 4, 5, \ldots, \lambda - 4, \lambda - 5\}$ regardless the labeling of x_{2s-1} and x_{2s}. (Note that each x_i gets neighbors labeled 3 and $\lambda - 3$ inside the polarity gadgets.)
  Similarly for $k = 4$, it holds that $f(y_s) \neq f(x_{s+1})$ and $f(z_2^s), f(z_4^s) \in \{2, 4\}$.
– If x_i is labeled 0, then its neighbor z_j is labeled either λ or $\lambda - 1$ and symmetrically if $f(x_1) = \lambda$ then $f(z_j) \in \{0, 1\}$.
– In each clause gadget the labels of z_1, z_2 and z_3 must be distinct since they share a common neighbor (e.g., the vertex w_1). Then from both sets $\{0, 1\}$ and $\{\lambda - 1, \lambda\}$ at least one label is used on $\{z_1, z_2, z_3\}$.

Then ϕ defined by (*) is a NAE-satisfying assignment for Φ, i.e., each clause contains some positively as well as also some negatively valued literals. This concludes the proof of NP-hardness of the problem. Membership in NPis obvious.

5 Conclusion

We have shown NP-hardness of determining the minimum span of $L(2,1)$-labelings of regular graphs by proving that for every $k \geq 3$, the decision problem whether $\lambda_{(2,1)}(G) \leq k+2$ is NP-complete for k-regular graphs G. Note that the bound $k+2$ is the minimum possible, no k-regular graph allows an $L(2,1)$-labeling of span less than $k+2$.

We conjecture that for every $k \geq 3$, there exists a constant c_k (depending on k) such that the decision problem $\lambda_{(2,1)}(G) \leq \lambda$ restricted to k-regular graphs is NP-complete for every fixed $\lambda \in \{k+2, k+3, \dots, c_k\}$ and polynomially solvable for all other values of λ. The latter is certainly true for small λ (i.e., $\lambda \leq k+1$). The upper bound is more interesting. In particular, if our conjecture is true, it still remains a question how far is the c_k from $\lambda_k = \max\{c : \exists k\text{-regular } G \text{ s.t. } \lambda_{(2,1)}(G) > c\}$. We conjecture that $c_k \neq \lambda_k$, i.e., that in the upper part of the spectrum there will be space for nontrivial polynomial time algorithms. Note finally that $\lambda_k \leq k^2+k-2$ follows from [9], and that $\lambda_k \leq k^2-1$ if the conjecture of Griggs and Yeh is true.

References

1. Calamoneri, T. The $L(h,k)$-labeling probelm: a survey. Tech. Rep. 04/2004, Dept. of Comp. Sci, Univ, of Rome - "La Sapienza", 2004.
2. Fiala, J., and Kratochvíl, J. Partial covers of graphs. *Discussiones Mathematicae Graph Theory 22* (2002), 89–99.
3. Fiala, J., Kratochvíl, J., and Kloks, T. Fixed-parameter complexity of λ-labelings. *Discrete Applied Mathematics 113*, 1 (2001), 59–72.
4. Garey, M. R., and Johnson, D. S. *Computers and Intractability.* W. H. Freeman and Co., New York, 1979.
5. Georges, J. P., and Mauro, D. W. On regular graphs optimally labeled with a condition at distance two. *SIAM Journal of Discrete Mathematics 17*, 2 (2003), 320–331.
6. Griggs, J. R., and Yeh, R. K. Labelling graphs with a condition at distance 2. *SIAM Journal of Discrete Mathematics 5*, 4 (1992), 586–595.
7. Hale, W. K. Frequency assignment: Theory and applications. *Proc. of the IEEE 68*, 12 (1980), 1497–1514.
8. Král', D. Coloring powers of chordal graphs. *SIAM J. Discrete Math. 18*, 3 (2004), 451–461.
9. Král', D., and Škrekovski, R. A theorem about the channel assignment problem. *SIAM J. Discrete Math. 16*, 3 (2003), 426–437.
10. Kratochvíl, J., Proskurowski, A., and Telle, J. A. Covering regular graphs. *Journal of Combinatorial Theory B 71*, 1 (Sept 1997), 1–16.
11. Sakai, D. Labeling chordal graphs: distance two condition. *SIAM Journal of Discrete Mathematics 7*, 1 (1994), 133–140.
12. Schaefer, T. J. The complexity of the satisfability problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing* (1978), ACM, pp. 216–226.

A Polymerase Based Algorithm for SAT

Giuditta Franco[1,2]

[1] Dept of Computer Science, University of Verona, Italy
franco@sci.univr.it
[2] Dept of Mathematics, University of South Florida, USA
gfranco@cas.usf.edu

Abstract. A DNA algorithm based on polymerase extension and cutting operations is proposed to solve an instance of SAT with significant dimension. It suggests a combination of efficient DNA operations following the schema of the extract model. Its generating step is performed by a variant of a recently introduced technique called XPCR, while the extraction of the solutions is implemented by a combination of polymerase extension and restriction endonuclease action.

Keywords: DNA Computing, Polymerase, Restriction Enzymes, SAT, XPCR.

1 Introduction

Solving NP-Complete problems in linear time is one of the goals of DNA Computing, where information is stored in bio-polymers, and enzymes manipulate them in a massively parallel way according to strategies that can produce computationally universal operations [6]. The massive parallelism and the non determinism of the DNA computations allow to attack hard problems even by using the brute force search of the solutions.

After the seminal Adleman's experiment, where the solution of an instance of Hamiltonian Path Problem was found within DNA sequences [1], Lipton showed that the satisfiability problem can be solved by using essentially the same biotechniques [9]. The schema of the Adleman-Lipton *extract model* consists of two main steps: the combinatorial *generation* of the solution space (a test tube containing the pool of DNA strands encoding all the possible solutions) and the *extraction* of (the strands encoding) the true solutions.

The more recent method to generate a solution space is called *mix-and-split* and was introduced in [3] to generate an initial RNA pool for the knight problem, that is a combinatorial library of binary numbers. It was used in [2] to generate the initial pool for a 20-variable 3-SAT problem, the biggest instance solved in lab so far. It combines a chemical synthesis of DNA sequences and an enzymatic extension method; in fact, two half libraries are combined by primer extension while each of them was synthesized chemically on two columns by repetitive mixing and splitting steps. This method takes the advantages of an extension method but performs a big part of the work by chemical synthesis, that is quite sophisticated and expensive.

The extraction of DNA strands including a given sequence of bases has been the critical step in the Adleman-Lipton paradigm. The traditional methods for separating the strands containing given substrands from the other ones by Watson-Crick complementarity were showed to be not complete, that is, some solutions could be lost [8]. More recently micro-fluid devices for separating strands were introduced [15], and an automated mix of thermo-cycler and gel-electrophoresis was developed in [2] in order to extract the strands encoding the satisfying assignments of a formula from a large pool of molecules.

Here we present a DNA algorithm solving SAT in linear time and based on polymerase extension, enzymatic cuts, and electrophoresis, that are considered standard cheap easy and efficient methods to perform DNA computations.

The generation step of this algorithm is going to be implemented by a particular application of the XPCR, that is a polymerase-based technique introduced in [4] as extraction tool[1] and then used in [5] as generation tool. Lab experiments were carried on for testing the technique in different situations. Also, the implementation of the generation and the extraction XPCR-based procedures resulted to be convenient with respect to the standard methods, in terms of efficiency, speed and feasibility [4,5].

The XPCR generation algorithm proposed in [5] starts from *four* specific DNA sequences and generates the whole SAT solution space of 2^n different sequences (where n is the number of boolean variables) in linear time, in fact it performs $2(n-1)$ recombinations. Here an improved generation procedure is proposed, which starts from *two* specific DNA sequences and performs $\lceil \log_2 n \rceil$ recombinations.

The extraction step is going to be implemented by a suitable combination of enzymatic operations. In particular it needs a polymerase extension and 'blunt' cuts of a restriction enzyme, we use for example the SmaI endonuclease. The restriction enzymes are known as molecular scalpels because of their cut precision, and they perform DNA computations with an efficiency of 100%. Usually the problem for using enzymes in DNA computing is that there exists only a limited number of them in nature, so we are able to perform a limited number of different cuts. However, in our algorithm only one kind of cut is required, regardless the dimension of the problem.

The idea of this extracting step was inspired by [13], where hairpin formations of single strands were used as test of inconsistent assignments of variables and destroyed by a restriction enzyme. But here totally different algorithmic procedure is used, where polymerase extension is an important feature to select the 'good' strings. Also, we manipulate a solution space with 2^n elements instead of one with 3^m elements, where m is the number of clauses of the considered SAT instance.

In the following section we describe the DNA computing background necessary to understand the algorithm proposed in the section 4. The SAT problem

[1] Invention covered by a patent from U.S. Department of Commerce, Docket number: BUCH-001PRV.

with a DNA encoding is described in the section 3, while some final remarks in the last section conclude the paper.

2 DNA Computing Background

In this section we recall the notions related to electrophoresis, to the polymerase action that is basic for describing traditional PCR and XPCR techniques, to splicing rules and to restriction enzymes.

All the operations are performed on a test tube containing many copies of each DNA sequence. In the following we say *pool P* such a test tube; formally it is a multiset of strings (over a given alphabet), that is a set of strings having positive multiplicity. The union operation $\mathbf{P_1} \cup \mathbf{P_2}$ produces the pool resulting after a mix (called also merge) of the two pools P_1 and P_2.

There is a natural association between a DNA sequence and the number of its bases, that number is called the *length* of the sequence. Electrophoresis is a standard gel-based technique that selects DNA sequences with respect to their length. Its precision depends on the kind of gel, and even sequences that differs of one basis can be separated. We indicate with $\mathbf{El_r(P)}$ the pool containing all the sequences of P having length exactly r, or, in other words, the pool obtained from P after an electrophoresis where the sequences r long were selected.

PCR is one of the most important and efficient tool in biotechnological manipulation and analysis of long DNA molecules. The main ingredients of this reaction are polymerase enzymes which implement a very simple and efficient duplication algorithm on double oriented strings. The PCR procedure is based on: i) templates, ii) a copy rule applied to templates, iii) initial short strings (primers) that say where the copying process has to start. Polymerase enzyme 'writes' a string step by step, in the $5' - 3'$ direction, by coping (in complementary form) the bases of the template which drives the copy process (as in the Figure 1).

The three parts of the Polymerase Chain Reaction are carried out in the same pool, but at different temperatures. The first part of the process separates the two DNA chains and this is done simply by heating the pool to 90-95 degrees centigrade; we indicate this operation on the pool P with $\mathbf{H(P)}$. Since the primers cannot bind to the DNA strands at such a high temperature, the pool is then cooled to 55 degrees. At this temperature, the primers bind to their complementary portions along the DNA strands. We denote this step of priming, or cooling down, with $\mathbf{C(P)}$.

The final step of the reaction is intended to make a complete copy of the templates. The temperature of the pool is raised at around 75 degrees (to allow only primers to be annealed) in which the Taq polymerase works well (this is the temperature of the hot springs where the bacterium was discovered). It begins adding nucleotides to elongate the primer and eventually makes a complementary copy of the template. If the template contains an A nucleotide, the enzyme adds a T nucleotide to the primer. If the template contains a G, it adds a C to the new chain, and so on to the end of the DNA strand. We indicate this step

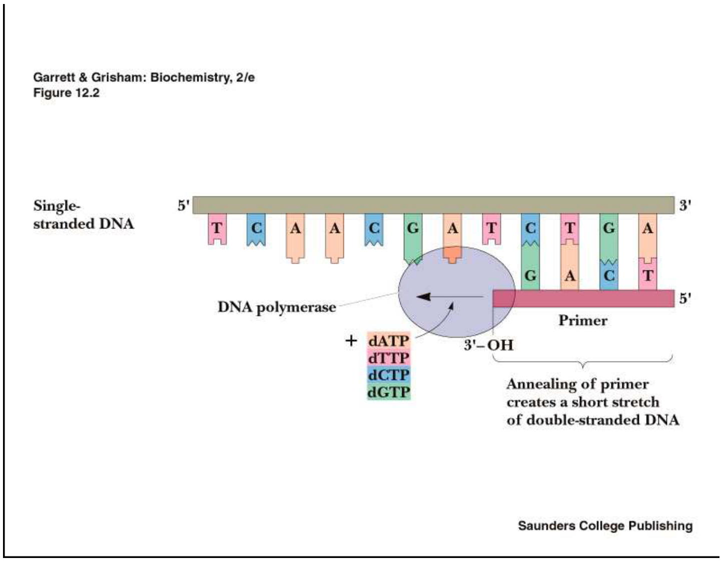

Garrett & Grisham: Biochemistry, 2/e
Figure 12.2
Single-stranded DNA
5'
3'
T C A A C G A T C T G A
G A C T
5'
DNA polymerase
Primer
+ dATP dTTP dCTP dGTP
3'–OH
Annealing of primer creates a short stretch of double-stranded DNA
Saunders College Publishing

Fig. 1. Polymerase extension

as $\mathbf{Taq}(\mathbf{P})$, where a polymerase extension is performed by elongating all the annealed primers of P.

The three steps in the Polymerase Chain Reaction - the separation of the strands, annealing the primer to the template, and the synthesis of new strands - take less than two minutes. We indicate with $\mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$ a standard PCR with primers α and $\overline{\beta}$ applied on the pool P; it amplifies all the portions having as prefix the sequence α and as suffix the sequence β (because the sequences are considered, as usual, in the $5'-3'$ orientation).

Starting from an heterogeneous pool of DNA double strands sharing a common prefix α and a common suffix β, and given a specified string γ, by means of $\mathbf{XPCR}_\gamma(\mathbf{P})$ we recombine all the strings of the pool P that contain γ as substring. In particular, the procedure $XPCR_\gamma(P)$ implements the following version of *null context splicing rule* r_γ introduced in [7]:

$$r_\gamma: \quad \alpha\,\phi\,\gamma\,\psi\,\beta,\ \alpha\,\delta\,\gamma\,\eta\,\beta \quad\longrightarrow\quad \alpha\,\phi\,\gamma\,\eta\,\beta,\ \alpha\,\delta\,\gamma\,\psi\,\beta,\ \alpha\,\phi\,\gamma\,\psi\,\beta,\ \alpha\,\delta\,\gamma\,\eta\,\beta$$

where $\alpha\phi\gamma\psi\beta$ and $\alpha\delta\gamma\eta\beta$ are elements of P.

The procedure $XPCR_\gamma(P)$ consists of the following steps.

input pool P of strings having α as prefix and β as suffix

- **split** P into P_1 and P_2 (with the same approximate size)
- $P_1 := \mathbf{PCR}_{(\alpha,\gamma)}(\mathbf{P_1})$ **and** $P_2 := \mathbf{PCR}_{(\gamma,\beta)}(\mathbf{P_2})$ (*cutting step*[2], Figure 2)

[2] In order to implement these PCRs in parallel, the encodings of primers must be such that their melting temperatures is approximately the same.

- $P := \mathbf{P_1} \cup \mathbf{P_2}$
- $P := \mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$ (*recombination step*, Figure 3).

output P.

After the cutting step we find in the test tubes P_1 and P_2 an exponential amplification of the double stranded DNA strings $\alpha \ldots \gamma$ and $\gamma \ldots \beta$ respectively (see Figure 2), that are shorter than the initial molecules.

Since products linearly amplified keep the initial length, an electrophoresis could be performed after the cutting steps for eliminating the longest strands; it would clean the signal from the noise caused by these amplifications and would decrease the production of unspecific matter.

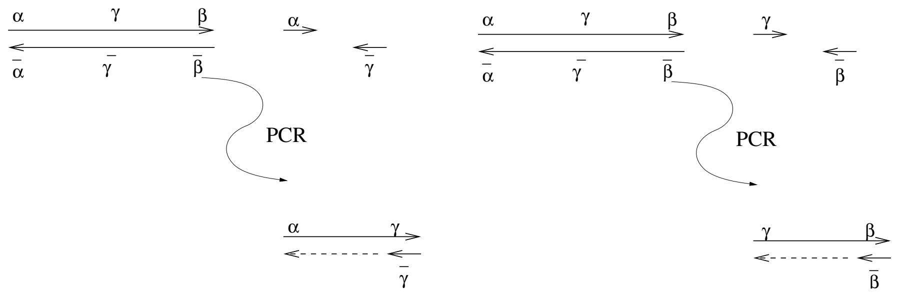


Fig. 2. Cutting step of $XPCR_\gamma$

In the recombination step, left parts $\alpha \cdots \gamma$ and right parts $\gamma \cdots \beta$ of the initial sequences having γ as subsequence are recombined in all possible ways, regardless to the specificity of the sequences between α and γ, or γ and β. Therefore, not only the whole sequences containing γ are restored but also new sequences are generated by recombination (see Figure 3).

The recombination of strings having a specific common substring γ is performed in nature by some restriction enzymes. They cleave the double helix at the end of γ wherever they find the sequence, in such a way that (after a cut) a common overhang γ allows the recombinations of different DNA molecule along it. The recombining behaviour of restriction enzymes is perfectly modeled by *splicing rules*, that are widely investigated in literature from a formal language theory viewpoint [7,12]. Generally the restriction enzymes will bind to DNA at a specific recognition site and then cleave DNA mostly within, but sometimes outside of this recognition site.

This operation is very useful in biological situations (as mutagenesis) and in DNA Computing, but the limited number of enzymes stops the scale up of such computations to solve problems with high dimension. While the procedure $XPCR_\gamma$ has no limit regarding the choice of (the primer) γ, and it was tested working as expected [4].

Some enzymes perform a *blunt* cut, that is straight both strands (no overhang). Here we use just one of this enzyme, called SmaI. It is a *restriction*

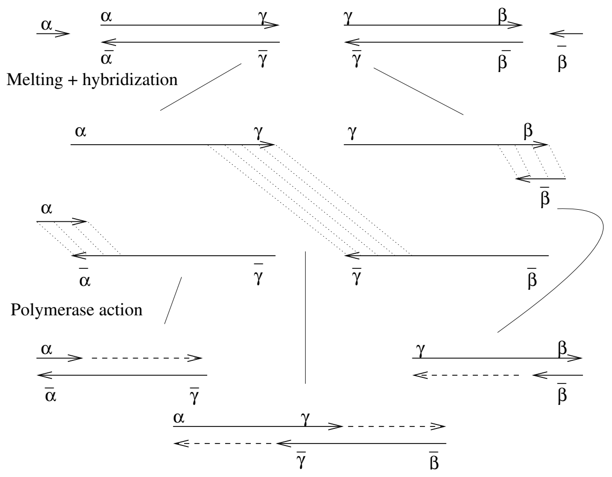


Fig. 3. Recombination step of $XPCR_\gamma$

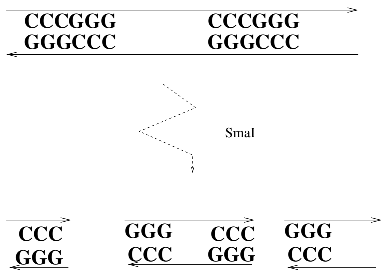


Fig. 4. Cut operation of the enzyme SmaI

endonuclease, thus it cuts only double stranded molecules, and in particular it has the restriction site CCCGGG. It means that, when we put this enzyme in a pool P, all the double strands containing CCCGGG are cut between the last C and the first G (as in Figure 4). We indicate with $\mathbf{SmaI_{cut}}$ **(P)** the pool containing the products of *SmaI* acting on P.

Now we choose the number of variables and clauses of an instance of SAT and briefly present the encodings of the initial data within DNA sequences.

3 SAT and DNA Encodings

Propositional satisfiability (or SAT) is the problem of deciding if it is possible to assign truth values to the variables in a propositional formula to make it true, using the standard interpretation for logical connectives. We consider SAT problems in conjunctive normal form (CNF): a formula in CNF is a a conjunction of clauses, where a clause is a disjunction of literals, and a literal is a variable or a negated variable. A k-$SAT(n, m)$ problem, in particular, can be seen as a boolean system of m disjunctive equations on n variables $x_1, \ldots, x_n$ and their negations $\neg x_1, \ldots, \neg x_n$ (that are literals l_i, where $i = 1, \ldots, n$ and $l_i \in \{x_i, \neg x_i\}$), where each equation (clause) has length smaller than or equal to k. If there exists an assignment satisfying a given formula or system we say it is satisfiable, and it belongs to the SAT class, otherwise we say it is unsatisfiable.

SAT was the first problem shown to be NP-complete (Cook, '71), and it is nowadays one of most studied NP-complete problems. A remarkable consequence of the theory of NP-completeness is that a large collection of problems (several thousand) from many different fields can be represented as instances of the k-SAT problem. If $k \geq 3$ then any instance of k-SAT can be reduced to an instance of 3-SAT at the cost of linearly increasing the size of the problem, therefore in the following we will consider only the 3-SAT problem, just to facilitate the explanation.

In terms of what is known theoretically about the probability of satisfiability for random 3-SAT, the *threshold conjecture* [11] is that there is some specific ratio of clauses to variables above which the probability of satisfiability approaches 1, and below which it approaches 0. In general NP-complete problems exhibit 'phase transition' phenomena, analogous to those in physical systems, with the hardest problems occurring at the phase boundary: across the phase dramatic changes occur in the computational difficulty, and the problems become easier to solve away from the boundary [14]. Experimental evidence strongly suggests a threshold with $\frac{m}{n} \approx 4.2$.

Therefore as a suggestion for an experiment we consider for example a significant instance of a 30-variable 3-SAT problem with 126 clauses. We have the variables $x_1, x_2, \ldots, x_{30}$ and the clauses $C_1, C_2, \ldots, C_{126}$ that are the disjunction of at most three literals.

As usual [9], we encode the variables with the DNA sequences X_1, X_2, ..., X_{30} respectively, and their negations $\neg x_1$, $\neg x_2$, ..., $\neg x_{30}$ with the sequences Y_1, Y_2, ..., Y_{30} respectively. For the sake of simplicity, we say that the literal l_i is encoded by the DNA sequence L_i, meaning that, if $l_i = x_i$ then $L_i = X_i$, and if $l_i = \neg x_i$ then $L_i = Y_i$.

We consider the following two initial sequences, where the sequence γ contains[3] the restriction site of SmaI, while α and β are two fixed sequences (long 18) necessary to perform the XPCR procedure

$$Z_1 = \alpha X_1 \gamma X_2 \gamma X_3 \cdots \gamma X_{30} \beta \quad \text{and} \quad Z_2 = \alpha Y_1 \gamma Y_2 \gamma Y_3 \cdots \gamma Y_{30} \beta.$$

We assume that each L_i has length 20, as it is a better choice for a primer length, and may not contain CCCGGG as subsequence. The sequences encoding the literals have to be 'very different' to each other, to avoid mismatches during the running of the algorithm described in the next section. The main encoding requirements here are that the elements of the set $\{X_1, \ldots, X_{30}, Y_1, \ldots, Y_{30}\}$ have no common subwords at least ten long that are complementary to each other, do not begin with sequences of G and do not end with sequences of C long more than five.

[3] More precisely, we choose to take γ = CCCCCCCCGGGGGGGG. It has length 16.

These conditions avoid mismatches in the pools involved by the operations of the algorithm, in fact such kind of properties are preserved by concatenation and splicing [10].

The length of the initial sequences Z_1 and Z_2 is 1100, resulting from the thirty literals twenty long, the twenty nine occurrences of γ, the prefix and suffix eighteen long. Note that the 2^{30} elements of the solution space $\alpha\xi_1\gamma\xi_2\cdots\gamma\xi_{30}\beta$ with $\xi_i \in \{X_i, Y_i\}$ have all length 1100.

4 Algorithm for Solving SAT

```
input: $P = \{Z_1, Z_2\}$
    begin
    $p := 1$, $q := 1$
  1. (Generation step)
     for $p = 1, \ldots, 5$
        begin
           $P := XPCR_\gamma(P)$
           $P := El_{1100}(P)$
           $p := p + 1$
        end
  2. (Extraction step)
     for $q = 1, \ldots, 126$
     Consider $C_q = l_i^{(q)} \vee l_j^{(q)} \vee l_k^{(q)}$ and set $P_q := \{L_i^{(q)}, L_j^{(q)}, L_k^{(q)}\}$
        begin
           $P := P \cup P_q$
           $P := H(P)$
           $P := C(P)$
           $P := Taq(P)$
           $P := SmaI_{cut}(P)$
           $P := El_{1100}(P)$
           $P := PCR_{(\alpha,\beta)}(P)$
        end
      end
     output: P
```

The output pool P contains the solutions, if there exist, otherwise it is empty (and the instance is not in SAT). In general, the generation (first) step is performed $\lceil \log_2 n \rceil$ times and the extraction (second) step m times. In fact, the general form of the algorithm is obtained simply by substituting 5 with $\lceil \log_2 n \rceil$, where n is the number of variables, 1100 with the length of the initial sequences, and 126 with the number m of clauses.

4.1 Generation Step

Firstly we analyze the first cycle of the operation *for* of the generation step.

All the recombinations along the common substring γ are performed by means of XPCR over the initial pool. In each of the initial strings of our example there are twenty nine occurrences of γ. The primer γ in the cutting step of the XPCR (see the details of the XPCR steps in section 2) anneals with one of these occurrences, chosen randomly. If we denote with $\gamma_1, \gamma_2, \ldots, \gamma_{29}$ the occurrences of the same sequence γ, after the cutting step we obtain from P the two pools P_1 and P_2 respectively:

$$\{\alpha X_1\gamma_1,\ \alpha Y_1\gamma_1,\ \alpha X_1\gamma_1 X_2\gamma_2,\ \alpha Y_1\gamma_1 Y_2\gamma_2,\ \ldots,\ \alpha X_1 \cdots X_{29}\gamma_{29},\ \alpha Y_1 \cdots Y_{29}\gamma_{29}\}$$

and

$$\{\gamma_1 X_2\gamma_2 X_3 \cdots \beta,\ \gamma_1 Y_2\gamma_2 Y_3 \cdots \beta,\ \gamma_2 X_3 \cdots \beta,\ \gamma_2 Y_3 \cdots \beta,\ \ldots,\ \gamma_{29} X_{30}\beta,\ \gamma_{29} Y_{30}\beta\}$$

The recombination step of the XPCR procedure on $P_1 \cup P_2$ produces a pool with sequences of different lengths (the shorter ones are of kind $\alpha X_1\gamma Y_{30}\beta$ and the longest ones of kind $\alpha Y_1\gamma \cdots Y_{29}\gamma X_2\gamma X_3 \cdots X_{30}\beta$) where only the sequences 1100 long contain just one of each literal (in the sound order).

Therefore, after the operation $El_{1100}(P)$ we have a pool with sequences that are product of (only) one recombination of the initial sequences, by means of a null context splicing rule r_γ. The pool includes all the sequences of the following type

$$\alpha X\gamma \cdots \gamma X\gamma Y\gamma \cdots \gamma Y\beta, \qquad \alpha Y\gamma \cdots \gamma Y\gamma X\gamma \cdots \gamma X\beta.$$

Note that after the first cycle *for* all the possible couples $L_i\gamma L_{i+1}$ with $i = 1, \ldots, 29$ are present in the pool as subsequences.

After the cutting step of the XPCR in the second cycle, we have analogously two pools P_1 and P_2 containing respectively:

$$\{\alpha X_1\gamma_1 X_2 \cdots X_i\gamma_i Y_{i+1} \cdots Y_j\gamma_j,\ \ \alpha Y_1\gamma_1 Y_2 \cdots Y_i\gamma_i X_{i+1} \cdots X_j\gamma_j\}_{j>i}$$

for $i = 1, \ldots, 28$ and $j = 2, \ldots, 29$, and

$$\{\gamma_i X_{i+1}\gamma_{i+1} \cdots X_j\gamma_j Y_{j+1} \cdots Y_{30}\beta,\ \ \gamma_i Y_{i+1}\gamma_{i+1} \cdots Y_j\gamma_j X_{j+1} \cdots X_{30}\beta\}_{j>i+1}$$

for $i = 1, \ldots, 28$ and $j = 3, \ldots, 30$.

After the recombination step and the electrophoresis selecting the sequences 1100 long, all the possible sequences of the following type are present in the pool:

$$\alpha X \cdots X\gamma Y \cdots Y\gamma X \cdots X\gamma Y \cdots Y\beta,\ \alpha Y \cdots Y\gamma X \cdots X\gamma Y \cdots Y\gamma X \cdots X\beta$$

In other words, there are all the recombinations where, for at most three different values of i, the 'crossing points' $X_i\gamma_iY_{i+1}$ or $Y_i\gamma_iX_{i+1}$ occur as subsequences.

Note that, in the pool resulting after the second cycle *for*, all the possible recombinations $L_i\gamma L_{i+1}\gamma L_{i+2}\gamma L_{i+3}$ with $i = 1, \ldots, 27$ are present as subsequences.

For example, we obtain the subsequence $X_4Y_5X_6Y_7$ as product of the first two cycles in the following way:

– the first cycle produces $\alpha X_1...X_4\gamma Y_5...Y_{30}\beta$ and $\alpha X_1...X_6\gamma Y_7...Y_{30}\beta$ by means of XPCR application on the initial sequences along the fourth and the sixth occurrences of γ respectively.
– the XPCR of second cycle recombines the sequences of the pool previously obtained also with respect to the fifth occurrence of γ, so producing:

$$\alpha X_1 \cdots X_4\gamma_4Y_5\gamma_5Y_6 \cdots Y_{30}\beta, \quad \alpha X_1 \cdots \gamma_5X_6\gamma_6Y_7 \cdots Y_{30}\beta$$

$$\downarrow$$

$$\ldots,\ \alpha X_1 \cdots X_4\gamma_4Y_5\gamma_5X_6\gamma_6Y_7 \cdots Y_{30},\ \ldots$$

At this point it is clear how after the k-th cycle *for*, all the possible recombinations of 2^k consecutive literals are present in the pool as subsequences. Since after the generation step all the possible recombinations of n literals have to be present, then $\lceil \log_2 n \rceil$ cycles suffices to produce the solution space.

4.2 Extraction Step

Once obtained the solution space, in the extraction step the (sequences encoding the) *true* solutions are selected from (those encoding) the *possible* solutions. Its implementation combines a polymerase extension with a cut of the restriction endonuclease SmaI, and after an electrophoresis the sequences of interest are amplified by a standard PCR.

The key process consists of the first three steps, which recall those of a standard PCR but produce a different result depending on the choice of 'primers'. Here we have three primers instead of two, and overall they are designed to anneal with a same side of the double strands.

To each step of the cycle *for* is associated a different clause, and after that step all the sequences satisfying that clause have been extracted. The iteration of such a cycle for all clauses on a same pool allows the final pool to contain only the sequences that satisfy *all* the clauses.

We explain the computation of the extraction step with an example, by considering the process which corresponds to the first cycle *for* related to the clause $C_1 = x_{10} \vee \neg x_{27} \vee x_{15}$. It starts on the pool P containing the solution space, and firstly it sets $P_1 = \{X_{10},\ Y_{27},\ X_{15}\}$ as indicated by C_1.

1. $\mathbf{P} := \mathbf{P} \cup \mathbf{P_1}$
   Many copies of *literal primers*, that are the elements of P_1, are added to the pool (also free nucleotides if necessary, but this is not relevant from an algorithmic point of view).

2. $\mathbf{P} := \mathbf{H}(\mathbf{P})$
   When the pool is heated then the two strands of all the molecules present in the pool come apart[4], see the Figure 5.

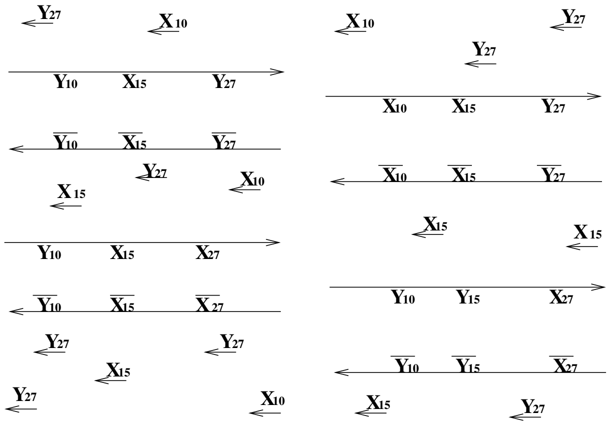


Fig. 5. Denaturing of all the molecules

3. $\mathbf{P} := \mathbf{C}(\mathbf{P})$
   By cooling down the pool, literal primers bind to their complementary strands, see the Figure 6.
4. $\mathbf{P} := \mathbf{Taq}(\mathbf{P})$
   A polymerase extension elongates the literal primers wherever they are attached, see Figure 7.
5. $\mathbf{P} := \mathbf{SmaI_{cut}}(\mathbf{P})$
   Since the restriction site of the SmaI is contained (only) in γ, which is uniformly distributed in the molecules, and since the enzyme cuts only double stranded molecules, after this step only the single strands have length 1100. They encode assignments satisfying the clause C_1 because contain at least one of the literal primers in P_1. The algorithm extracts them by means of the operation $\mathbf{El_{1100}}(\mathbf{P})$, and then amplifies them with the $\mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$ which restores their double stranded form.

The sequences selected by the first step are filtered by the second step in such a way that only those satisfying also the clause C_2 are retained, and so on for all the $m = 126$ clauses.

[4] This process is called *denaturation.*The complementary of a strand α is denoted by $\overline{\alpha}$.

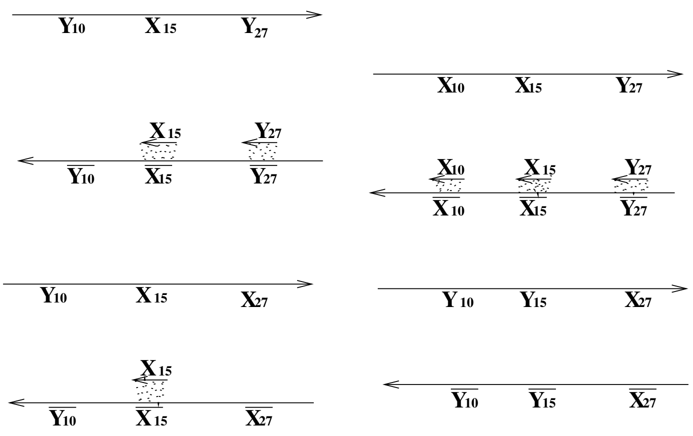


Fig. 6. Selecting by literal primers

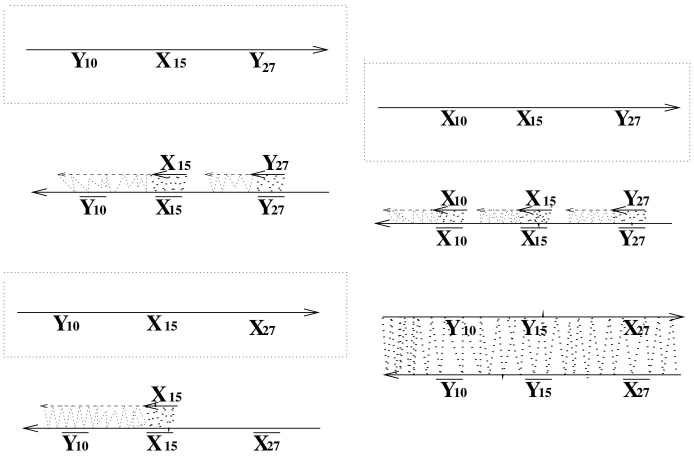


Fig. 7. Detecting of the ‘good’ assignments as single strands

From a theoretical viewpoint, every solution is present in the final pool since a primer eventually binds to its complementary, thus the algorithm is complete.

While, from an experimental viewpoint, the correctness depends on the encodings. In fact, when mismatches due to ‘wrong’ encoding occur, a literal primer anneals in a wrong location and that sequence is selected as good assignment even if it is not. On the other side, this case is quite rare, because it should happen that there exists an assignment that satisfies all the clauses but one, and that the error occurs just for that clause. That is, since each solution is filtered m times, to have a non-solution assignment in the final pool, mismatch errors should occur on that assignment for all the clauses which it does not satisfy, and we can consider it having low probability.

5 Conclusion

The main result of the paper is the design of a non-implemented yet DNA algorithm for solving a SAT instance. It is described for an instance of significant size in order to suggest a prompt implementation, but its generalization to any number n of variables and m of clauses is indicated.

The algorithm essentially consists of enzymatic operations, that result convenient with respect to the other methods in terms of efficiency and precision. The implementation of the first part of the algorithm would improve the results obtained by the last lab experiment carried on for testing the XPCR technique on a generation process. While the lab implementation of the second part would show that the extraction step can be performed by using only the polymerase extension and a restriction enzyme, that are notoriously cheap and simple to implement biotechnologies.

It seems to us that this algorithm could be significant in DNA Computing area, because it improves noteworthly the existing methods for performing the generating and extracting steps of the traditional extract model. With respect to [2] for example, no chemical synthesis and *ad hoc* automated thermocycler are required.

Acknowledgment

The author deeply thanks Prof. Vincenzo Manca of Department of Computer Science of the University of Verona, for his important suggestions, the interesting discussions, and his support in the whole idea of this paper.

References

1. L. M. Adleman, *Molecular Computation of solutions to combinatorial problems*, Science **266**, pp 1021-1024, 1994.
2. R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, L. M. Adleman, *Solution of a 20-variable 3-SAT problem on a DNA computer*, Science **296**, pp 499-502, 2002.
3. D. Faulhammer, A. R. Cukras, R. J. Lipton, L. F. Landweber, *Molecular computation: RNA solution to chess problems*, Proc. Natl. Acad. Sci. USA **98**, pp 1385-1389, 2000.
4. G. Franco, C. Giagulli, C. Laudanna, V. Manca, *DNA Extraction by XPCR*, C. Ferretti G. Mauri C. Zandron et al. eds, Proceedings of DNA 10, LNCS 3384, Springer-Verlag Berlin Eidelberg, pp 106-114, 2005.
5. G. Franco, V. Manca, C. Giagulli, C. Laudanna, *DNA Recombination by XPCR*, Proceedings of DNA11, London, Ontario, June 2005, to appear.
6. R. Freund, L. Kari, G. Păun, *DNA Computing Based on Splicing: The Existence of Universal Computers*, Theory of Computing Systems **32** (1), Springer-Verlag New York, pp 69-112, 1999.
7. T. Head, *Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors*, Bulletin of Mathematical Biology **49**, pp 737-759, 1987.

8. D. Li, *Is DNA computing viable for 3-SAT problems?*, TCS **290**, pp 2095-2107, 2003.
9. R. J. Lipton, *DNA solutions of hard computational problems*, Science **268**, pp 542-544, 1995.
10. N. Jonoska, K. Mahalingam, *Languages of DNA based code words*, J Chen and J. Reif eds, Proceedings of DNA9, LNCS 2943, pp 61-73, 2004.
11. S. Kirkpatrick, B. Selman, *Critical behaviour in the satisfiability of random Boolean expressions*, Science **264**, pp 1297-1301, 1994.
12. V. Manca, *A Proof of Regularity for Finite Splicing*, Aspects of Molecular Computing: Essays Dedicated to Tom Head on the Occasion of His 70th Birthday, LNCS 2950, Springer-Verlag, pp 309-318, 2004.
13. K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya, *Molecular Computation by DNA Hairpin Formation*, Science **288**, pp 1223-1226, 2000.
14. Selman Bart, Mitchell David G., Levesque Hector J., *Generating hard satisfiability problems*, Artificial Intelligence **81**, pp 17-29, 1996.
15. D. van Noort, F. Gast, J. S. McCaskill, *DNA computing in microreactors*, N. Jonoska N. C. Seeman eds, Proceedings of DNA7, LNCS 2340, Springer, pp 44-53, 2002.

Laxity Helps in Broadcast Scheduling*

Stanley P.Y. Fung[1], Francis Y.L. Chin[1], and Chung Keung Poon[2]

[1] Department of Computer Science, The University of Hong Kong, Hong Kong
{pyfung, chin}@cs.hku.hk

[2] Department of Computer Science, City University of Hong Kong, Hong Kong
ckpoon@cs.cityu.edu.hk

Abstract. We study the effect of laxity, or slack time, on the online scheduling of broadcasts with deadlines. The laxity of a request is defined to be the ratio between its span (difference between release time and deadline) and its processing time. All requests have a minimum guaranteed laxity. We give different algorithms and lower bounds on the competitive ratio for different ranges of values of laxity, which not only represents a tradeoff between the laxity and the competitive ratio of the system, but also bridges between interval scheduling and job scheduling techniques and results. We also give an improved algorithm for general instances in the case when requests can have different processing times.

1 Introduction

The application of broadcasting in networks has been receiving much attention recently. Broadcasting has an advantage over point-to-point communication in that it can satisfy the requests of different users, who are requesting the same piece of information, simultaneously by a single broadcast. The advantage is more clearly seen when most of the requests are asking for common information like popular movies and weather information. Broadcasting is of even greater importance with the growing popularity of wireless and satellite networks which are inherently broadcasting in nature. There are some commercial systems that make use of broadcasting technology. For example, in the DirecPC system [1], clients make requests over phone lines and the server broadcasts the data via a satellite. In this paper, we study algorithms for the online scheduling of broadcasts with deadlines.

The Model. In the literature on broadcast scheduling, there are many different research work based on different assumptions of the network model. Here, we focus on the model in which the server holds a number of pages while requests come in and ask for pages (pull-based model). At any time, the server is allowed to broadcast only one page and all the requests asking for the same page can be satisfied simultaneously. Note that the server is not allowed to break down a

* The work described in this paper was fully supported by two grants from the Research Grants Council of the Hong Kong SAR, China [CityU 1198/03E and HKU 7142/03E].

page into different parts and send them over the network simultaneously. Also, we assume the receiver does not have any buffer to cache part of a page previously broadcasted by the server. This is an online problem, meaning that the server does not know the requests of the clients until they arrive, and the server have to determine which page to broadcast without knowing future requests.

Past Work. Broadcast scheduling was first studied for the case where requests do not have deadlines, and the objective is to minimize the maximum or the average flow time (also called response time, the time between arrival and completion of requests). Broadcast scheduling with deadlines is first studied in [11] and [12]. Each request is associated with a deadline, and the objective is to maximize the total profit of satisfied requests (completed before their deadlines). The two papers, however, considered different models in relation to how broadcasts can be preempted.

The preemptivity of online scheduling in general (and broadcasting in particular) can be classified into three different models. (For example see [8].) In the *nonpreemptive* model, a job that gets started must run to completion without interruption. In the *preemption-resume* model, jobs can be preempted and later resumed at the last point of execution. In the *preemption-restart* model, a job can be preempted, but its partial progress will be lost; hence, it must be restarted from the beginning if it is to be processed again later. In this case preemptions can also be called *abortions*. Note that the nonpreemptive and preemption-restart models are identical in the offline case.

While the preemption-resume and nonpreemptive models have been widely studied, there seems to be comparatively few results for the online scheduling of jobs in the preemption-restart model. Hoogeveen et al. [8] considered the case in which jobs have no weights (i.e. the objective is to maximize the utilization of processor), and Chrobak et al. [4] considered the case where in addition all jobs have equal length.

These three models have also been considered in online broadcast scheduling. Kalyanasundaram and Velauthapillai [11] considered the preemption-resume model, and Kim and Chwa [12] considered the preemption-restart model. The nonpreemptive broadcast model is also considered in a different context called *batching with incompatible job families* [9]. We consider the preemption-restart broadcasting model in this paper.

We distinguish between the case where all pages have the same length (the *unit-length* case), and the case where pages can have different lengths (the *variable-length* case). For the unit-length case, a 4.56-competitive algorithm is given in [14], while a lower bound of 4 follows from an interval scheduling problem [13]. For the variable-length case, there is a $(e\Delta + e + 1)$-competitive algorithm [14] where Δ is the ratio of maximum to minimum page lengths and $e \approx 2.718$. There is also a lower bound of $\sqrt{\Delta}$ [2].

Laxity. The power of broadcasting lies in its ability to combine multiple requests and serve them together by a single broadcast. For this to be effective, requests should have a certain amount of *laxity*, where the laxity of a request is defined to be the ratio of its span (the length of time interval between its deadline and release time) and its processing time (the length of the page it requests). This allows the system to delay processing a request and to serve it together with some other requests for the same page that arrives later. If a request has no laxity, it must be scheduled immediately or else it is lost, and hence the power of broadcasting is not utilized. In fact, in all of the above-mentioned lower bounds, all requests are tight (with no laxity). To actually analyze the effect of broadcasting, we assume all requests have a minimum laxity $\alpha > 1$. Intuitively, with larger laxity, the system will be able to schedule more requests together. However a large laxity may be unsatisfactory to users. In this paper we analyze the relation between the laxity of requests and the total profit obtained.

The issue of laxity has been considered in the unicast (i.e. non-broadcast) context. It is also called *slack*, *patience* [7] or *stretch factor* [5]. Online job (unicast) scheduling with laxity is widely studied, for example by Kalyanasunaram and Pruhs [10] in the preemption-resume model, and by Goldman et al [6] and Goldwasser [7] in the nonpreemptive model. The assumption of minimum laxity has also been used in preemption-resume broadcast scheduling with deadlines [11], and in nonpreemptive broadcast (or batching) problems [9].

Interestingly, the effect of laxity also allows this problem to bridge between interval scheduling and job scheduling. In most previous results on broadcast scheduling, the techniques used are similar to interval scheduling in which the most important concern is whether a broadcast should be preempted. When a broadcast is completed without being preempted, for example, the next broadcast is usually the one with the most pending requests. However our results indicate that as laxity increases, the problem has more job scheduling flavor, which involves selection of jobs based on both weights and deadlines. We will bring techniques from job scheduling into this problem.

Our results. In this paper we consider the effect of laxity in the preemption-restart broadcasting model. In Section 3 we first give an algorithm when the laxity is smaller than 2. It adapts an abortion criteria which varies with laxity and how much the current broadcast has been completed. It achieves a tradeoff in the competitive ratios, with smaller competitive ratios for larger laxity. Next we give a simple 2.618-competitive algorithm for the case where the laxity is at least 2. It does not use preemption and thus also applies to the nonpreemptive case. Unlike previous algorithms which only consider abortion conditions, our algorithm also considers how to select broadcasts after another broadcast is completed. In Section 4 we give lower bounds on the competitive ratio for different ranges of laxity α: for example, 8/3 for $1 < \alpha < 4/3$, 12/5 for $4/3 < \alpha < 1.4$, 2 for $1.4 < \alpha < 3/2$, and $\max(1 + 1/\lceil\alpha\rceil, 5/4)$ for $\alpha \geq 2$. For $\alpha < 2$ we extend the technique of Woeginger's lower bound of interval scheduling, while for $\alpha \geq 2$ we make use of job scheduling results. The lower bound does not approach 1 even with arbitrarily large laxity. Finally in Section 5 we consider the variable page length case, giving a $(\Delta + 2\sqrt{\Delta} + 2)$-competitive algorithm for general instances, which improves the previous bound of $e\Delta + e + 1$. This algorithm makes use of a simple observation and its analysis is significantly simpler than earlier algorithms. We also state a lower bound result with laxity.

2 Notations

There are a number of pages in a server, each having a possibly different length. Requests arrive to the server online, i.e. no information about the request is known until it arrives. Each request j has a release time $r(j)$, a deadline $d(j)$, the page $p(j)$ that it requests, and a weight $w(j)$. All $r(j), d(j)$ and $w(j)$ are real numbers. We define the processing time of a request, $l(j)$, to be the length of the page it requests. A request j fully completed before its deadline gives a profit of $w(j) \times l(j)$. For the unit length case we assume all pages have length 1, and hence weights and profits are equivalent. Define $\alpha = \min_j\{(d(j) - r(j))/l(j)\}$ to be the minimum laxity of all requests. We assume there is a pool that contains all pending requests. Requests that cannot be completed by their deadlines even if started immediately will be automatically removed from the pool.

A broadcast of a page J serves all pending requests of J simultaneously. We consider the preemption-restart model, in which a broadcast can be preempted (aborted) at any time, but must be restarted from the beginning if it is to be broadcast again. Let $|J| = \sum_{p(j)=J} w(j) \times l(j)$ denote the profit of a page J, i.e. the total profit of all pending requests j for the page J at a particular time. This is the profit obtained of broadcasting J. Our objective is to maximize the total profit of requests completed before their deadlines. We sometimes abuse the terminology and use 'page', 'broadcast (of a page)', and 'set of requests (for a page)' interchangeably. A schedule is specified by a sequence of broadcasts $J_1, J_2, \ldots$ where each broadcast J_i starts at time $s(J_i)$. If $s(J_i)+l(J_i) > s(J_{i+1})$, J_i is aborted by J_{i+1}, otherwise it is completed.

We measure the performance of online algorithms by their competitive ratios. Let OPT denote the offline optimal algorithm. An online algorithm A is R-*competitive* if, for any instance I, the profit $A(I)$ obtained by A is at least $1/R$ times the profit $OPT(I)$ obtained by OPT.

3 Unit Length Pages: Upper Bounds

3.1 Minimum Laxity $\alpha < 2$

We first consider the unit-length case where the minimum laxity is less than 2. We only consider those α that are rational. Let $\alpha = 1 + p/m$ for integers p, m where $p < m$ and m is the minimum possible. Intuitively, our algorithm uses an 'abortion ratio' which increases with time while a page is being broadcast.

Algorithm MultiLevel. Let $\beta > 1$ be the unique positive real root of $\beta - \beta^{1/m} - 1 = 0$. Suppose a page J starts being broadcast at time t. Then for $i = 1, 2, \ldots, m$, a new request for page J' arriving at $[t + (i-1)/m, t + i/m)$ (together with pending requests for J' in the pool [1]) will abort J if and only if $|J'| \geq \beta^{i/m}|J|$.

[1] It is possible that $J' = J$; we also consider requests currently being served by broadcast J to be in the pool if they can still be completed before their deadlines if restarted.

When a broadcast completes, the page with the maximum profit (for all pending requests) will be broadcast next.

The following table lists the choice of β for different values of laxity and the competitive ratios. Note that some of these ratios are larger than the 4.56 upper bound for the case of arbitrary instances given in [14].

α	1.2	1.25	1.33	1.4	1.5	1.6	1.67	1.75	1.8
β	2.164	2.221	2.325	2.164	2.618	2.164	2.325	2.221	2.164
R	4.714	4.638	4.510	4.448	4.236	4.221	4.08	4.04	4.026

Theorem 1. *For laxity $1 < \alpha < 2$ where $\alpha = 1 + p/m$, MultiLevel is $(\frac{\beta}{\beta-1} + \beta^{2-\alpha} + 1)$-competitive, where β is the root of $\beta - \beta^{1/m} - 1 = 0$.*

Proof. Let M denote the schedule produced by MultiLevel. Divide M into a set of *basic subschedules*, where each basic subschedule consists of zero or more aborted broadcasts followed by one completed broadcast. We charge the profits obtained by OPT to the basic subschedules in M. If every basic subschedule with a completed broadcast J receives charges at most R times that of $|J|$, then the algorithm is R-competitive. The general idea of the proof is to consider, for a fixed basic subschedule, the maximum-profit OPT schedule that is 'consistent' with the basic subschedule. For example, OPT cannot serve requests with profit much higher than that being served by M at the same time, unless it is already completed in M, because M will not ignore this request if it is still pending.

Without loss of generality we can assume all broadcasts made by M are of one of the lengths (time units) $1/m, 2/m, \ldots, 1$, since otherwise lengthening them to the closest length listed above will not decrease the profit obtained by OPT. Since the last broadcast in a basic subschedule must be completed, it must be of length 1. Note that all broadcasts of OPT must be of unit length since they are completed. (We can assume OPT will not make aborted broadcasts.)

Let $r = \beta^{1/m}$. We first consider the case that all basic subschedules satisfy two assumptions: (1) All broadcasts except the last one are of length $1/m$. (2) For any two consecutive broadcasts J and J' in the basic subschedule, $|J| = |J'|/r$. We will remove these assumptions later. Consider a basic subschedule. The maximum-profit OPT subschedule corresponding to this basic subschedule occurs when the total number of broadcasts in the basic subschedule is a multiple of m plus 2, with the OPT broadcasts as in Fig. 1. (The reason of this will become clear later in the proof.) Hence, let the basic subschedule be $(J_0, J_1, \ldots, J_m, \ldots, J_{km}, J_{km+1})$ for any $k \geq 0$. For those requests that are satisfied in an OPT broadcast started at time t and which are completed by M before time t, we charge their profits to the basic subschedule in M where it is completed. Let O_i be the set of requests in a broadcast started by OPT within J_i that are not completed by M before $s(O_i)$, i.e., they are still pending in M.

Consider the last broadcast O_{km+1} made by OPT. It consists of two parts: O^1_{km+1} which are those requests arriving on or before $s(J_{km+1}) + 1 - p/m$, and O^2_{km+1} which are those requests arriving after this time. For O^1_{km+1} we have $|O^1_{km+1}| < r^{m-p}|J_{km+1}|$ since otherwise they would abort J_{km+1}. For O^2_{km+1},

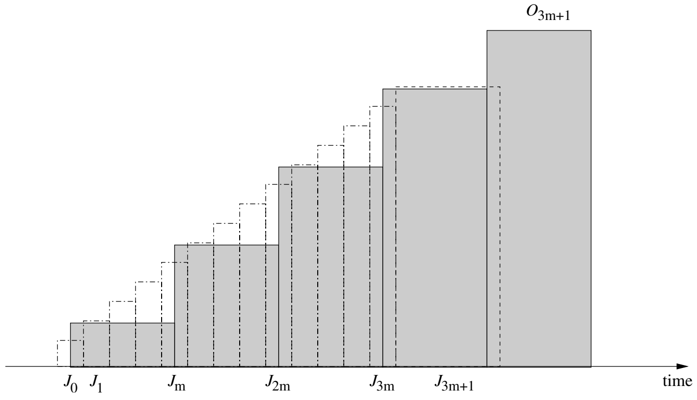


Fig. 1. A basic subschedule with $k = 3, m = 4$. Each rectangle represents an aborted or completed broadcast, and the height of a rectangle represents its profit. Empty rectangles are those of MultiLevel, shaded rectangles are those of OPT.

they must still be pending in M after J_{km+1} completes due to their laxity, and hence the broadcast after J_{km+1} must have profit at least as large as $|O^2_{km+1}|$. We charge their profits to the next basic subschedule. Similarly, this basic subschedule may also receive a charge of profit at most $|J_0|$ from the previous basic subschedule. For O_i where $i = 0, m, 2m, \ldots, km$, we have $|O_i| < r|J_i|$. Finally, requests in J_{km+1} may be satisfied in OPT in some future time, and their profits, which are at most $|J_{km+1}|$ in total, are also charged to this basic subschedule. Without loss of generality we normalize the profits so that $|J_{km+1}| = 1$. Then the total OPT profits charged to this basic subschedule is at most

$$\begin{aligned}
&|J_0| + r(|J_0| + |J_m| + |J_{2m}| + \cdots + |J_{km}|) + r^{m-p}|J_{km+1}| + |J_{km+1}| \\
&\le 1/r^{km+1} + r(1/r^{km+1} + \cdots + 1/r) + r^{m-p} + 1 \\
&= \frac{1}{r^{km+1}} + \frac{1 - 1/r^{km+m}}{1 - 1/r^m} + r^{m-p} + 1 \\
&= \frac{1}{r^{km+1}} - \frac{1}{r^{km+m}(1 - 1/r^m)} + \frac{1}{1 - 1/r^m} + r^{m-p} + 1 \\
&= \frac{1}{\beta^k}\left(\frac{1}{\beta^{1/m}} - \frac{1}{\beta - 1}\right) + \frac{\beta}{\beta - 1} + \beta^{1-p/m} + 1.
\end{aligned}$$

By choosing $\beta^{1/m} = \beta - 1$, the first term in the above expression is zero, i.e., the expression is invariant of the value of k. In this case the competitive ratio is

$$R \le \frac{\beta}{\beta - 1} + \beta^{1-p/m} + 1 = \frac{\beta}{\beta - 1} + \beta^{2-\alpha} + 1.$$

We now remove assumptions (1) and (2). For any basic subschedule that does not satisfy the assumptions, we apply the following transformations to all broadcasts except the last one. First, for any two consecutive broadcasts J, J' in a basic subschedule, if ℓ is the length of J served in the schedule (not the length of the page), we increase the profit of J to $|J'|/\beta^{\ell}$ if it is not already so. The schedule remains valid (all abortion ratios are satisfied) and the profits of OPT will not decrease, since the maximum possible profits of OPT broadcasts at any time (w.r.t. the basic subschedule) will not be decreased. Next, a broadcast of length i/m and profit $|J|$ for $i > 1$ is transformed to i broadcasts of length $1/m$, with profits $|J|, r|J|, r^2|J|, \ldots, r^{i-1}|J|$. Again the schedule remains valid, and the maximum possible OPT profits are not decreased. The transformed schedule satisfies the two assumptions, and the transformation can only increase the competitive ratio. □

3.2 Minimum Laxity $\alpha \geq 2$

Next we consider the case when the laxity is at least 2, i.e. the span of a request is at least twice its length. In this case we use a simple algorithm that never aborts, but is more careful in selecting a page to broadcast after a broadcast completes:

Algorithm EH. When a broadcast completes at time t, set H to be the page with the maximum profit among pending requests, E to be the page with the maximum profit among those pending requests with deadlines before $t+2$. Note that although E is chosen based on those requests with early deadlines, it may also contain requests with late deadlines. Let E' denote the subset of requests for page E with deadlines before $t+2$. Let $\beta = (\sqrt{5}+1)/2$. If $|H| \geq \beta|E'|$, broadcast H, else broadcast E. A page being broadcast is never aborted.

Theorem 2. *For $\alpha \geq 2$, EH is $(\sqrt{5}+3)/2 \approx 2.618$-competitive.*

Proof. We charge the profits obtained by OPT to the broadcasts in EH. Consider a completed broadcast J in EH. Let O be the single page started being broadcast by OPT when EH is broadcasting J. For requests in O that are completed by EH in or before J, we charge their profits to those earlier broadcasts made by EH. Below we only consider requests in O that are still pending in EH. We separate O into two parts: O_1 which are those requests in O having deadlines before $s(J)+2$, and O_2 which are the remaining requests. We distinguish the following cases.

Case 1: $J = H \neq E$. For O_1, they must be released before $s(J)$ by the laxity assumption. We have $|O_1| \leq |E'|$ since E is chosen to be the highest-profit page among those requests with deadlines before $s(J)+2$. So $|O_1| \leq |E'| \leq |J|/\beta$ by the choice of EH. For O_2, since they remain pending in EH when J completes, we charge their profits to the next broadcast after J. Similarly, this J receives a charge C from the previous broadcast, where $|C| \leq |J|$. Also, requests in J may be satisfied later by OPT, which we also charge their profits to this J in EH. Thus the total profits charged to J is at most $|C|+|O_1|+|J| \leq |J|+|J|/\beta+|J| = (2+1/\beta)|J|$.

Case 2: $J = E$. Similar to Case 1 we have $|O_1| \leq |E'|$ and O_2 is charged to the next broadcast. J also receives a charge C from the previous broadcast, where $|C| \leq |H| \leq \beta|E'| \leq \beta|E|$. Requests in E' cannot be started by OPT after $s(J)+1$ since their deadlines is earlier than $s(J)+2$, so J may receive a 'future' charge only from requests in $E - E'$. Thus the total charge to J is at most $|C| + |O_1| + |E - E'| \leq \beta|E| + |E'| + |E - E'| = (1+\beta)|J|$.

By setting $\beta = (\sqrt{5}+1)/2$, the total charge to a broadcast J in any case is at most $\frac{\sqrt{5}+3}{2}|J|$. Hence the algorithm is $(\sqrt{5}+3)/2$-competitive. □

Remark: Note that EH is a non-preemptive algorithm. Since OPT does not use abortions, EH also applies with the same competitive ratio in the nonpreemptive case. This nonpreemptive broadcast problem is also studied in [9] where a 3-competitive algorithm is given.

4 Unit Length Pages: Lower Bounds

4.1 Minimum Laxity $\alpha < 3/2$

We first describe our lower bounds for the unit-length case when the minimum laxity α is smaller than 3/2. Let $R = 4 - \epsilon$ for some arbitrarily small $\epsilon > 0$. Define $lax = \alpha - 1$. Choose a small positive number $d \ll 1 - lax$, and another small positive δ. Define $\delta_i = \delta/2^i$. These are small constants we need to use later. We divide the proof into two steps.

(1) Description of instance construction. The proof is based on a modification of Woeginger's lower bound construction for interval scheduling [13]. In the following each request is asking for a different page. For $0 < v \leq w$ and $d, \delta > 0$, define SET(v, w, d, δ) to be a set of requests $\{j_1, j_2, \dots, j_q\}$ with the following properties:

- The weights $w(j_i)$ of the requests fulfill $w(j_1) = v$, $w(j_q) = w$ and $w(j_i) < w(j_{i+1}) \leq w(j_i) + \delta$ for $1 \leq i \leq q-1$.
- The release time $r(j_i)$ and deadline $d(j_i)$ of the requests fulfill $0 = r(j_1) < r(j_2) < \dots < r(j_q) < d$, $1 + lax = d(j_1) < d(j_2) < \dots < d(j_q) < 1 + lax + d$.
- All requests have length 1 and laxity α.

Since $d(j_q) - r(j_1) < 1 + lax + d < 2$, any algorithm can complete at most one request in each set.

Fix an online algorithm A. At $t = 0$, SET$(v_0, w_0, d_0, \delta_0)$ = SET$(1, R, d, \delta)$ arrives. Define $LST_0 = lax + d$; this is the latest time that A must fix its decision and start broadcasting a page, or else it will fail to meet the deadline of any request. Let $t_0 \leq LST_0$ be the earliest time such that during the time interval $[t_0, LST_0]$, A is broadcasting the same page. That means A fixed its decision at time t_0. Without loss of generality we can assume that A will not switch to broadcast other requests in the same set after this time. Let b_0 be the page being broadcast by A at time LST_0. If $w(b_0) = 1$, no more request arrives. Otherwise, a shifted copy of SET$(v_1, w_1, d_1, \delta_1)$ arrives, where $v_1 = w(b_0)$, $w_1 = Rv_1 - v_1$, $d_1 \ll d$ is smaller than the time between $d(b_0)$ and $d(a_0)$, and a_0 is the request immediately preceding b_0 in the set. The first and last request in this set arrive at times r_1 and r'_1 respectively, where $t_0 + 1 - lax - d_1 < r_1 < r'_1 < t_0 + 1 - lax$. Define LST_1 to be the latest time when A must fix its decision on which request in the second set to broadcast. It is given by $LST_1 = r'_1 + lax$.

The numbers are defined so that they have the following two properties:

– $r_1 > LST_0$: this is because $r_1 > t_0 + 1 - lax - d_1 \geq 1 - lax - d_1$, and $LST_0 = lax + d$, as long as $d_1 < 1 - 2lax - d$ the property holds. Since $lax < 1/2$, d and d_1 can always be chosen small enough.
– $LST_1 < t_0 + 1$: this is because $LST_1 = r'_1 + lax < (t_0 + 1 - lax) + lax$.

The first property ensures that the adversary can decide the weights of the requests in the new set after A fixed its choice of broadcast. The second property ensures that A cannot serve a request in both sets. Hence, if A completes the broadcast of page b_0, it cannot serve any more requests, and no more requests are released. Otherwise, A aborts b_0 and starts a new page in the new set. Then the same process repeats.

In general, suppose A is broadcasting a page in the $(i+1)^{th}$ SET$(v_i, w_i, d_i, \delta_i)$. Let LST_i be the latest time A must fix its decision to broadcast a page in this set; it is given by $LST_i = r'_i + lax$. Let t_i be the earliest time such that in $[t_i, LST_i]$, A is broadcasting the same page. Call this page b_i. If $w(b_i) = v_i$, no more requests are released. Otherwise, a new $(i+2)^{th}$ SET$(v_{i+1}, w_{i+1}, d_{i+1}, \delta_{i+1})$ is released, where $v_{i+1} = w(b_i)$, $w_{i+1} = \max(Rv_{i+1} - \sum_{j=1}^{i+1} v_j, v_i)$, d_{i+1} is the time between $d(b_i)$ and $d(a_i)$, and a_i is the request immediately preceding b_i.The first and last request in this new set arrive at times r_{i+1} and r'_{i+1} respectively, where $t_i + 1 - lax - d_{i+1} < r_{i+1} < r'_{i+1} < t_i + 1 - lax$ (see Fig. 2).

Again we have the following two properties:

– $r_{i+1} > LST_i$: since $r_{i+1} > t_i + 1 - lax - d_{i+1} \geq r_i + 1 - lax - d_{i+1}$, and $LST_i \leq r_i + d_i + lax$, as long as $d_{i+1} < 1 - 2lax - d_i$ this property holds.
– $LST_{i+1} < t_i + 1$: this is because $LST_{i+1} < (t_i + 1 - lax) + lax = t_i + 1$.

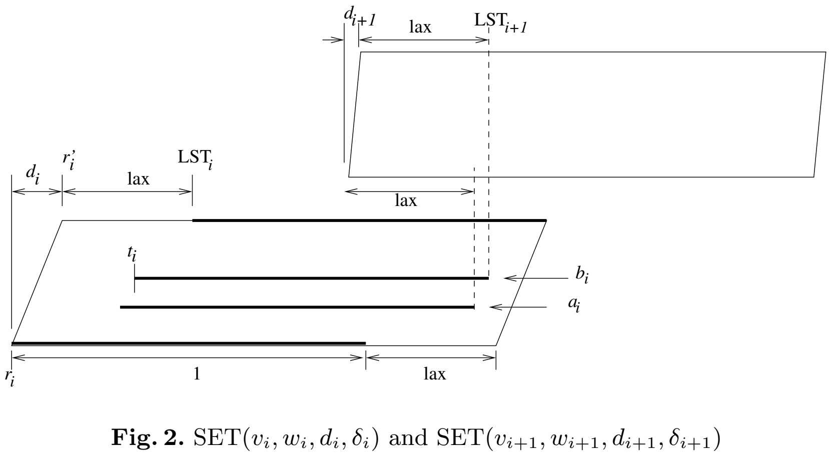


Fig. 2. SET$(v_i, w_i, d_i, \delta_i)$ and SET$(v_{i+1}, w_{i+1}, d_{i+1}, \delta_{i+1})$

A either completes b_i and no more requests are released, or aborts and broadcasts a page in the new set, and then the same process repeats. In the beginning, we have $w_i = Rv_i - \sum_{j=1}^{i} v_j > v_i$, and $v_i < v_{i+1} \leq w_i$, so that the v_i's form an increasing sequence satisfying the recurrence $v_{i+1} \leq Rv_i - \sum_{j=1}^{i} v_j$, and it is shown in [13] that this sequence cannot be an infinite increasing sequence for $R < 4$. Therefore, eventually we have $w_i = v_i > Rv_i - \sum_{j=1}^{i} v_i$ after a finite number of steps. At this final step the set contains only one request with profit v_i. No matter how A chooses, it obtains a profit of v_i. We conclude that A can only serve one request in total.

(2) Profits by the optimal offline algorithm. We now consider the OPT schedule for the instance. Ideally, we want OPT to schedule a request in each set (as in the original construction in [13]), but it may not be possible due to the earlier arrival of the sets (in the original construction a new set only arrives just before the deadlines of the previous set). Recall b_i is the request chosen by A in the $(i+1)$-th set, and a_i is the request that just precedes b_i. Suppose A completes a broadcast of page b_{i-1} in the i-th set, of profit v_i, while attempted to broadcast but aborted $b_0, b_1, \ldots, b_{i-2}$ in all earlier sets. OPT broadcasts $a_0, a_1, \ldots, a_{i-1}$ together with w_i, assuming that is feasible. (If A is forced to serve the single request in the last set the analysis is similar.) Then similar to [13] the total profit by OPT is at least $(4 - 2\epsilon)v_i$, so OPT obtains a profit arbitrarily close to 4 times what A obtains. Below we consider how OPT schedules a subset of these requests.

We resort to figures and defer the formal proof to the full paper. For any two consecutive sets, OPT can serve a_i in the first set together with any request in second set. However, for any three consecutive sets, OPT can only choose requests from two sets (see Fig. 3).

If $\alpha < 4/3$, we can partition the sets into groups of three, and choose the heavier two requests from each group without interfering other groups. (see Fig. 3). Thus OPT obtains a 2/3 of the profits of those requests.

For $4/3 \leq \alpha < 3/2$, this is not always possible, but we can divide the sets into larger groups and apply the same idea. For example, when $\alpha = 1.4$, we divide the sets into groups of five, and it is always possible to serve requests in the second, fourth and fifth set (see Fig. 3). Although we are not selecting

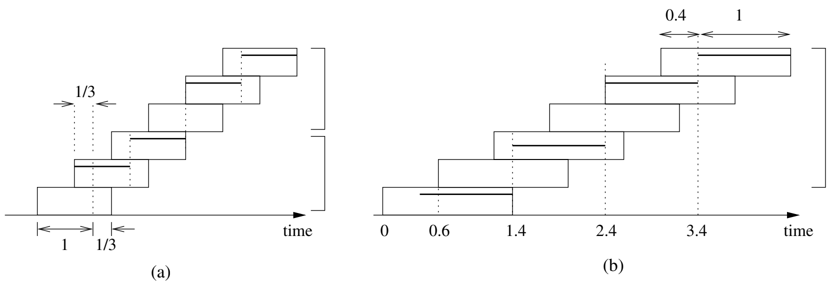


Fig. 3. Each rectangle represents a set. (a) when $lax < 1/3$, choose two sets out of every three. (b) when $lax < 0.4$, choose three sets out of every five.

the heaviest three requests, we can still guarantee a 3/5 fraction of profits, by the following simple observation: for any $0 \leq v_1 \leq v_2 \leq v_3 \leq v_4 \leq v_5$, we have $\frac{v_2+v_4+v_5}{v_1+v_2+v_3+v_4+v_5} \geq \frac{3}{5}$. Hence, we have a lower bound of $4 \times 3/5 = 2.4$. Similarly we can obtain a lower bound of 16/7 when $\alpha < 10/7$, and a lower bound of 2 when $\alpha < 3/2$. In general we have:

Theorem 3. *For any integer $k \geq 1$ and $\alpha < 1 + k/(2k+1) < 3/2$, no deterministic algorithm is better than $4(k+1)/(2k+1)$-competitive.*

4.2 Minimum Laxity $\alpha \geq 2$

Next we consider the case where the minimum laxity is at least 2.

Theorem 4. *For $\alpha \geq 2$, no deterministic algorithm is better than $(1 + 1/\lceil\alpha\rceil)$-competitive.*

Proof. We only consider the case where α is an integer; otherwise we round it up to the nearest integer to obtain the result. All requests in this proof have weight 1 and laxity exactly α. At time 0, α requests each asking for a different page arrive. Without loss of generality assume the online algorithm A broadcasts a page J. Just before it finishes (at time $1-\epsilon$) another request for J arrives. No matter A aborts the current broadcast or not, it can satisfy at most α requests. OPT can broadcast J at time 1 and some other page at time 0, satisfying all $\alpha + 1$ requests. □

In [3] we considered an s-uniform unit job scheduling problem in which each job arrives at integer time, is of unit length, has span exactly s (an integer) and no broadcasting is allowed. We gave a lower bound of $\frac{5}{4} - \Theta(\frac{1}{s})$ for the competitive ratio of this problem.[2] Since the case of broadcast with laxity at least α is a generalized case of unicast with laxity exactly s, where $s \geq \alpha$, which are s-uniform instances, the lower bound carries to the broadcast problem. (Since all time parameters are integers and jobs are of unit length, the ability of abortion in broadcasting needs not be used.) For s sufficiently large the lower bound is arbitrarily close to 5/4. We therefore have:

Theorem 5. *For any laxity α, no deterministic or randomized algorithm can be better than 5/4-competitive.*

The theorem implies that even with arbitrarily large laxity, the competitiveness cannot approach 1. This bound is stronger than that in Theorem 4 for sufficiently large α.

5 Variable Length Pages

Without laxity assumptions, there is a $(e\Delta + e + 1)$-competitive algorithm [14] for the variable-length case, where $\Delta = \lceil$maximum page length/minimum page

[2] The conference version does not contain this result.

length], and $e \approx 2.718$ is the base of the natural logarithm. In this section we give an improved algorithm ACE (for 'Another Completes Earlier'), which makes use of a simple observation: if the broadcast of a page for a newer request has the same or larger profit and an earlier completion time than the page currently being broadcast, we should abort the current page in favour of the newer page.

Algorithm ACE. Let $\beta = 1 + \sqrt{\Delta}$. Let J be the page currently being broadcast. A new request for page J' (together with all pending requests for J' in the pool) will abort J if either one of the following holds: (1) $|J'| \geq \beta|J|$, or (2) $|J'| \geq |J|$ and the completion time of J' (if we start J' now) is earlier than the completion time of J (if we continue to broadcast J). When a broadcast completes, broadcast a page with the maximum profit among pending requests.

Theorem 6. *ACE is* $(\Delta + 2\sqrt{\Delta} + 2)$*-competitive.*

Proof. (Sketch) Without loss of generality assume the shortest page is of length 1 and the longest is of length Δ. Let A be the schedule produced by ACE. As before we divide the schedule into a set of basic subschedules, and we focus on a single basic subschedule. We can assume that all abortions in A are due to condition (1) of the algorithm, and further we assume that all broadcasts made in A are Δ units long (we omit the details in this version of the paper).

Consider a basic subschedule $(J_1, \ldots, J_k)$. We have $|J_i| \leq |J_k|/\beta^{k-i}$ since all abortions are due to condition (1). Consider the broadcasts started by OPT within the broadcast of J_i. There can be at most Δ such broadcasts. If the requests in these broadcasts are completed in A before they are started in OPT, we charge their profits to those earlier broadcasts in A. Now consider those requests that are not completed in A before. For any OPT broadcast O completed before J_i is completed or aborted, we have $|O| < |J_i|$ since otherwise they would abort J_i in A. For an OPT broadcast O that is completed after J_i is completed or aborted, we have $|O| < \beta|J_i|$ or else it would also abort $|J_i|$ in A. In this case no more OPT broadcasts after this O are charged to this J_i.

Therefore, in the worst case, OPT schedules at most $(\Delta - 1)$ length-1 broadcasts each of profit at most $|J_i|$, followed by a length-1 broadcast of profit at most $\beta|J_i|$. Therefore, the total OPT profit corresponding to J_i is at most $(\Delta - 1 + \beta)|J_i|$. Summing over all J_i's, and considering that J_k may be served in OPT later and will be charged to this basic subschedule, we have that the total OPT profits charged to this basic subschedule is at most

$$\sum_{i=1}^{k} (\Delta - 1 + \beta)|J_i| + |J_k| \leq (\Delta - 1 + \beta) \sum_{i=1}^{k} \frac{|J_k|}{\beta^{k-i}} + |J_k| < \left(\frac{\beta(\Delta - 1 + \beta)}{\beta - 1} + 1 \right) |J_k|.$$

Hence the competitive ratio is $\frac{\beta(\Delta-1+\beta)}{\beta-1} + 1$. By setting $\beta = 1 + \sqrt{\Delta}$, this ratio is minimized and is equal to $\Delta + 2\sqrt{\Delta} + 2$. □

In the variable-length case, without assumptions on laxity, there is a lower bound of $\sqrt{\Delta}$ on the competitive ratio. Here we note that laxity may not help much in this case.

Note that laxity is related to resource augmentation using a faster processor, as discussed in [10]. If the online algorithm has a speed-s processor, then the laxity of all jobs become at least s (for the online algorithm). Thus any algorithm for laxity-s instances can be applied. Since OPT does not have this laxity advantage, the competitive ratio in this case will be even smaller (or the same) compared with the case where both the offline and online algorithm receive jobs with laxity (which is the standard case). Therefore the existence of an R-competitive online algorithm for laxity-s instances implies an s-speed R-competitive algorithm for general instances. On the other hand, a lower bound of R on s-speed online algorithm on general instances implies the same lower bound on laxity-s instances.

In [12] it was shown that using resource augmentation does not drastically improve the competitive ratio: no deterministic online algorithm with a constant speedup (can broadcast a constant number of pages more than the offline algorithm) can be constant competitive. In fact, the proof shows something stronger, that no deterministic online algorithm can be better than $O(\sqrt{\Delta})$-competitive with constant speedup.

Therefore, the lower bound for faster processor implies a lower bound of competitiveness with laxity:

Theorem 7. *For constant α, any deterministic online algorithm has competitive ratio $\Omega(\sqrt{\Delta})$.*

References

1. DirecPC Homepage, http://www.direcpc.com.
2. W.-T. Chan, T.-W. Lam, H.-F. Ting and P. W. H. Wong, New results on on-demand broadcasting with deadline via job scheduling with cancellation, in *Proc. 10th COCOON*, LNCS 3106, pages 210–218, 2004.
3. F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall and T. Tichý, Online competitive algorithms for maximizing weighted throughput of unit jobs, *to appear in Journal of Discrete Algorithms*, a preliminary version appeared in Proc. 21st STACS, 2004.
4. M. Chrobak, W. Jawor, J. Sgall and T. Tichý, Online scheduling of equal-length jobs: randomization and restarts help, in *Proc. 31st ICALP*, LNCS 3142, pages 358–370, 2004.
5. B. DasGupta and M. A. Palis, On-line real-time preemptive scheduling of jobs with deadlines on multiple machines, *Journal of Scheduling*, **4**:297–312, 2001.
6. S. A. Goldman, J. Parwatikar and S. Suri, Online scheduling with hard deadlines, *Journal of Algorithms*, **34**(2):370–389, 2000.
7. M. H. Goldwasser, Patience is a virtue: the effect of slack on competitiveness for admission control, *Journal of Scheduling*, **6**:183–211, 2003.
8. H. Hoogeveen, C. N. Potts and G. J. Woeginger, On-line scheduling on a single machine: maximizing the number of early jobs, *Operations Research Letters*, **27**(5):193–197, 2000.
9. R. Y. S. Hung and H. F. Ting, Online scheduling a batch processing system with incompatible job families, 2005, manuscript.

10. B. Kalyanasundaram and K. Pruhs, Speed is as powerful as clairvoyance, *Journal of the ACM*, **47**(4):617–643, 2000.
11. B. Kalyanasundaram and M. Velauthapillai, On-demand broadcasting under deadline, in *Proc. 11th ESA*, LNCS 2832, pages 313–324, 2003.
12. J.-H. Kim and K.-Y. Chwa, Scheduling broadcasts with deadlines, *Theoretical Computer Science*, **325**(3):479–488, 2004.
13. G. J. Woeginger, On-line scheduling of jobs with fixed start and end times, *Theoretical Computer Science*, **130**(1):5–16, 1994.
14. F. Zheng, S. P. Y. Fung, F. Y. L. Chin, C. K. Poon and Y. Xu, Improved on-line broadcast scheduling with deadlines, *Submitted for publication*.

Enforcing and Defying Associativity, Commutativity, Totality, and Strong Noninvertibility for One-Way Functions in Complexity Theory[⋆]

Lane A. Hemaspaandra[1,⋆⋆], Jörg Rothe[2,⋆⋆⋆], and Amitabh Saxena[3,†]

[1] University of Rochester, USA
[2] Heinrich-Heine-Universität Düsseldorf, Germany
rothe@cs.uni-duesseldorf.de
[3] La Trobe University, Australia
asaxena@cs.latrobe.edu.au

Abstract. Rabi and Sherman [RS97,RS93] proved that the hardness of factoring is a sufficient condition for there to exist one-way functions (i.e., p-time computable, honest, p-time noninvertible functions) that are total, commutative, and associative but not strongly noninvertible. In this paper we improve the sufficient condition to $\mathrm{P} \neq \mathrm{NP}$.

More generally, in this paper we completely characterize which types of one-way functions stand or fall together with (plain) one-way functions—equivalently, stand or fall together with $\mathrm{P} \neq \mathrm{NP}$. We look at the four attributes used in Rabi and Sherman's seminal work on algebraic properties of one-way functions (see [RS97,RS93]) and subsequent papers—strongness (of noninvertibility), totality, commutativity, and associativity—and for each attribute, we allow it to be required to hold, required to fail, or "don't care." In this categorization there are $3^4 = 81$ potential types of one-way functions. We prove that each of these 81 feature-laden types stand or fall together with the existence of (plain) one-way functions.

Keywords: Computational complexity, complexity-theoretic one-way functions, associativity, commutativity, strong noninvertibility.

⋆ Work supported in part by the NSF under grant NSF-CCF-0426761 and by the DFG under grant RO 1202/9-1. Work done in part while the first two authors were visiting Julius-Maximilians-Universität Würzburg and while the second author was visiting the University of Rochester.

⋆⋆ Department of Computer Science, University of Rochester, Rochester, NY 14627, USA.

⋆⋆⋆ Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany.

† Department of Computer Science and Computer Engineering, La Trobe University, Bundoora, VIC 3086, Australia.

1 Introduction

1.1 Motivation

In this paper, we study properties of one-way functions, i.e., properties of functions that are easy to compute, but hard to invert. One-way functions are important cryptographic primitives and are the key building blocks in many cryptographic protocols. Various models to capture "noninvertibility" and, depending on the model used, various candidates for one-way functions have been proposed. The notion of noninvertibility is usually based on the average-case complexity model in cryptographic applications, whereas noninvertibility for complexity-theoretic one-way functions is usually defined in the worst-case model. Though the average-case model is very important, we note that even the challenge of showing that any type of one-way function exists in the "less challenging" worst-case model remains an open issue after many years of research. It is thus natural to wonder, as a first step, what assumptions are needed to create various types of complexity-theoretic one-way functions. In this paper, we seek to characterize in terms of class separations this existence issue.

Complexity-theoretic one-way functions of various sorts, and related notions, were studied early on by, for example, Berman [Ber77], Brassard, Fortune, and Hopcroft [BFH78,Bra79], Ko [Ko85], and especially Grollmann and Selman [GS88], and have been much investigated ever since; see, e.g., [AR88,Wat88,HH91,Sel92,RS93,HRW97,RS97,HR99,BHHR99,HR00, HPR01,RH02,FFNR03,HT03,Hom04]. The four properties of one-way functions to be investigated in this paper are strongness, totality, commutativity, and associativity. Intuitively, strong noninvertibility—a notion proposed by Rabi and Sherman [RS97,RS93] and later studied in other papers as well [HR99,HPR01, Hom04]—means that for a two-ary function, given some function value and one of the corresponding arguments, it is hard to determine the other argument. It has been known for decades that one-way functions exist if and only if P $\neq$ NP. But the Rabi-Sherman paper brought out the natural issue of trying to understand what complexity-theoretic assumptions characterized the existence of one-way functions with certain algebraic properties. Eventually, Hemaspaandra and Rothe [HR99] proved that strong, total, commutative, associative one-way functions exist if and only if P $\neq$ NP. (As mentioned earlier, one-way functions with these properties are the key building blocks in Rabi, Rivest, and Sherman's cryptographic protocols for secret-key agreement and for digital signatures (see [RS97,RS93]).)

This paper provides a detailed study of the four properties of one-way functions mentioned above. For each possible combination of possessing, not possessing, and being oblivious to possession of the property, we study the question of whether such one-way functions can exist. Why should one be interested in knowing if a one-way function possesses "negative" properties, such as noncommutativity? On one hand, negative properties can also have useful applications. For example, Saxena, Soh, and Zantidis [SS05,SSZ05] propose authentication protocols for mobile agents and digital cash with signature chaining that use as their key building blocks strong, associative one-way functions for which commutativity in fact is a disadvantage. More generally, it seems natural to try to catalog which types of one-way functions are created by, for example, simply assuming $\mathrm{P} \neq \mathrm{NP}$.

1.2 Summary of Our Results

This paper is organized as follows. In Sections 2 and 3, we formally define the notions and notation used, and we provide some basic lemmas that allow us to drastically reduce the number of cases we have to consider. We will state the full definitions later, but stated merely intuitively, a function is said to be strongly noninvertible if given the output and one argument one cannot efficiently find a corresponding other argument; and a function is said to be strong if it is polynomial-time computable, strongly noninvertible, and satisfies the natural honesty condition related to strong noninvertibility (so-called s-honesty). In Section 4, we prove that the condition $\mathrm{P} \neq \mathrm{NP}$ characterizes all 27 cases induced by one-way functions that are strong. As a corollary, we also obtain a $\mathrm{P} \neq \mathrm{NP}$ characterization of all 27 cases where one requires one-way-ness but is oblivious to whether or not the functions are strong. In Section 5, we consider functions that are required to be one-way but to not be strong. We show that $\mathrm{P} \neq \mathrm{NP}$ characterizes all of these 27 cases. Thus, $\mathrm{P} \neq \mathrm{NP}$ characterizes all 81 cases overall.

Table 1 summarizes our results for the 16 key cases in which each of the four properties considered is either enforced or defied.[1] Definition 2.4 provides the classification scheme used in this table. The left column of Table 1 has 16 quadruples of the form (s, t, c, a), where s regards "strong,", t means "total," c means "commutative," and a means "associative." The variables s, t, c, and a take on a value from $\{\mathrm{Y}, \mathrm{N}\}$, where Y means presence (i.e., "yes"), and N means absence (i.e., "no") of the given property. The center column of Table 1 states the conditions characterizing the existence of (s, t, c, a)-OWFs, and the right column of Table 1 gives the references to the proofs of the results stated.

1.3 General Proof Strategy

We do not attempt to brute-force all 81 cases. Rather, we seek to turn the cases' structure and connectedness against themselves. So, in Section 3 we will reduce the 81 cases to their 16 key cases that do not contain "don't care" conditions. Then, also in Section 3, we will show how to derive the nontotal cases from the total cases, thus further reducing our problem to 8 key cases.

As Corollary 4.6 and, especially, much of Section 5 will show, even among the 8 key cases we share attacks, and find and exploit implications.

Thus, the proof in general consists both of specific constructions—concrete (and in some cases rather difficult to discover) realizations forcing for the first time given patterns of properties—and the framework that minimizes the number of such constructions needed.

[1] In light of the forthcoming Lemma 3.2, those cases in which one is oblivious to whether some property holds follow immediately from the cases stated in Table 1.

Table 1. Summary of results

Properties (s, t, c, a)	Characterization	References/Comments
(N, N, N, N)	P $\neq$ NP	Lemma 5.2 + Lemma 3.4
(N, N, N, Y)	P $\neq$ NP	Lemma 5.5 + Lemma 3.4
(N, N, Y, N)	P $\neq$ NP	Lemma 5.1 + Lemma 3.4
(N, N, Y, Y)	P $\neq$ NP	Lemma 5.5 + Lemma 3.4
(N, Y, N, N)	P $\neq$ NP	[HPR01]; see also Lemma 5.2
(N, Y, N, Y)	P $\neq$ NP	Lemma 5.5
(N, Y, Y, N)	P $\neq$ NP	Lemma 5.1
(N, Y, Y, Y)	P $\neq$ NP	Lemma 5.5
(Y, N, N, N)	P $\neq$ NP	Lemma 4.5 + Lemma 3.4
(Y, N, N, Y)	P $\neq$ NP	Lemma 4.4 + Lemma 3.4
(Y, N, Y, N)	P $\neq$ NP	Lemma 4.3 + Lemma 3.4
(Y, N, Y, Y)	P $\neq$ NP	Lemma 4.2 + Lemma 3.4
(Y, Y, N, N)	P $\neq$ NP	Lemma 4.5
(Y, Y, N, Y)	P $\neq$ NP	Lemma 4.4
(Y, Y, Y, N)	P $\neq$ NP	Lemma 4.3
(Y, Y, Y, Y)	P $\neq$ NP	[HR99]; see also Lemma 4.2

2 Preliminaries and Notations

Fix the alphabet $\Sigma = \{0, 1\}$. The set of strings over Σ is denoted by Σ^*. Let ε denote the empty string. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For any string $x \in \Sigma^*$, let $|x|$ denote the length of x. Let $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ be some standard pairing function, that is, some total, polynomial-time computable bijection that has polynomial-time computable inverses and is nondecreasing in each argument when the other argument is fixed. Let nat : $\Sigma^* \rightarrow \mathbb{N}$ be the standard bijection between Σ^* and $\mathbb{N}$, the set of nonnegative integers, i.e., nat maps the lexicographically ith string in Σ^* to the integer $i - 1$. Let FP denote the class of polynomial-time computable functions (this includes both total and nontotal functions). This paper focuses completely on mappings from $\Sigma^* \times \Sigma^*$ to Σ^* (they are allowed to be many-to-one and they are allowed to be nontotal, i.e., they may map many distinct pairs of strings from $\Sigma^* \times \Sigma^*$ to one and the same string in Σ^*, and they need not be defined for all pairs in $\Sigma^* \times \Sigma^*$). (The study of 2-argument one-way functions of course is needed if associativity and commutativity are to be studied.) For each function f, let domain(f) denote the set of input pairs on which f is defined, and denote the image of f by image(f).

Definition 2.1 presents the standard notion of a (complexity-theoretic, many-one) one-way function, suitably tailored to the case of two-ary functions in the standard way; see [RS93,RS97,HR99,HPR01,Hom04]. (For general introductions to or surveys on one-way functions, see [Sel92], [BHHR99], and [HO02, Chapter 2]. For general background on complexity see, e.g., [HO02,BC93].) Our one-way functions are based on noninvertibility in the worst-case model, as opposed to noninvertibility in the average-case model that is more appealing for cryptographic applications. The notion of honesty in Definition 2.1 below is needed in order to preclude functions from being noninvertible simply due to the trivial reason that some family of images lacks polynomially short preimages.

Definition 2.1 (One-Way Function). *Let σ be a function (it may be either total or nontotal) mapping from $\Sigma^* \times \Sigma^*$ to Σ^*.*

1. *We say σ is* honest *if and only if there exists a polynomial p such that for each $z \in \mathrm{image}(\sigma)$, there exists a pair $(x, y) \in \mathrm{domain}(\sigma)$ such that $\sigma(x, y) = z$ and $|x| + |y| \leq p(|z|)$.*
2. *We say σ is* (polynomial-time) noninvertible *if and only if there exists no function f in* FP *such that for all $z \in \mathrm{image}(\rho)$, we have $\sigma(f(z)) = z$.*
3. *We say σ is a* one-way function *if and only if σ is polynomial-time computable, honest, and noninvertible.*

The four properties of one-way functions that we will study in this paper are strongness, totality, commutativity, and associativity. A function σ mapping from $\Sigma^* \times \Sigma^*$ to Σ^* is said to be *total* if and only if σ is defined for each pair in $\Sigma^* \times \Sigma^*$, and is said to be *nontotal* if it is not total. We say that a function is *partial* if it is either total or nontotal; this says nothing, but makes it clear that we are not demanding that the function be total.

We now define the remaining three properties. Rabi, Rivest, and Sherman (see [RS97,RS93]) introduced the notion of strongly noninvertible associative one-way functions (strong AOWFs, for short). Rivest and Sherman (as attributed in [RS97,RS93]) designed cryptographic protocols for two-party secret-key agreement and Rabi and Sherman designed cryptographic protocols for digital signatures, both of which need strong, total AOWFs as their key building blocks. They also sketch protocols for multiparty secret-key agreement that required strong, total, commutative AOWFs. Strong (and sometimes total and commutative) AOWFs have been intensely studied in [HR99,BHHR99,HPR01,Hom04].

Though Rabi and Sherman's [RS97] notion of associativity is meaningful for total functions, it is not meaningful for nontotal two-ary functions, as has been noted and discussed in [HR99]. Thus, we here follow Hemaspaandra and Rothe's [HR99] notion of associativity, which is appropriate for both total and nontotal two-ary functions, and is designed as an analog to Kleene's 1952 [Kle52] notion of complete equality of partial functions.

Definition 2.2 (Associativity and Commutativity). *Let σ be any partial function mapping from $\Sigma^* \times \Sigma^*$ to Σ^*. Extend Σ^* by $\Gamma = \Sigma^* \cup \{\perp\}$, where $\perp$ is a special symbol indicating, in the usage "$\sigma(x, y) = \perp$," that σ is not defined for the pair (x, y). Define an extension $\widehat{\sigma}$ of σ, which maps from $\Gamma \times \Gamma$ to Γ, by*

$$(2.1) \qquad \widehat{\sigma}(x, y) = \begin{cases} \sigma(x, y) & \text{if } x \neq \perp \text{ and } y \neq \perp \text{ and } (x, y) \in \mathrm{domain}(\sigma) \\ \perp & \text{otherwise.} \end{cases}$$

1. *We say σ is* associative *if and only if for each $x, y, z \in \Sigma^*$, $\widehat{\sigma}(\widehat{\sigma}(x, y), z) = \widehat{\sigma}(x, \widehat{\sigma}(y, z))$.*
2. *We say σ is* commutative *if and only if for each $x, y \in \Sigma^*$, $\widehat{\sigma}(x, y) = \widehat{\sigma}(y, x)$.*

Informally speaking, strong noninvertibility (see [RS97,RS93]) means that even if a function value and one of the corresponding two arguments are given, it is hard to compute the other argument. It is known that, unless P = NP, some noninvertible functions are not strongly noninvertible [HPR01]. And, perhaps counterintuitively, it is known that, unless P = NP, some strongly noninvertible functions are not noninvertible [HPR01]. That is, unless P = NP, strong noninvertibility does not imply noninvertibility. Strong noninvertibility requires a variation of honesty that is dubbed s-honesty in [HPR01]. The notion defined now, as "strong (function)" in Definition 2.3, is in the literature typically called a "strong one-way function." This is quite natural. However, to avoid any possibility of confusion as to when we refer to that and when we refer to the notion of a "one-way function" (see Definition 2.1; as will be mentioned later, neither of these notions necessarily implies the other), we will throughout this paper simply call the notion below "strong" or "a strong function," rather than "strong one-way function."

Definition 2.3 (Strong Function). *Let σ be any partial function mapping from $\Sigma^* \times \Sigma^*$ to Σ^*.*

1. *We say σ is* s-honest *if and only if there exists a polynomial p such that the following two conditions are true:*
   (a) *For each $x, z \in \Sigma^*$ with $\sigma(x, y) = z$ for some $y \in \Sigma^*$, there exists some string $\hat{y} \in \Sigma^*$ such that $\sigma(x, \hat{y}) = z$ and $|\hat{y}| \leq p(|x| + |z|)$.*
   (b) *For each $y, z \in \Sigma^*$ with $\sigma(x, y) = z$ for some $x \in \Sigma^*$, there exists some string $\hat{x} \in \Sigma^*$ such that $\sigma(\hat{x}, y) = z$ and $|\hat{x}| \leq p(|y| + |z|)$.*
2. *We say σ is* (polynomial-time) invertible with respect to the first argument *if and only if there exists an inverter $g_1 \in \mathrm{FP}$ such that for every string $z \in \mathrm{image}(\sigma)$ and for all $x, y \in \Sigma^*$ with $(x, y) \in \mathrm{domain}(\sigma)$ and $\sigma(x, y) = z$,*
$$\sigma(x, g_1(\langle x, z \rangle)) = z.$$
3. *We say σ is* (polynomial-time) invertible with respect to the second argument *if and only if there exists an inverter $g_2 \in \mathrm{FP}$ such that for every string $z \in \mathrm{image}(\sigma)$ and for all $x, y \in \Sigma^*$ with $(x, y) \in \mathrm{domain}(\sigma)$ and $\sigma(x, y) = z$,*
$$\sigma(g_2(\langle y, z \rangle), y) = z.$$
4. *We say σ is* strongly noninvertible *if and only if σ is neither invertible with respect to the first argument nor invertible with respect to the second argument.*
5. *We say σ is* strong *if and only if σ is polynomial-time computable, s-honest, and strongly noninvertible.*

In this paper, we will look at the $3^4 = 81$ categories of one-way functions that one can get by requiring the properties strong/total/commutative/associative to either: hold, fail, or "don't care." For each, we will try to characterize whether such one-way functions exist.

We now define a classification scheme suitable to capture all possible combinations of these four properties of one-way functions.

Definition 2.4 (Classification Scheme for One-Way Functions). *For each $s, t, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$, we say that a partial function $\sigma : \Sigma^* \times \Sigma^* \to \Sigma^*$ is an (s, t, c, a)* one-way function (an (s, t, c, a)-OWF, for short) *if and only if*

1. *σ is a one-way function,*
2. *if $s = \mathrm{Y}$ then σ is strong,*
3. *if $s = \mathrm{N}$ then σ is not strong,*
4. *if $t = \mathrm{Y}$ then σ is a total function,*
5. *if $t = \mathrm{N}$ then σ is a nontotal function,*
6. *if $c = \mathrm{Y}$ then σ is a commutative function,*
7. *if $c = \mathrm{N}$ then σ is a noncommutative function,*
8. *if $a = \mathrm{Y}$ then σ is an associative function, and*
9. *if $a = \mathrm{N}$ then σ is a nonassociative function.*

For example, a function is a $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF exactly if it is a strong, total, commutative, associative one-way function. And note that, under this definition, whenever a setting is $*$, we don't place any restriction as to whether the corresponding property holds or fails to hold—that is, $*$ is a "don't care" designator. For example, a function is a $(*, \mathrm{Y}, *, *)$-OWF exactly if it is a total one-way function. Of course, all $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs are $(*, \mathrm{Y}, *, *)$-OWFs. That is, our 81 classes do not seek to partition, but rather to allow all possible simultaneous settings and "don't care"s for these four properties. However, the 16 such classes with no stars are certainly pairwise disjoint.

3 Groundwork: Reducing the Cases

In this section, we show how to tackle our ultimate goal, stated as Goal 3.1 below, by drastically reducing the number of cases that are relevant among the 81 possible cases.

Goal 3.1. *For each $s, t, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$, characterize the existence of (s, t, c, a)-*OWF*s in terms of some suitable complexity-theoretic condition.*

Since $*$ is a "don't care," for a given $*$ position the characterization that holds with that $*$ is simply the "or" of the characterizations that hold with each of Y and N substituted for the $*$. For example, clearly there exist $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, *)$-OWFs if and only if either there exist $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs or there exist $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWFs. And cases with more than one $*$ can be "unwound" by repeating this. So, to characterize all 81 cases, it suffices to characterize the 16 cases stated in Table 1.

Lemma 3.2. 1. *For each $t, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$, there exist $(*, t, c, a)$-*OWF*s if and only if either there exist (Y, t, c, a)-*OWF*s or there exist (N, t, c, a)-*OWF*s.*
2. *For each $s, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$, there exist $(s, *, c, a)$-*OWF*s if and only if either there exist (s, Y, c, a)-*OWF*s or there exist (s, N, c, a)-*OWF*s.*
3. *For each $s, t, a \in \{\mathrm{Y}, \mathrm{N}, *\}$, there exist $(s, t, *, a)$-*OWF*s if and only if either there exist (s, t, Y, a)-*OWF*s or there exist (s, t, N, a)-*OWF*s.*
4. *For each $s, t, c \in \{\mathrm{Y}, \mathrm{N}, *\}$, there exist $(s, t, c, *)$-*OWF*s if and only if either there exist (s, t, c, Y)-*OWF*s or there exist (s, t, c, N)-*OWF*s.*

It is well known (see [BDG95] and Proposition 1 of [Sel92]) that $\mathrm{P} \neq \mathrm{NP}$ if and only if $(*,*,*,*)$-OWFs exist, i.e., $\mathrm{P} \neq \mathrm{NP}$ if and only if there exist one-way functions, regardless of whether or not they possess any of the four properties. So, in the upcoming proofs, we will often focus on just showing that $\mathrm{P} \neq \mathrm{NP}$ implies the given type of OWF exists.

Lemma 3.3. *For each $s,t,c,a \in \{\mathrm{Y},\mathrm{N},*\}$, if there are (s,t,c,a)-OWFs then $\mathrm{P} \neq \mathrm{NP}$.*

Next, we show that all cases involving nontotal one-way functions can be easily reduced to the corresponding cases involving total one-way functions. Thus, we have eliminated the eight "nontotal" of the remaining 16 cases, provided we can solve the eight "total" cases.

Lemma 3.4. *For each $s,c,a \in \{\mathrm{Y},\mathrm{N}\}$, if there exists an (s,Y,c,a)-OWF, then there exists an (s,N,c,a)-OWF.*

Proof. Fix any $s,c,a \in \{\mathrm{Y},\mathrm{N}\}$, and let σ be any given (s,Y,c,a)-OWF. For each string $w \in \Sigma^*$, let w^+ denote the successor of w in the standard lexicographic ordering of Σ^*, and for each string $w \in \Sigma^+$, let w^- denote the predecessor of w in the standard lexicographic ordering of Σ^*.

Define a function $\rho : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$\rho(x,y) = \begin{cases} (\sigma(x^-,y^-))^+ & \text{if } x \neq \varepsilon \neq y \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that ρ is nontotal, since it is not defined on the pair $(\varepsilon,\varepsilon)$. It is a matter of routine to check that ρ is a one-way function, i.e., polynomial-time computable, honest, and noninvertible. It remains to show that ρ inherits all the other properties from σ as well. To this end, we show the following proposition. The proof of Proposition 3.5 can be found in the full version of this paper [HRS04].

Proposition 3.5. *1. σ is commutative if and only if ρ is commutative.*
2. σ is associative if and only if ρ is associative.
3. σ is strong if and only if ρ is strong.

This completes the proof of Lemma 3.4. ∎

Lemmas 3.2, 3.3, and 3.4 imply that it suffices to deal with only the "total" cases. That is, to achieve Goal 3.1, it would be enough to show that if $\mathrm{P} \neq \mathrm{NP}$ then each of the following eight types of one-way functions exist: $(\mathrm{Y},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWFs, $(\mathrm{Y},\mathrm{Y},\mathrm{Y},\mathrm{N})$-OWFs, $(\mathrm{Y},\mathrm{Y},\mathrm{N},\mathrm{Y})$-OWFs, $(\mathrm{Y},\mathrm{Y},\mathrm{N},\mathrm{N})$-OWFs, $(\mathrm{N},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWFs, $(\mathrm{N},\mathrm{Y},\mathrm{Y},\mathrm{N})$-OWFs, $(\mathrm{N},\mathrm{Y},\mathrm{N},\mathrm{Y})$-OWFs, and $(\mathrm{N},\mathrm{Y},\mathrm{N},\mathrm{N})$-OWFs. In the following sections, we will study each of these cases.

4 Strongness and Being Oblivious to Strongness: $(\mathbf{Y},t,c,a)$-OWFs and $(*,t,c,a)$-OWFs

In this section, we consider the "strong"-is-required cases and those cases where the property of strongness is a "don't care" issue. We start with the 27 "strong" cases. Theorem 4.1 below characterizes each of these cases by the condition $\mathrm{P} \neq \mathrm{NP}$. The proof of Theorem 4.1 follows from the upcoming Lemmas 4.2 through 4.5, via Lemmas 3.2, 3.3, and 3.4.

Theorem 4.1. *For each* $t, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$*, there exist* (Y, t, c, a)-OWF*s if and only if* $\mathrm{P} \neq \mathrm{NP}$.

Lemma 4.2 is already known from Hemaspaandra and Rothe's work [HR99].

Lemma 4.2. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF*s.*

Using Lemmas 3.3 and 4.2, we can exploit the equivalence of $\mathrm{P} \neq \mathrm{NP}$ and the existence of $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs in the upcoming proofs of Lemmas 4.3, 4.4, and 4.5. That is, in these proofs, we start from a strong, total, commutative, associative one-way function.

Lemma 4.3. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWF*s.*

Proof. By Lemmas 3.3 and 4.2, the condition $\mathrm{P} \neq \mathrm{NP}$ is equivalent to the existence of some $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF, call it σ. Recall from the proof of Lemma 3.4 that, in the standard lexicographic ordering of Σ^*, w^+ denotes the successor of $w \in \Sigma^*$ and w^- denotes the predecessor of $w \in \Sigma^+$. We use the following shorthand: For $w \in \Sigma^*$, let $w^{2+} = (w^+)^+$, and for $w \in \Sigma^*$ with $w \notin \{\varepsilon, 0\}$, let $w^{2-} = (w^-)^-$. Define a function $\rho : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$\rho(x, y) = \begin{cases} \varepsilon & \text{if } x = y = 0 \\ 0 & \text{if } x = y = \varepsilon \\ \varepsilon & \text{if } (x = \varepsilon \wedge y = 0) \vee (x = 0 \wedge y = \varepsilon) \\ \varepsilon & \text{if } \{x, y\} \cap \{\varepsilon, 0\} \neq \emptyset \wedge \{x, y\} \cap (\Sigma^* - \{\varepsilon, 0\}) \neq \emptyset \\ (\sigma(x^{2-}, y^{2-}))^{2+} & \text{otherwise.} \end{cases}$$

It is easy to see that ρ is one-way, strong, total, and commutative. This fact can be seen to follow from the construction of ρ and from σ having all these properties. However, ρ is not an associative function, since $\rho(\varepsilon, \rho(\varepsilon, 0)) = 0 \neq \varepsilon = \rho(\rho(\varepsilon, \varepsilon), 0)$.

Thus, ρ is a $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWF. ∎

Lemma 4.4. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{Y}, \mathrm{Y}, \mathrm{N}, \mathrm{Y})$-OWF*s.*

Proof. Assuming $\mathrm{P} \neq \mathrm{NP}$. By Lemma 4.2, let σ be a $(\mathrm{Y}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF. Define a function $\rho : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$\rho(x, y) = \begin{cases} y & \text{if } x, y \in \{0, 1\} \\ (\sigma(x^{3-}, y^{3-}))^{3+} & \text{if } x \notin \{\varepsilon, 0, 1\} \wedge y \notin \{\varepsilon, 0, 1\}) \\ \varepsilon & \text{otherwise,} \end{cases}$$

where we use the following shorthand: Recall from the proof of Lemma 3.4 that, in the standard lexicographic ordering of Σ^*, w^+ denotes the successor of $w \in \Sigma^*$ and w^- denotes the predecessor of $w \in \Sigma^+$. For $w \in \Sigma^*$, let $w^{3+} = ((w^+)^+)^+$, and for $w \in \Sigma^*$ with $w \notin \{\varepsilon, 0, 1\}$, let $w^{3-} = ((w^-)^-)^-$.

It is easy to see, given the fact that σ is a $(\mathrm{Y},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWF, that ρ is a strongly noninvertible, s-honest, total one-way function. However, unlike σ, ρ is noncommutative, since

$$\rho(0,1) = 1 \neq 0 = \rho(1,0).$$

To see that ρ, just like σ, is associative, let three arbitrary strings be given, say a, b, and c. Distinguish the following cases:

Case 1: *Each of a, b, and c is a member of* $\{0,1\}$. Then, associativity follows from the definition of ρ:

$$\rho(a,\rho(b,c)) = \rho(a,c) = c = \rho(b,c) = \rho(\rho(a,b),c).$$

Case 2: *None of a, b, and c is a member of* $\{\varepsilon,0,1\}$. Then the associativity of ρ follows immediately from the associativity of σ. That is,

$$\begin{aligned}\rho(a,\rho(b,c)) &= \rho(a,(\sigma(b^{3-},c^{3-}))^{3+})\\ &= (\sigma(a^{3-},\sigma(b^{3-},c^{3-})))^{3+}\\ &= (\sigma(\sigma(a^{3-},b^{3-}),c^{3-}))^{3+}\\ &= \rho((\sigma(a^{3-},b^{3-}))^{3+},c)\\ &= \rho(\rho(a,b),c).\end{aligned}$$

Note here that both $(\sigma(a^{3-},b^{3-}))^{3+}$ and $(\sigma(b^{3-},c^{3-}))^{3+}$ are strings that are not members of $\{\varepsilon,0,1\}$.

Case 3: *At least one of a, b, and c is not a member of* $\{0,1\}$, *and at least one of a, b, and c is a member of* $\{\varepsilon,0,1\}$. In this case, it follows from the definition of ρ that

$$\rho(a,\rho(b,c)) = \varepsilon = \rho(\rho(a,b),c).$$

Thus, ρ is a $(\mathrm{Y},\mathrm{Y},\mathrm{N},\mathrm{Y})$-OWF. ∎

Lemma 4.5. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there are* $(\mathrm{Y},\mathrm{Y},\mathrm{N},\mathrm{N})$-OWF*s*.

Proof. Assume $\mathrm{P} \neq \mathrm{NP}$. By Lemma 4.2, let σ be a $(\mathrm{Y},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWF. Define a function $\rho : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$\rho(x,y) = \begin{cases} \varepsilon & \text{if } x = y = 0\\ 0 & \text{if } x = y = \varepsilon\\ \varepsilon & \text{if } x = \varepsilon \wedge y = 0\\ 0 & \text{if } x = 0 \wedge y = \varepsilon\\ \varepsilon & \text{if } \{x,y\} \cap \{\varepsilon,0\} \neq \emptyset \wedge \{x,y\} \cap (\Sigma^* - \{\varepsilon,0\}) \neq \emptyset\\ (\sigma(x^{2-},y^{2-}))^{2+} & \text{otherwise.}\end{cases}$$

Again, it follows from the properties of σ and the construction of ρ that ρ is one-way, strong, and total. However, ρ is not commutative, since

$$\rho(\varepsilon,0) = \varepsilon \neq 0 = \rho(0,\varepsilon).$$

Furthermore, ρ is not associative, since

$$\rho(\varepsilon, \rho(\varepsilon, 0)) = 0 \neq \varepsilon = \rho(\rho(\varepsilon, \varepsilon), 0).$$

Thus, ρ is a $(\mathrm{Y}, \mathrm{Y}, \mathrm{N}, \mathrm{N})$-OWF. ∎

Next, we note Corollary 4.6, which follows immediately from Theorem 4.1 via Lemmas 3.2 and 3.3. That is, in light of Lemmas 3.2 and 3.3, Theorem 4.1 provides also a $\mathrm{P} \neq \mathrm{NP}$ characterization of all 27 cases where one requires one-way-ness but is oblivious to whether or not the functions are guaranteed to be strong.

Corollary 4.6. *For each* $t, c, a \in \{\mathrm{Y}, \mathrm{N}, *\}$*, there are* $(*, t, c, a)$-OWF*s if and only if* $\mathrm{P} \neq \mathrm{NP}$.

5 Nonstrongness: (N, t, c, a)-OWFs

It remains to prove the 27 "nonstrong" cases. All 27 have $\mathrm{P} \neq \mathrm{NP}$ as a necessary condition. For each of them, we also completely characterize the existence of such OWFs by $\mathrm{P} \neq \mathrm{NP}$.

First, we consider two "total" and "nonstrong" cases in Lemmas 5.1 and 5.2 below. Note that Hemaspaandra, Pasanen, and Rothe [HPR01] constructed one-way functions that in fact are not strongly noninvertible. Unlike Lemmas 5.1 and 5.2, however, they did not consider associativity and commutativity. Note that, in the proofs of Lemmas 5.1 and 5.2, we achieve "nonstrongness" while ensuring that the functions constructed are s-honest. That is, they are not "nonstrong" because they are not s-honest, but rather they are "nonstrong" because they are not strongly noninvertible.

Lemma 5.1. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWF*s.*

Proof. Assuming $\mathrm{P} \neq \mathrm{NP}$, we define an $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWF that is akin to the one constructed in Theorem 3 of [HPR01]. Define a function $\sigma : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$\sigma(x, y) = \begin{cases} 1\rho(x) & \text{if } x = y \\ 0 \min(x, y) \max(x, y) & \text{if } x \neq y, \end{cases}$$

where $\min(x, y)$ denotes the lexicographically smaller of x and y, $\max(x, y)$ denotes the lexicographically greater of x and y, and $\rho : \Sigma^* \to \Sigma^*$ is a total one-ary one-way function, which exists assuming $\mathrm{P} \neq \mathrm{NP}$. Note that σ is polynomial-time computable, total, honest, and s-honest. Clearly, if σ could be inverted in polynomial time then ρ could be too. Thus, σ is a one-way function. However, although σ is s-honest, it is not strong. To prove that σ is not strongly noninvertible, we show that it is invertible with respect to each of its arguments. Define a function $f_1 : \Sigma^* \to \Sigma^*$ by

$$f_1(a) = \begin{cases} y & \text{if } (\exists x, y, z \in \Sigma^*)\, [a = \langle x, 0z \rangle \wedge z = xy \wedge x <_{\text{lex}} y] \\ y & \text{if } (\exists x, y, z \in \Sigma^*)\, [a = \langle x, 0z \rangle \wedge z = yx \wedge y <_{\text{lex}} x] \\ x & \text{if } (\exists x, z \in \Sigma^*)\, [a = \langle x, 1z \rangle] \\ \varepsilon & \text{otherwise,} \end{cases}$$

where $x <_{\text{lex}} y$ indicates that x is strictly smaller than y in the lexicographic ordering of Σ^*. Note that f_1 is in FP and that f_1 inverts σ with respect to the first argument. Although this is already enough to defy strong noninvertibility of σ, we note that one can analogously show that σ also is invertible with respect to the second argument.

To see that σ is commutative, note that if $x = y$ then $\sigma(x, y) = 1\rho(x) = \sigma(y, x)$, and if $x \neq y$ then $\sigma(x, y) = 0 \min(x, y) \max(x, y) = \sigma(y, x)$. To see that σ is nonassociative, note that $\sigma(\sigma(1, 0), 001) = \sigma(001, 001) = 1\rho(001) \neq 0100001 = \sigma(1, 00001) = \sigma(1, \sigma(0, 001))$.

Thus, σ is an $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{N})$-OWF, which completes the proof. ∎

Lemma 5.2. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{N}, \mathrm{Y}, \mathrm{N}, \mathrm{N})$-OWF*s.*

The proofs of Lemmas 5.2 and 5.3 are omitted here; they can be found in the full version of this paper [HRS04].

Next, we observe that the two remaining "total" and "nonstrong" cases are connected: Lemma 5.3 shows that, given an $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF, one can construct an $(\mathrm{N}, \mathrm{Y}, \mathrm{N}, \mathrm{Y})$-OWF. Thus, by Lemma 3.4, characterizing via $\mathrm{P} \neq \mathrm{NP}$ just the case of $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs will suffice to solve all the four remaining cases (namely, NYYY, NYNY, NNYY, and NNNY) at once.

Lemma 5.3. *If there exist* $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF*s, then there exist* $(\mathrm{N}, \mathrm{Y}, \mathrm{N}, \mathrm{Y})$-OWF*s.*

We now turn to completely characterizing the existence of $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs. A transformation from the literature that might seem to come close to establishing "if $\mathrm{P} \neq \mathrm{NP}$, then $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs exist" has been shown to be flawed unless an unlikely complexity class collapse occurs.[2] However, the following result of Rabi and Sherman does provide evidence that $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWFs indeed exist.

Theorem 5.4. [RS97,RS93] *If factoring is not in polynomial time, then there exist* $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF*s.*

We now improve that sufficient condition to $\mathrm{P} \neq \mathrm{NP}$.

[2] In more detail: Rabi and Sherman [RS93,RS97], assuming $\mathrm{P} \neq \mathrm{NP}$, constructed a *nontotal*, commutative, associative (in a slightly weaker model of associativity for partial functions that completely coincides with our model when speaking of total functions) one-way function that appears to fail to possess strong noninvertibility. They also proposed a construction that they claim can be used to transform every nontotal AOWF whose domain is in P to a total AOWF. However, their claim does not provide an $(\mathrm{N}, \mathrm{Y}, \mathrm{Y}, \mathrm{Y})$-OWF, due to some subtle technical points. First, Rabi and Sherman's construction—even if their claim were valid—is not applicable to the nonstrong, nontotal, commutative AOWF they construct, since this function seems to not have a domain in P. Second, it it is not at all clear that their above-mentioned "construction to add totality" has the properties they assert for it. In particular, let UP as usual denote Valiant's [Val76] class representing "unambiguous polynomial time." Hemaspaandra and Rothe show in [HR99] that any proof that the Rabi–Sherman claim about their transformation's action is in general valid would immediately prove that UP = NP, which is considered unlikely.

Lemma 5.5. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist* $(\mathrm{N},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWF*s and* $(\mathrm{N},\mathrm{Y},\mathrm{N},\mathrm{Y})$-OWF*s.*

Proof. By Lemma 5.3, it suffices to handle the case of $(\mathrm{N},\mathrm{Y},\mathrm{Y},\mathrm{Y})$-OWFs. So, assume $\mathrm{P} \neq \mathrm{NP}$. This implies that there exists a total, one-way function $f : \Sigma^* \to \Sigma^*$. Define the function $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ by

$$g(x,y) = \begin{cases} 0f(a) & \text{if } x = 1a \text{ and } y = 1a \\ \varepsilon & \text{otherwise.} \end{cases}$$

Clearly, g is a one-way function that is total and commutative. Also, g is associative since it is not hard to see that

$$(\forall a,b,c)[g(a,g(b,c)) = g(g(a,b),c) = \varepsilon].$$

Though g is easily seen to be s-honest, g fails to be strongly noninvertible, and so is not strong. In particular, given the output and a purported first argument, here is how to find a second argument consistent with the first argument when one exists. If the output is ε and the purported first argument is z, then output ε as a second argument. If the output is $0y$ and the purported first argument is $1x$, then if $f(x) = y$ a good second argument is $1x$. In every other case, the output and purported first argument cannot have any second argument that is consistent with them, so we safely (though irrelevantly, except for achieving totality of our inverter if one desires that) in this case have our inverter output ε. ∎

Theorem 5.6. *For each* $t,c,a \in \{\mathrm{Y},\mathrm{N},*\}$*, there exist* (N,t,c,a)-OWF*s if and only if* $\mathrm{P} \neq \mathrm{NP}$*.*

The proof of Theorem 5.6 follows immediately from Lemmas 5.1, 5.2, and 5.5, via Lemmas 3.2, 3.3, and 3.4.

In conclusion, this paper studied the question of whether one-way functions can exist, where one imposes either possession, nonpossession, or being oblivious to possession of the properties of strongness, totality, commutativity, and associativity. We have shown that $\mathrm{P} \neq \mathrm{NP}$ is a necessary and sufficient condition in each of the possible 81 cases.

Acknowledgments. This work was done in part while the first two authors were visiting Klaus Wagner's University of Würzburg group, and while the second author was visiting the University of Rochester. We thank the hosts for their hospitality.

References

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.

[BC93] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1993.

[BDG95] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, second edition, 1995.

[Ber77] L. Berman. *Polynomial Reducibilities and Complete Sets.* PhD thesis, Cornell University, Ithaca, NY, 1977.

[BFH78] G. Brassard, S. Fortune, and J. Hopcroft. A note on cryptography and NP $\cap$ coNP $-$ P. Technical Report TR-338, Department of Computer Science, Cornell University, Ithaca, NY, April 1978.

[BHHR99] A. Beygelzimer, L. Hemaspaandra, C. Homan, and J. Rothe. One-way functions in worst-case cryptography: Algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, December 1999.

[Bra79] G. Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, 25(2):232–233, 1979.

[FFNR03] S. Fenner, L. Fortnow, A. Naik, and J. Rogers. Inverting onto functions. *Information and Computation*, 186(1):90–103, 2003.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

[HH91] J. Hartmanis and L. Hemachandra. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163, 1991.

[HO02] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* EATCS Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, 2002.

[Hom04] C. Homan. Tight lower bounds on the ambiguity in strong, total, associative, one-way functions. *Journal of Computer and System Sciences*, 68(3):657–674, 2004.

[HPR01] L. Hemaspaandra, K. Pasanen, and J. Rothe. If P $\neq$ NP then some strongly noninvertible functions are invertible. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, pages 162–171. Springer-Verlag *Lecture Notes in Computer Science #2138*, August 2001.

[HR99] L. Hemaspaandra and J. Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *Journal of Computer and System Sciences*, 58(3):648–659, June 1999.

[HR00] L. Hemaspaandra and J. Rothe. Characterizing the existence of one-way permutations. *Theoretical Computer Science*, 244(1–2):257–261, August 2000.

[HRS04] L. Hemaspaandra, J. Rothe, and A. Saxena. Enforcing and defying associativity, commutativity, totality, and strong noninvertibility for one-way functions in complexity theory. Technical Report TR-854, Department of Computer Science, University of Rochester, Rochester, NY, December 2004. Revised in April, 2005. Also appears as ACM Computing Research Repository (CoRR) Technical Report cs.CC/050304, April 2005.

[HRW97] L. Hemaspaandra, J. Rothe, and G. Wechsung. On sets with easy certificates and the existence of one-way permutations. In *Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 264–275. Springer-Verlag *Lecture Notes in Computer Science #1203*, March 1997.

[HT03] C. Homan and M. Thakur. One-way permutations and self-witnessing languages. *Journal of Computer and System Sciences*, 67(3):608–622, 2003.

[Kle52] S. Kleene. *Introduction to Metamathematics*. D. van Nostrand Company, Inc., New York and Toronto, 1952.

[Ko85] K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.

[RH02] J. Rothe and L. Hemaspaandra. On characterizing the existence of partial one-way permutations. *Information Processing Letters*, 82(3):165–171, May 2002.

[RS93] M. Rabi and A. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical Report CS-TR-3183/UMIACS-TR-93-124, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.

[RS97] M. Rabi and A. Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, 1997.

[Sel92] A. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25(3):203–221, 1992.

[SS05] A. Saxena and B. Soh. A novel method for authenticating mobile agents with one-way signature chaining. In *Proceedings of the 7th International Symposium on Autonomous Decentralized Systems*, pages 187–193. IEEE Computer Society Press, April 2005.

[SSZ05] A. Saxena, B. Soh, and D. Zantidis. A digital cash protocol based on additive zero knowledge. In *Proceedings of the 3rd International Workshop on Internet Communications Security*, pages 672–680. Springer Verlag *Lecture Notes in Computer Science #3482*, May 2005.

[Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

[Wat88] O. Watanabe. On hardness of one-way functions. *Information Processing Letters*, 27(3):151–157, 1988.

Synthesis from Temporal Specifications Using Preferred Answer Set Programming

Stijn Heymans, Davy Van Nieuwenborgh[⋆], and Dirk Vermeir[⋆⋆]

Dept. of Computer Science, Vrije Universiteit Brussel,
VUB Pleinlaan 2, B1050 Brussels, Belgium
{sheymans, dvnieuwe, dvermeir}@vub.ac.be

Abstract. We use extended answer set programming (ASP), a logic programming paradigm which allows for the defeat of conflicting rules, to check satisfiability of computation tree logic (CTL) temporal formulas via an intuitive translation. This translation, to the best of our knowledge the first of its kind for CTL, allows CTL reasoning with existing answer set solvers.

Furthermore, we demonstrate how preferred ASP, where rules are ordered according to preference for satisfaction, can be used for synthesizing synchronization skeletons of processes in a concurrent program from a temporal specification. We argue that preferred ASP is put to good use since a preference order can be used to make explicit some of the decisions tableau algorithms make, e.g. declaratively specifying a preference for maximal concurrency makes synthesis more transparent and thus less error-prone.

1 Introduction

Temporal logics [7] are widely used for expressing properties of nonterminating programs. Transformation semantics, such as *Hoare's logic* are not appropriate here since they depend on the program having a final state that can be verified to satisfy certain properties. Temporal logics on the other hand have a notion of (infinite) time and may express properties of a program along a time line, without the need for that program to terminate. E.g., formulas may express that from each state a program should be able to reach its initial state: $\mathsf{AGEF}\mathit{initial}$.

Two well-known temporal logics are *linear temporal logic (LTL)* [7,20] and *computation tree logic (CTL)* [7,9,4], which basically differ in their interpretation of time: the former assumes that time is linear, i.e. for every state of the program there is only one successor state, while time is branching for the latter, i.e. every state may have different successor states, corresponding to nondeterministic choices for the program.

Another knowledge representation framework is *answer set programming (ASP)* [11,3], a logic programming paradigm with a stable model semantics for negation as failure. A *logic program* corresponds to knowledge one wishes to represent, or, more

[⋆] Supported by the FWO.

[⋆⋆] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

specifically, to an encoding of a particular problem, e.g. a planning problem [17,6]; the *answer sets* of the program then provide its intentional knowledge, or the solutions of the encoded problem, e.g. a plan for a planning problem.

Under the open answer set, there are some programs that do not have any solutions. There are cases, however, where it is not a feasible strategy to have no answer sets at all, e.g. in large modular programs where different modules are contributed by different parties, there could be only 2 modules that contradict each other, although a majority does not. One would then still like to deduce knowledge that is not related to this contradiction (if one module says a and another one says $\neg a$, but both say b, it is reasonable to keep b as a conclusion, while being unsure about a or $\neg a$ – the normal answer set semantics, however, would yield no answers at all). The *extended answer set semantics* [24] solves this by *defeating* rules with *competing* rules, and thus extracts as much knowledge from the program as possible, while providing alternatives for conflicting rules.

We relate the temporal logic CTL to extended ASP by reducing satisfiability checking of CTL formulas to satisfiability checking of predicates w.r.t. a logic program under the extended answer set semantics. To the best of our knowledge, this is the first account of a translation of CTL reasoning to answer set programming. The translation allows for CTL reasoning through existing answer set solvers.

A related approach, i.e. reasoning with temporal logics through ASP, is taken in [12], where bounded model checking of asynchronous concurrent systems is simulated by computing (normal) answer sets of programs. These results are generalized in [13] where bounded model checking for LTL is translated to ASP. Since LTL and CTL are incomparable, i.e. there are LTL formulas for which no equivalent CTL formula exists, and vice versa, the translation in [13] from LTL to ASP is not applicable to the CTL case that we consider here. Another translation of LTL reasoning to ASP can be found in [22,21] in the context of planning with, among others, temporal constraints or goals.

We take the application of ASP to temporal reasoning a step further by considering ASP as a vehicle for the synthesis of synchronization skeletons of processes in concurrent programs, given a CTL specification. In the literature, synthesis from a temporal logic specification is usually done by tableau-like algorithms, e.g. in [8,1] for a CTL specification or in [18] for a LTL specification, or by a reduction to automata as in [16]. We argue that preferred ASP, i.e. ASP where there is a preference on the satisfaction of rules as defined in [24], can make declaratively explicit some implicit decisions made by those tableau algorithms, resulting in a more transparent synthesis method. More specifically, we discuss how to obtain, using preferred ASP, concurrent programs that are as concurrent as the temporal specification allows.

A preferred ASP approach to synthesis has the further advantage that an implementation is available: in order to illustrate the theoretical results, we use the OLPS solver [19], available for download from `http://tinf2.vub.ac.be/olp`, to synthesize the well-known mutual exclusion problem.

The remainder of the paper is organized as follows. In Section 2, we present the extended and preferred answer set semantics. The simulation of CTL reasoning with extended ASP, as well as its complexity, is discussed in Section 3. Before concluding and giving directions for further research in Section 5, we present in Section 4 a synthesis method from a CTL specification using preferred ASP.

2 Preferred Answer Set Programming

We introduce the extended answer set semantics as in [24]. A *term* is a constant or a variable, where the former will be written lower-case and the latter upper-case. An *atom* is of the form $p(t_1, \dots, t_n)$ where p is an n-ary predicate name and t_i, $1 \leq i \leq n$, are terms. A *literal* is an atom a or a classically negated atom $\neg a$; an *extended literal* is a literal l or a literal preceded with the *negation as failure* symbol *not*: $not\ l$. A *program* is a finite set of rules $\alpha \leftarrow \beta$ where α is a set of literals with $|\alpha| \leq 1$, i.e. α is empty or a singleton, and β is a finite set of extended literals. We usually denote a rule as $a \leftarrow \beta$ or $\leftarrow \beta$, and we call the latter a *constraint*. The positive part of the body is $\beta^+ = \{l \mid l \in \beta, l \text{ literal}\}$, the negative part is $\beta^- = \{l \mid not\ l \in \beta\}$, e.g. for $\beta = \{a, not\ \neg b, not\ c\}$, we have that $\beta^+ = \{a\}$ and $\beta^- = \{\neg b, c\}$.

For compactness, we assume that the rule $a(\{x_1, \dots, x_n\}) \leftarrow$ is equivalent with rules $a(x_1) \leftarrow, \dots, a(x_n) \leftarrow$. We may type arguments as in the rule $p(a : t) \leftarrow$ which stands for $p(a) \leftarrow t(a)$.

A *ground* atom, (extended) literal, rule, or program does not contain variables. Substituting every variable in a program P with every possible constant in P yields the ground program $gr(P)$. All following definitions in this section assume ground programs and ground (extended) literals; to obtain the definitions for unground programs, replace every occurrence of a program P by $gr(P)$, e.g. an extended answer set of an unground P is an extended answer set of $gr(P)$.

The *Herbrand Base* $\mathcal{B}_P$ of a program P is the set of all atoms that can be formed using the language of P. For a set X of literals, we take $\neg X = \{\neg l \mid l \in X\}$ where $\neg\neg a$ is a; X is *consistent* if $X \cap \neg X = \emptyset$. Let $\mathcal{L}_P$ be the set of literals that can be formed with P, i.e. $\mathcal{L}_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$. An *interpretation* I of P is any consistent subset of $\mathcal{L}_P$. For a literal l, we write $I \models l$, if $l \in I$, which extends for extended literals $not\ l$ to $I \models not\ l$ if $I \not\models l$. In general, for a set of extended literals X, $I \models X$ if $I \models x$ for every extended literal $x \in X$. A rule $r : a \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models a$ whenever $I \models \beta$, i.e. r is *applied* whenever it is *applicable*. A constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. The set of satisfied rules in P w.r.t. I is the *reduct* P_I.

For a *simple* program P (i.e. a program without *not*), an interpretation I is a *model* of P if I satisfies every rule in P, i.e. $P_I = P$; it is an *answer set* of P if it is a minimal model of P, i.e. there is no model J of P such that $J \subset I$. For programs P containing *not*, the *GL-reduct* w.r.t. an interpretation I is P^I, where P^I contains $\alpha \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P and $\beta^- \cap I = \emptyset$. I is an *answer set* of P if I is an answer set of P^I. A rule $a \leftarrow \beta$ is *defeated* w.r.t. I if there is a *competing* rule $\neg a \leftarrow \gamma$ that is applied w.r.t. I, i.e. $I \models \{\neg a\} \cup \gamma$. An *extended answer set* I of a program P is an answer set of P_I such that all rules in $P \backslash P_I$ are *defeated*. An n-ary predicate p is *satisfiable* w.r.t. a program P iff there is an extended answer set M of P with some $p(x_1, \dots, x_n) \in M$.

Example 1. The knowledge that one either likes karaoke or not (rules r_1 and r_2), that the karaoke bar is on a boat (r_3), that one is afraid of water (r_4), unless there is a boat, and that a boat is usually, but not necessarily, on the water (r_5), can be represented by the following program:

$$\begin{array}{llll} r_1: & \mathit{karaoke} \leftarrow \mathit{not}\ \neg\mathit{karaoke} & r_2: & \neg\mathit{karaoke} \leftarrow \mathit{not}\ \mathit{karaoke} \\ r_3: & \mathit{boat} \leftarrow \mathit{karaoke} & & \\ r_4: & \neg\mathit{water} \leftarrow & r_5: & \mathit{water} \leftarrow \mathit{boat} \end{array}$$

We have the extended answer sets $M_1 = \{\mathit{karaoke}, \mathit{boat}, \mathit{water}\}$, $M_2 = \{\mathit{karaoke}, \mathit{boat}, \neg\mathit{water}\}$, $M_3 = \{\neg\mathit{karaoke}, \neg\mathit{water}\}$, with reducts $P_{M_1} = P\backslash\{r_4\}$, $P_{M_2} = P\backslash\{r_5\}$, and $P_{M_3} = P$. One sees that in M_1 the rule r_4 is defeated by r_5.

Resolving conflicts by defeating rules leads to different alternative extended answer sets, as in Example 1. Usually however, a user may have some particular preferences on the satisfaction of the rules. As in [24], we impose a strict partial order[1] $<$ on the rules in P, indicating these preferences, which results in an *ordered logic program (OLP)* $\langle P, <\rangle$.This preferential ordering will induce an ordering $\sqsubseteq$ among the possible alternative extended answer sets as follows: for interpretations M and N of P, M is "more preferred" than N, denoted $M \sqsubseteq N$, if $\forall r_2 \in P_N\backslash P_M \cdot \exists r_1 \in P_M\backslash P_N \cdot r_1 < r_2$. Intuitively, for every rule that is satisfied by N and not by M, and which thus appears to be a counterexample for M being better than N, there is a better rule that is satisfied by M and not by N, i.e. M can counter the counterexample of N. We have that M is "strictly better" than N, $M \sqsubset N$, if $M \sqsubseteq N$ and not $N \sqsubseteq M$. An extended answer set is a *preferred answer set* of $\langle P, <\rangle$ if it is minimal w.r.t. $\sqsubset$ among the extended answer sets. An n-ary predicate p is *preferred satisfiable* iff there is a preferred answer set M of P with some $p(x_1, \ldots, x_n) \in M$.

Example 2. Considering Example 1, the knowledge that one is afraid of water may result in the preference relation, $r_4 < r_5$. We have then, using $r_4 < r_5$, that $M_2 \sqsubset M_1$, and $M_3 \sqsubset M_1$ and $M_3 \sqsubset M_2$ since M_3 satisfies all rules, such that M_3 is the preferred answer set.

3 CTL Reasoning with Extended Answer Set Programming

Let AP be the finite set of available proposition symbols. *Computation tree logic (CTL)* formulas are defined as follows:

- every proposition symbol $P \in AP$ is a formula,
- if p and q are formulas, so are $p \wedge q$ and $\neg p$,
- if p and q are formulas, then $\mathsf{EX}p$, $\mathsf{E}(p \mathbin{\mathsf{U}} q)$, $\mathsf{AX}p$, and $\mathsf{A}(p \mathbin{\mathsf{U}} q)$ are formulas.

The semantics of a CTL formula is given by *(temporal) structures*. A structure K is a tuple (S, R, L) with S a countable set of states, $R \subseteq S \times S$ a total relation in S, i.e. $\forall s \in S \cdot \exists t \in S \cdot (s, t) \in R$, and $L : S \to 2^{AP}$ a function labeling states with propositions. Intuitively, R indicates the permitted transitions between states and L indicates which propositions are true at certain states.

A path π in K is an infinite sequence of states $(s_0, s_1, \ldots)$ such that $(s_{i-1}, s_i) \in R$ for each $i > 0$. For a path $\pi = (s_0, s_1, \ldots)$, we denote the element s_i with π_i. For a structure $K = (S, R, L)$, a state $s \in S$, and a formula p, we inductively define when K is a *model* of p at s, denoted $K, s \models p$:

[1] A strict partial order on X is an anti-reflexive and transitive relation on X.

- $K, s \models P$ iff $P \in L(s)$ for $P \in AP$,
- $K, s \models \neg p$ iff not $K, s \models p$.
- $K, s \models p \wedge q$ iff $K, s \models p$ and $K, s \models q$.
- $K, s \models \mathsf{EX}p$ iff there is a $(s, t) \in R$ and $K, t \models p$,
- $K, s \models \mathsf{AX}p$ iff for all $(s, t) \in R$, $K, t \models p$,
- $K, s \models \mathsf{E}(p \mathbin{\mathsf{U}} q)$ iff there exists a path π in K with $\pi_0 = s$ and $\exists k \geq 0 \cdot (K, \pi_k \models q \wedge \forall j < k \cdot K, \pi_j \models p)$,
- $K, s \models \mathsf{A}(p \mathbin{\mathsf{U}} q)$ iff for all paths π in K with $\pi_0 = s$ we have $\exists k \geq 0 \cdot (K, \pi_k \models q \wedge \forall j < k \cdot K, \pi_j \models p)$.

$K, s \models \mathsf{EX}p$ ($K, s \models \mathsf{AX}p$) can be read as "there is some neXt state where p holds" ("p holds in all next states"), and $K, s \models \mathsf{E}(p \mathbin{\mathsf{U}} q)$ ($K, s \models \mathsf{A}(p \mathbin{\mathsf{U}} q)$) as "there is some path from s along which p holds Until q holds (and q eventually holds)" ("for all paths from s, p holds until q holds (and q eventually holds)").

Some common abbreviations for CTL formulas are $\mathsf{EF}p = \mathsf{E}(true \mathbin{\mathsf{U}} p)$ (there is some path on which p will eventually hold), $\mathsf{AF}p = \mathsf{A}(true \mathbin{\mathsf{U}} p)$ (p will eventually hold on all paths), $\mathsf{EG}p = \neg\mathsf{AF}\neg p$ (there is some path on which p holds globally), and $\mathsf{AG}p = \neg\mathsf{EF}\neg p$ (p holds everywhere on all paths). Furthermore, we have the standard propositional abbreviations $p \vee q = \neg(\neg p \wedge \neg q)$, $p \Rightarrow q = \neg p \vee q$, and $p \Leftrightarrow q = (p \Rightarrow q) \wedge (q \Rightarrow p)$.

A structure $K = (S, R, L)$ *satisfies* a CTL formula p if there is a state $s \in S$ such that $K, s \models p$; we also call K a *model* of p. A CTL formula p is *satisfiable* iff there is a model of p.

Example 3. Consider the expression of *absence of starvation* $t \Rightarrow \mathsf{AF}c$ [4] for a process in a mutual exclusion problem (more about mutual exclusion in Section 4). The formula demands that if a process tries (t) to enter a critical region, it will eventually succeed in doing so (c) for all possible future execution paths.

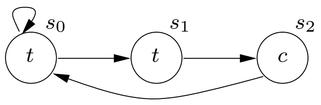


Fig. 1. Example Structure $t \Rightarrow \mathsf{AF}c$

We will usually represent structures by diagrams as in Figure 1, where states are nodes, transitions between nodes define R, and the labels of the nodes contain the propositions true at the corresponding states. The structure K defined by Figure 1 does not satisfy $t \Rightarrow \mathsf{AF}c$ at s_0 since on the path $(s_0, s_0, \ldots)$ c never holds. We have however, $K, s_1 \models t \Rightarrow \mathsf{AF}c$ and $K, s_2 \models t \Rightarrow \mathsf{AF}c$, where the latter holds trivially since $t \notin L(s_2)$.

From a synthesis viewpoint, we are mainly interested in finite structures and thus in n*-satisfiability*, where a CTL formula p is n*-satisfiable* iff there exists a model $K = (S, R, L)$ of p with $|S| = n$, n a non-negative integer. Note that for sufficiently large n, satisfiability is equivalent to n-satisfiability.

Theorem 1 (Small Model Theorem for CTL [7]). *Let p_0 be a CTL formula. Then p_0 is satisfiable iff p_0 has a finite model of size $\leq \exp(\text{length}(p_0))$.*

N-satisfiability of CTL formulas can be reduced to satisfiability of predicates w.r.t. programs under the extended answer set semantics. In order to keep the treatment simple, we will assume that the only allowed temporal constructs are EG, EU, and EX. They are actually adequate in the sense that other temporal constructs can be equivalently, i.e. preserving satisfiability, rewritten using only those three [15]. Before giving the translation of a CTL formula to a program we define the *closure* of a formula, identifying its subformulas. For a formula p, the closure of p is the minimal set $\mathit{clos}(p)$ such that

- $p \in \mathit{clos}(p)$,
- if $\neg q \in \mathit{clos}(p)$, then $q \in \mathit{clos}(p)$,
- if $q \wedge r \in \mathit{clos}(p)$, then $\{q, r\} \subseteq \mathit{clos}(p)$.
- if $\mathsf{EG}q \in \mathit{clos}(p)$, then $q \in \mathit{clos}(p)$,
- if $\mathsf{E}(q \mathbin{\mathsf{U}} r) \in \mathit{clos}(p)$, then $\{q, r\} \subseteq \mathit{clos}(p)$,
- if $\mathsf{EX}q \in \mathit{clos}(p)$, then $q \in \mathit{clos}(p)$.

For a formula p and a non-negative n, we then construct a program consisting of two parts: a generating part G^n and a defining part D_p^n. The program G^n creates n state constants with rule (g_1). The rules (g_2) allow to introduce transitions between states and the rules (g_3) enable any proposition $P \in AP$ to be true at a state or not:

$$\mathit{state}(\{s_0, \dots, s_{n-1}\}) \leftarrow \qquad (g_1)$$
$$\mathit{next}(S : \mathit{state}, N : \mathit{state}) \leftarrow \qquad \neg\mathit{next}(S : \mathit{state}, N : \mathit{state}) \leftarrow \qquad (g_2)$$
$$[P](S : \mathit{state}) \leftarrow \qquad \neg[P](S : \mathit{state}) \leftarrow \qquad (g_3)$$

where $[P]$ is the predicate corresponding to the proposition P. Finally, in order to make the resulting transition relation total, it imposes the restriction that every state should have a successor: $\mathit{succ}(S) \leftarrow \mathit{next}(S, N)$ and $\leftarrow \mathit{state}(S), \mathit{not}\ \mathit{succ}(S)$ (g_4). The program D_p^n introduces for every non-propositional CTL formula in $\mathit{clos}(p)$ the following rules (we write $[q]$ for the predicate corresponding to the CTL formula $q \in \mathit{clos}(p)$):

$$[\neg q](S) \leftarrow \mathit{not}\ [q](S) \qquad (d_1)$$
$$[q \wedge r](S) \leftarrow [q](S), [r](S) \qquad (d_2)$$
$$[\mathsf{EG}q](S) \leftarrow [q](S), \mathit{next}(S, N), [\mathsf{EG}q]^{1}(N) \qquad (d_3^1)$$
$$[\mathsf{EG}q]^{1}(S) \leftarrow [q](S), \mathit{next}(S, N), [\mathsf{EG}q]^{2}(N) \qquad (d_3^2)$$
$$\vdots$$
$$[\mathsf{EG}q]^{n-1}(S) \leftarrow [q](S), \mathit{next}(S, N), [q](N) \qquad (d_3^n)$$
$$[\mathsf{E}(q \mathbin{\mathsf{U}} r)](S) \leftarrow [r](S) \qquad (d_4)$$
$$[\mathsf{E}(q \mathbin{\mathsf{U}} r)](S) \leftarrow [q](S), \mathit{next}(S, N), [\mathsf{E}(q \mathbin{\mathsf{U}} r)](N) \qquad (d_5)$$
$$[\mathsf{EX}q](S) \leftarrow \mathit{next}(S, N), [q](N) \qquad (d_6)$$

The rules ($d_{\{1,2,6\}}$) are direct translations of the CTL semantics. Rules (d_3^i) ensure there is a finite path of at least $n + 1$ nodes along which q holds; there must be a duplicate s_i on this path which can be used to expand the path into an infinite one.

Rules (d_4) and (d_5) are in accordance with the characterization $\mathsf{E}(q\ \mathsf{U}\ r) \equiv r \vee (q \wedge \mathsf{EXE}(q\ \mathsf{U}\ r))$ [7], and make implicit use of the minimality of answer sets to eventually ensure realization of r.

Combining the two programs, we can reduce n-satisfiability checking for CTL formulas to satisfiability of predicates.

Theorem 2. *Let p be a CTL formula. p is n-satisfiable iff $[p]$ is satisfiable w.r.t. $G^n \cup D_p^n$.*

We call satisfiability checking of a CTL formula using the reduction in Theorem 2, *ASP satisfiability checking*.

Example 4. Consider the formula $t \Rightarrow \mathsf{AF}c$ from Example 3. We have that G^n is the program

$$\begin{array}{rlr}
state(\{s_0, \ldots, s_{n-1}\}) \leftarrow & & \\
next(S : state, N : state) \leftarrow & & \neg next(S : state, N : state) \leftarrow \\
[t](S : state) \leftarrow & & \neg[t](S : state) \leftarrow \\
[c](S : state) \leftarrow & & \neg[c](S : state) \leftarrow \\
succ(S) \leftarrow & next(S, N) & \\
\leftarrow & state(S), not\ succ(S) &
\end{array}$$

To obtain the defining part of the program, we first rewrite $t \Rightarrow \mathsf{AF}c$ such that it contains only $\neg, \wedge, \mathsf{EG}, \mathsf{EU}$, and EX using the equivalences: $t \Rightarrow \mathsf{AF}c \equiv \neg t \vee \mathsf{AF}c \equiv \neg t \vee \neg\mathsf{EG}\neg c \equiv \neg(t \wedge \mathsf{EG}\neg c)$. The closure of this last formula is $\{\neg(t \wedge \mathsf{EG}\neg c), t \wedge \mathsf{EG}\neg c, t, \mathsf{EG}\neg c, \neg c, c\}$ such that $D^n_{\neg(t\wedge\mathsf{EG}\neg c)}$ is the program

$$\begin{array}{rl}
[\neg(t \wedge \mathsf{EG}\neg c)](S) \leftarrow & not\ [t \wedge \mathsf{EG}\neg c](S) \\
[\neg c](S) \leftarrow & not\ [c](S) \\
[t \wedge \mathsf{EG}\neg c](S) \leftarrow & [t](S), [\mathsf{EG}\neg c](S) \\
[\mathsf{EG}\neg c](S) \leftarrow & [\neg c](S), next(S, N), [\mathsf{EG}\neg c]^{1}(N) \\
[\mathsf{EG}\neg c]^{1}(S) \leftarrow & [\neg c](S), next(S, N), [\mathsf{EG}\neg c]^{2}(N) \\
\vdots & \\
[\mathsf{EG}\neg c]^{n-1}(S) \leftarrow & [\neg c](S), next(S, N), [\neg c](N)
\end{array}$$

We then have that the CTL formula $t \Rightarrow \mathsf{AF}c$ is n-satisfiable iff the predicate $[\neg(t \wedge \mathsf{EG}\neg c)]$ is satisfiable w.r.t. $G^n \cup D^n_{\neg(t\wedge\mathsf{EG}\neg c)}$.

Satisfiability checking of CTL formulas is in general EXPTIME-complete [7]. Using the ASP-translation yields a NEXPTIME decision procedure.

Theorem 3. *Let p be a CTL formula. ASP satisfiability checking of p is in* NEXPTIME *w.r.t. the size of p.*

Proof. We can reduce reasoning with extended answer sets to the normal answer set semantics by replacing rules $a \leftarrow \beta$ with $a \leftarrow \beta, not\ \neg a$. Intuitively, if the body is true and the rule cannot be defeated, because the negated head is false, one must apply the rule. Define $E(G^n \cup D_p^n)$ as such a transformed program. From Theorem 4 in

[24] we have that the extended answer sets of $G^n \cup D_p^n$ are exactly the answer sets of $E(G^n \cup D_p^n)$.

Thus $[p]$ is satisfiable w.r.t. $G^n \cup D_p^n$ iff there exists an answer set of $E(G^n \cup D_p^n)$ containing some $[p](s_i)$ iff there exists an answer set of $gr(E(G^n \cup D_p^n))$ containing $[p](s_i)$. By [3], the latter can be done by a nondeterministic Turing Machine in time polynomial in the size of $gr(E(G^n \cup D_p^n))$.

The size of $E(G^n \cup D_p^n)$ is exponential w.r.t. the size of p. Indeed, the number of constants n in $E(G^n \cup D_p^n)$ may be exponential w.r.t. the size of p: by Theorem 1, one may need to introduce an exponential number of states to have equivalence of satisfiability and n-satisfiability. Not considering the rules (g_1) that introduce the constants, and taking $|AP|$ constant, one can see that the size of $E((G^n \cup D_p^n)\backslash g_1)$ is linear in the size of p, as is the size of the closure of p.

Grounding does not yield extra complexity, i.e. the size of $gr(E(G^n \cup D_p^n))$ is polynomial in the size of $E(G^n \cup D_p^n)$[2], resulting in a decision procedure that is in NEXPTIME w.r.t. the size of p. □

Provided EXPTIME $\neq$ NEXPTIME, this result would be less optimal than theoretically attainable for satisfiability checking of CTL formulas. As for Description Logics [2], for which rather efficient solvers, such as FACT [14], exist despite the high theoretical complexity, practical cases can be handled by answer set solvers such as DLV [10], SMODELS [23], or OLPS [19]. Note that the translation in [13] of LTL model checking to ASP is essentially in NEXPTIME as well, since only an exponential bound guarantees that [13]'s bounded model checking coincides with model checking.

Another reasoning problem for CTL is the *Branching-Time Model Checking Problem* [7], which involves checking, given a finite structure $K = (S, R, L)$, whether for each state $s \in S$, $K, s \models p$; if this is the case we call K a *branching-time model* of p[3]. As was the case for satisfiability checking, model checking can also be reduced to computing extended answer sets of a program.

For a structure $K = (S, R, L)$, let M_K be the program

$$\begin{array}{rlr} \mathit{state}(\{s_0, \ldots, s_{n-1}\}) \leftarrow & \text{for } S = \{s_0, \ldots, s_{n-1}\} & (m_1) \\ \mathit{next}(s_i, s_j) \leftarrow & \text{for } (s_i, s_j) \in R & (m_2) \\ [P](s_i) \leftarrow & \text{for } P \in L(s_i) & (m_3) \end{array}$$

i.e. M_K adds the facts defining K.

Theorem 4. *Let $K = (S, R, L)$ be a finite structure and p a CTL formula. K is a branching-time model of p iff $M_K \cup D_p^n \cup \{ \leftarrow \mathit{not}\ [p](s_i) \mid s_i \in S\}$ has an (extended) answer set, for $n = |S|$.*

The component $\{ \leftarrow \mathit{not}\ [p](s_i) \mid s_i \in S\}$ ensures that, for each state $s_i \in S$, $[p](s_i)$ is in every answer set, such that p is satisfied at each state.

[2] In general [5], grounding a program may result in an exponential blow-up, however, $E(G^n \cup D_p^n)$ is such that every rule contains at most 2 different variables and thus contributes to at most n^2 ground rules.

[3] Not to confuse with a model of p, which satisfies only one state.

Satisfiability checking of CTL formulas is EXPTIME-complete, but branching-time model checking for CTL can be done in deterministic polynomial time [7]. Similarly, branching-time model checking for CTL via ASP is one exponential level lower than satisfiability checking via ASP, i.e. in NP.

4 Synthesis from a CTL Specification

We recall some definitions and terminology, see e.g. [8]. A *concurrent program* $P = P_1 \parallel \ldots \parallel P_k$ consists of k processes P_i, $1 \leq i \leq k$, that run in parallel, where the parallelism is, typically, simulated by a nondeterministic interleaving of atomic actions of the processes. We represent processes as *synchronization skeletons*, thereby ignoring any details that are irrelevant to the problem of synchronizing the processes. For example, a process may have a state where it executes some critical code. In synchronization problems, we are then not interested in the actual code that is being executed, but more in the questions whether it is allowed, depending on the state of other processes, to enter the critical section, whether the process is executing the critical section, or whether it is not.

Formally, a synchronization skeleton is a finite state diagram consisting of uniquely labeled states, transitions, and *guarded commands* on the transitions. A guarded command is of the form $B \rightarrow A$, where the *guard* B is a predicate over states or *shared variables* and A is the *command* to be executed. Usually, states are subscripted by the index of the process that is in that particular state, e.g. c_1 indicates that process 1 is in the critical section. For example, the following skeleton

$$\begin{tikzcd} t_1 \arrow[r, "\neg c_2 \wedge x = 1 \rightarrow x := 2"] & c_1 \end{tikzcd}$$

may indicate that process 1 can enter the state c_1 from state t_1 if process 2 is currently not in state c_2 and $x = 1$ for the shared variable x; upon entering the state it executes the command by setting x to 2.

The global computation of the concurrent program can then be seen as a flowgraph system [8], where each state $(s_1, \ldots, s_k, x_1, \ldots, x_m)$ encodes the states s_i its constituting processes are currently in, as well as the value x_j of the m shared variables. For a state $(s_1, \ldots, s_i, \ldots, s_k, x_1, \ldots, x_m)$, a possible next state in the computation of the program is $(s_1, \ldots, s'_i, \ldots, s_k, x'_1, \ldots, x'_m)$ if the i-th process has a transition $s_i \rightarrow s'_i$ labeled by $B \rightarrow A$ such that B is true for $(s_1, \ldots, s_i, \ldots, s_k, x_1, \ldots, x_m)$ and $x'_1, \ldots, x'_m$ represent the values of the shared variables after executing A. Intuitively, a computation step consists of nondeterministically selecting an enabled process (one for which the guard B is true[4]), effectively simulating parallelism. A computation of the program is an infinite path in this flowgraph system.

CTL is used to specify the behavior of the concurrent program, i.e. its flowgraph system, as well as part of the behavior of the processes of the program. *Synthesis* is the task, given a CTL specification, to construct the synchronization skeletons of the processes, and in particular the guarded commands, such that the flowgraph system constructed from these processes satisfies the specification.

[4] We assume, as in [8], nonterminating processes such that there is always an enabled process.

We can distinguish 3 phases in the synthesis method: (1) provide the CTL specification, (2) generate a model if the specification is satisfiable, i.e. the flowgraph system, and (3) define the synchronization skeletons from the flowgraph system. In the sequel, we use (preferred) ASP for the second phase of the synthesis method, for more details on the other phases, we refer the reader to, e.g., [8].

We extend the CTL semantics of Section 3 to better suit the concurrent programming paradigm sketched above. As in [8], we define temporal structures as tuples $K = (S, R_1, \ldots, R_k, L)$ for programs consisting of k processes. The definition of satisfaction for such a structure K is as before with $R = R_1 \cup \ldots \cup R_k$. We introduce the temporal operator $\mathsf{X_i}$, $1 \leq i \leq k$, with $K, s \models \mathsf{EX_i}p$ iff there exists a $(s,t) \in R_i$ such that $K, t \models p$, while $K, s \models \mathsf{AX_i}p$ iff $K, s \models \neg\mathsf{EX_i}\neg p$. Intuitively, $K, s \models \mathsf{EX_i}p$ if there is a transition for process i to a state where p holds. The formula $\mathsf{EX}p$ is equivalent with $\mathsf{EX_1}p \vee \ldots \vee \mathsf{EX_k}p$.

Satisfiability checking of such CTL formulas can be reduced to ASP satisfiability checking of predicates w.r.t. a program $G^n_k \cup D^n_{p,k}$ where G^n_k is the program G^n from Section 3 with (g_2) replaced by k sets of rules (g^i_2), $1 \leq i \leq k$,

$$next_i(S : state, N : state) \leftarrow \qquad \neg next_i(S : state, N : state) \leftarrow \qquad (g^i_2)$$

and rules $next(S, N) \leftarrow next_i(S, N)$ (g_5) added.

The rules (g^i_2) enable the introduction of transitions for individual processes; (g_5) defines the union $R = R_1 \cup \ldots \cup R_k$. The closure of a formula p is modified such that if $\mathsf{EX_i}q \in clos(p)$, then $q \in clos(p)$, and if $\mathsf{EX}q \in clos(p)$, then $\{\mathsf{EX_1}q, \ldots, \mathsf{EX_k}q\} \subseteq clos(p)$. We then obtain the defining part $D^n_{p,k}$ by replacing (d_6) with k rules (d^i_6)

$$[\mathsf{EX_i}q](S) \leftarrow next_i(S, N), [q](N) \qquad (d^i_6)$$

Theorem 5. *Let p be a CTL formula. p is n-satisfiable iff $[p]$ is satisfiable w.r.t. $G^n_k \cup D^n_{p,k}$.*

If a CTL formula p is n-satisfiable, we will use the term *flowgraph system* for both a model of p and the corresponding answer set obtained with Theorem 5.

Example 5. Consider 2 synchronization skeletons P_1 and P_2, each modeling the 3 states it can assume: the process can be in the non-critical section of the code (ncs_i, $i \in \{1, 2\}$), it can try to access a critical section of code (try_i), and it can execute the critical section of code (cs_i).

In the *mutual exclusion* problem, one searches for the guarded commands of processes P_1 and P_2 such that they cannot both execute the critical section of the code at the same time. There are many CTL specifications around that model the behavior of the concurrent program executing both processes in parallel, see e.g. [7,4,1,15,18]. We repeat the specification of [8]:

1. Initially, both processes are in their non-critical section: $ncs_1 \wedge ncs_2$.
2. Both processes cannot be in the critical section at the same time (*mutual exclusion*): $\mathsf{AG}\neg(cs_1 \wedge cs_2)$.
3. If a process tries to access its critical section it must always eventually succeed in doing so (*absence of starvation*): $\mathsf{AG}(try_i \Rightarrow \mathsf{AF}cs_i)$.

4. Each process is always in exactly one section: $\mathsf{AG}(ncs_i \vee try_i \vee cs_i)$, $\mathsf{AG}(ncs_i \Rightarrow (\neg try_i \wedge \neg cs_i))$, $\mathsf{AG}(try_i \Rightarrow (\neg ncs_i \wedge \neg cs_i))$, $\mathsf{AG}(cs_i \Rightarrow (\neg ncs_i \wedge \neg try_i))$.
5. If a process is in the non-critical section, it will try to access the critical section in the next step (and it will do nothing else): $\mathsf{AG}(ncs_i \Rightarrow (\mathsf{AX_i}\, try_i \wedge \mathsf{EX_i}\, try_i))$. Note that $\mathsf{EX_i}\, try_i$ is necessary to ensure that there is a next state where try_i holds, $\mathsf{AX_i}\, try_i$ alone would not be sufficient.
6. If a process is trying to access the critical section, then, if it does a move, it will do so into the critical section: $\mathsf{AG}(try_i \Rightarrow \mathsf{AX_i}\, cs_i)$.
7. If a process is in the critical section, it will move to the non-critical section (and it will do nothing else): $\mathsf{AG}(cs_i \Rightarrow (\mathsf{AX_i}\, ncs_i \wedge \mathsf{EX_i}\, ncs_i))$.
8. If process P_j makes a move, process P_i will do nothing, i.e. they are *asynchronous processes*: $\mathsf{AG}(ncs_i \Rightarrow \mathsf{AX_j}\, ncs_i)$, $\mathsf{AG}(try_i \Rightarrow \mathsf{AX_j}\, try_i)$, $\mathsf{AG}(cs_i \Rightarrow \mathsf{AX_j}\, cs_i)$.
9. Some process can always move, i.e. the program is nonterminating: $\mathsf{AGEX}\mathit{true}$.

Let p be the conjunction of the above CTL formulas, rewritten such that it only contains the temporal operators $\mathsf{EX_i}, \mathsf{EU}$, and EG. The program $G_2^9 \cup D_{p,2}^9$, i.e. with 9 states and 2 processes, has then, among others, the two answer sets, or flowgraph systems, in Figure 2 and Figure 3. We listed only propositions in the states, but s_4 and s_5 are different, since they satisfy different temporal formulas.

Figure 2 is the flowgraph system (call it M_g) usually found in literature as a solution to the mutual exclusion problem, but the structure M_b in Figure 3 also satisfies the mutual exclusion specification. M_b only differs from M_g in the missing of transitions (s_1, s_3)

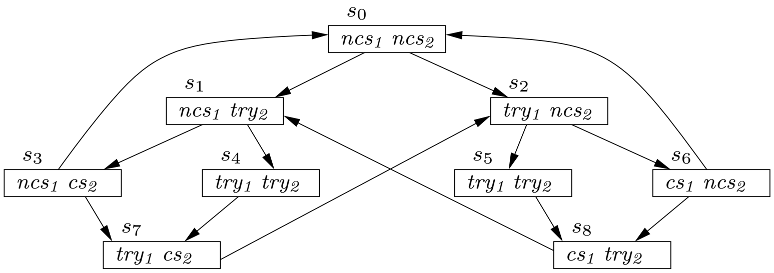


Fig. 2. Maximally Parallel Flowgraph System for Mutual Exclusion

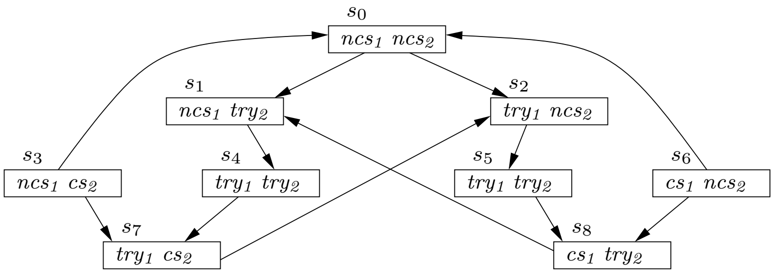


Fig. 3. Flowgraph System for Mutual Exclusion

and (s_2, s_6). It is then natural to wonder why M_g is preferred over M_b as a flowgraph system for the mutual exclusion problem.

The answer is fairly simple. Observing model M_b, one sees that when the system is in the state s_1 its only option is to execute a transition of P_1. This in contrast with M_g where the system can choose between P_1 and P_2. Thus M_g is more nondeterministic, and, by our model of concurrency, allows for more parallelization. That M_b is less parallelized has as a side-effect that P_2 can only enter the critical section if P_1 also tries to enter its critical section, thus, if P_1 decides to do nothing, P_2 is blocked as well. The same scenario cannot occur in M_g since P_2 can go, independently from P_1, from trying to enter to actually entering the critical section.

This drive for more parallelization is implicit in most CTL synthesis methods. E.g., when constructing a model from a tableau, rule [2.2] in [8] says

> *Choose C' to be some $C_j \in Blocks(D_i)$ such that $FRAG[C_i]$ is of minimal size. (Choose one with a maximal number of successors among those C_j with fragments of minimal size, and break ties by choosing the one with lowest index in a predefined ordering.)*

Without going into detail, $FRAG[C_i]$ is a part of the tableau that fulfills eventualities appearing in a node C_i. The relevant part of rule [2.2] for parallelization, as [8] indicates, is choosing nodes of maximal outdegree, since this increases the degree of nondeterministic choice in the model. Instead of leaving this for the model constructing algorithm to take care of, we make this *maximal parallelization* property declaratively explicit.

Definition 1 (Maximal Parallelization Property). *Let p be a CTL formula. A model $M_1 = (S, R_1, \ldots, R_k, L)$ of p is* ***more parallel*** *than a model $M_2 = (S, T_1, \ldots, T_k, L)$ of p, denoted $M_1 \preceq M_2$, if $\forall 1 \le i \le k \cdot T_i \subseteq R_i$. As usual, we have $M_1 \prec M_2$ if $M_1 \preceq M_2$ and not $M_2 \preceq M_1$.*

A model $M_1 = (S, R_1, \ldots, R_k, L)$ of p is ***maximally parallel*** *if it is minimal w.r.t. $\prec$. A CTL formula p is* ***maximally (n-)satisfiable*** *iff p is (n-)satisfiable by a maximally parallel model.*

It is clear that $\preceq$ is a partial order with $\prec$ its strict version. Intuitively, a model M_1 is more parallel than M_2 if they have the same states with the same labeling of the states, but, for each process, the set of transitions of M_2 is a subset of the set of transitions of M_1.

Example 6. We have that M_b is indeed not maximally parallel since $M_g \prec M_b$. Model M_g on the other hand is maximally parallel. Every state in a model of the CTL specification for mutual exclusion has a maximum of 2 outgoing transitions: process P_i, $i \in \{1, 2\}$, in a given state has only one possibility, i.e. from ncs_i to try_i, from try_i to cs_i, and from cs_i to ncs_i. Thus the only candidates in M_g for the inclusion of more transitions are s_4, s_7, s_5, and s_8.

For s_4 the only possible addition is a transition to s_8. However, this violates the absence of starvation property, since we then have an infinite path $(s_1, s_4, s_8, s_1, \ldots)$ without getting into the critical section for process 2. The transitions going out of s_7 can only be extended by a transition to a state $[cs_1\ cs_2]$, violating the mutual exclusion problem. s_5 and s_8 can be treated similarly.

If a CTL formula is satisfiable, it is always maximally satisfiable.

Theorem 6. *Let p be a CTL formula. p is maximally satisfiable iff p is satisfiable.*

As was the case for normal satisfiability, maximal satisfiability is essentially equivalent to maximal n-satisfiability. The proof is similar to the proof from Theorem 6.

Theorem 7. *Let p be a CTL formula. p is maximally satisfiable iff p is maximally n-satisfiable for an n exponential in the size of p.*

Preferred ASP is well-suited for the expression of such a maximal parallelization property. To obtain maximally parallel models, we define the order $<$ on $G_k^n \cup D_{p,k}^n$ such that $next_i(S : state, N : state) < \neg next_j(S : state, N : state)$ for $1 \leq i, j \leq k$. Intuitively, the program $\langle G_k^n \cup D_{p,k}^n, < \rangle$ attempts to introduce as many transitions as possible with the most preferred $next_i(S : state, N : state) \leftarrow$. It only allows defeat of such preferred rules with the less preferred $\neg next_i(S : state, N : state) \leftarrow$ if the CTL formula would otherwise be unsatisfiable. Theorem 7 ensures that we can restrict ourselves to maximal n-satisfiability.

Theorem 8. *Let p be a CTL formula. p is maximally n-satisfiable iff p is preferred satisfiable w.r.t. $\langle G_k^n \cup D_{p,k}^n, < \rangle$.*

Example 7. For the ordered program $\langle G_2^9 \cup D_{p,2}^9, < \rangle$ with $G_2^9 \cup D_{p,2}^9$ as in Example 5 and $<$ defined as in Theorem 8, we obtain the preferred answer set M_g from Figure 2.

For completeness, we briefly describe how [8] obtains synchronization skeletons from the flowgraph system. One first introduces shared variables for every set of propositions that appears more than once as a label of a state, and then one gives a different value to shared variables that represent different states (with the same label), e.g. the label of s_4 in M_g from Example 5 is updated with $TURN = 1$ and s_5 with $TURN = 2$.

Looking at the flowgraph system in Figure 2, one sees that the state transitions for P_1 are $ncs_1 \rightarrow try_1 \rightarrow cs_1 \rightarrow ncs_1 \rightarrow \ldots$ The guards for those transitions are deduced from the flowgraph system: P_1 goes from try_1 to cs_1 in the flowgraph system for global transitions $[try_1\ ncs_2] \rightarrow [cs_1\ ncs_2]$ or $[try_1\ try_2\ TURN = 1] \rightarrow [cs_1\ try_2]$, resulting in a guard $ncs_2 \vee (try_2 \wedge TURN = 1)$ for the transition $try_1 \rightarrow cs_1$ in the synchronization skeleton for P_1 (with empty command). The complete synchronization skeleton for P_1 is shown in Figure 4.

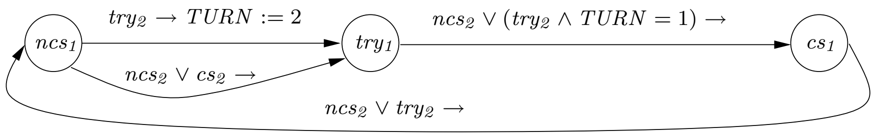

ncs_1
try_1
cs_1
$try_2 \rightarrow TURN := 2$
$ncs_2 \vee cs_2 \rightarrow$
$ncs_2 \vee (try_2 \wedge TURN = 1) \rightarrow$
$ncs_2 \vee try_2 \rightarrow$

Fig. 4. Synchronization Skeleton for Process P_1

5 Conclusions and Directions for Further Research

We reduced CTL reasoning to extended ASP, investigated the complexity, and indicated where and how preferred ASP can be a useful aid in the synthesis of concurrent programs from a CTL specification.

Noting that both LTL [13] and CTL can be caught within an ASP framework, it is interesting to investigate whether reasoning with the more general temporal logic CTL* can be reduced to ASP. Another promising application of preferred ASP lies in the debugging of a proposed synthesis for a specification: one can minimally repair the synthesis by defeating faulty state transitions.

References

1. P. C. Attie and E. A. Emerson. Synthesis of Concurrent Programs for an Atomic Read/Write Model of Computation. *ACM Trans. Program. Lang. Syst.*, 23(2):187–242, 2001.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
4. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.
5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
6. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. In *Proc. of CL 2000*, volume 1861 of *LNCS*, pages 807–821. Springer, 2000.
7. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers B.V., 1990.
8. E. A. Emerson and E. M. Clarke. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Sciene of Computer Programming*, 2(3):241–266, 1982.
9. E. A. Emerson and Joseph Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In *Proc. of the fourteenth annual ACM symposium on Theory of Computing*, pages 169–180. ACM Press, 1982.
10. W. Faber, N. Leone, and G. Pfeifer. Pushing goal derivation in DLP computations. In *Logic Programming and Non-Monotonic Reasoning*, volume 1730 of *LNAI*, pages 177–191. Springer Verslag, 1999.
11. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080, Cambridge, Massachusetts, 1988. MIT Press.
12. K. Heljanko and I. Niemelä. Answer Set Programming and Bounded Model Checking. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming*, pages 90–96. AAAI Press, 2001.
13. K. Heljanko and I. Niemelä. Bounded LTL Model Checking with Stable Models. In *Proc. of LPNMR 2001*, volume 2173 of *LNAI*, pages 200–212. Springer, 2001.
14. I. Horrocks. The FaCT system. In *Proc. of Tableaux'98*, volume 1397 of *LNAI*, pages 307–312. Springer, 1998.
15. M. R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.
16. O. Kupferman and M. Vardi. Synthesis with Incomplete Information. In *Proc. of ICTL 1997*, 1997.

17. V. Lifschitz. Answer Set Programming and Plan Generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
18. Z. Manna and P. Wolper. Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984.
19. D. Van Nieuwenborgh, S. Heymans, and D. Vermeir. An Ordered Logic Program Solver. In *Proc. of PADL 2005*, LNCS. Springer, 2005. To Appear.
20. A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985.
21. T. C. Son and E. Pontelli. Planning with Preferences Using Logic Programming. In *Proc. of LPNMR 2004*, volume 2923 of *LNCS*, pages 247–260. Springer, 2004.
22. T.C. Son, C. Baral, and S. A. McIlraith. Planning with Different Forms of Domain-Dependent Control Knowledge - An Answer Set Programming Approach. In *Proc. of LP-NMR 2001*, volume 2173 of *LNCS*, pages 226–239. Springer, 2001.
23. T. Syrjänen and I. Niemelä. The smodels system. In *Proc. of LPNMR 2001*, volume 2173 of *LNCS*, pages 434–438. Springer, 2001.
24. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *Proc. of JELIA 2002*, volume 2424 of *LNAI*, pages 432–443. Springer, 2002.

Model Checking Strategic Abilities of Agents Under Incomplete Information

Wojciech Jamroga and Jürgen Dix

Department of Computer Science, Clausthal University of Technology,
Julius Albert Str. 4, D-38678 Clausthal, Germany
{wjamroga, dix}@in.tu-clausthal.de

Abstract. In this paper we introduce and study the complexity of model checking alternating-time temporal logic (ATL) with *imperfect information*, using a fine-structured complexity measure. While ATL model checking with *perfect* information is *linear* in the size of the model when the number of agents is considered fixed, this is no longer true when the number of agents is considered parameters of the problem (fine structure).

Combining it with results from our previous papers, we get the surprising result that checking strategic abilities of agents under both perfect and imperfect information *belong to the same complexity class*: both problems are $\mathbf{\Sigma_2^P}$-complete.

Keywords: Computational complexity, multi-agent systems, temporal logic, strategic ability, games with incomplete information.

1 Introduction

The logic of ATL [2,3,4] was originally invented to capture properties of *open computer systems* (such as computer networks), where different components can act autonomously, and computations in such systems are effected by their combined actions. Alternatively, ATL can be seen as a logic for systems involving multiple agents, that allows one to reason about what agents can achieve in game-like scenarios. ATL can be understood as a generalisation of the well-known branching time temporal logic CTL [10,9], in which path quantifiers E ("there is a path") and A ("for every path") are replaced by *cooperation modalities* $\langle\!\langle A \rangle\!\rangle$ that express strategic abilities of agents and their teams.

In ATL, model checking is *linear in the size of the models and formulae* (i.e. in m, l, where m is the number of transitions in the model and l is the length of the formula). This seems to be a very good property, but unfortunately it guarantees less than one could expect. While it is well-known that the number of states in a model can be exponential in the size of a higher-level description of the system, it also turns out that the size of an ATL model is usually *exponential in the number of agents*, even when no higher level description is considered.

In previous work [14], we have shown that this problem is $\mathbf{\Sigma_2^P}$-complete for the ATL semantics based on concurrent game structures, and **NP**-complete for alternating transition systems. In this paper we consider ATL with *imperfect* (or *incomplete*) *information* and extend our results. Since no satisfying semantics based on alternating transition systems has been proposed so far for strategic abilities under incomplete information, we present our results for an extension of concurrent game structures only.

We show that (1) model checking an ATL formula with incomplete information is **NP**-complete in the number of transitions and the length of the formula (i.e. m, l), (2) model checking ATL with incomplete information is $\mathbf{\Sigma_2^P}$-complete in n, d, k, l (k being the number of agents, n number of states, and d maximal number of available decisions per agent per state).

The paper is organised as follows. In Section 2 we introduce ATL and its semantics, based on concurrent game structures. Several variants of ATL are considered and the notions of *perfect* and *imperfect* information in these systems are precisely defined. Section 3 contains our main results: Theorems 1, 2 and Propositions 3, 4. They show, rather surprisingly, that there is no major difference in the complexity between games of perfect and imperfect information. We conclude with Section 4.

2 Strategic Ability for Perfect and Imperfect Information

ATL [2,3,4] is a modal logic that combines the approach of temporal logic (especially the branching-time logic CTL) with elements of game theory. ATL introduces so called *cooperation modalities* $\langle\!\langle A \rangle\!\rangle$, parameterised with a set of agents A. Formula $\langle\!\langle A \rangle\!\rangle\varphi$ expresses the fact that agents A can play collectively to enforce temporal property φ. ATL formulae include temporal operators: "$\bigcirc$" ("in the next state"), $\Box$ ("always from now on") and $\mathcal{U}$ ("until"). Additional operator $\Diamond$ ("sometime") can be defined as $\Diamond\varphi \equiv \top\mathcal{U}\varphi$. Like in CTL, every occurrence of a temporal operator is preceded by exactly one cooperation modality. The broader language of ATL*, in which no such restriction is imposed, is not discussed in detail here.

Formally, the recursive definition of ATL formulae is:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\!\langle A \rangle\!\rangle \bigcirc \varphi \mid \langle\!\langle A \rangle\!\rangle \Box \varphi \mid \langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \varphi$$

A number of different semantics and model classes have been defined for ATL, most of them equivalent (cf. [11,12]). Among these, *concurrent game structures* [4] are probably the most natural and easiest to come up with when modelling concrete problem domains. Moreover, they are the easiest to extend to the incomplete information case, because actions have global identity in them (cf. [13]).

We begin with a short presentation of the ATL semantics based on concurrent game structures (Section 2.1). Then we discuss, in Section 2.2, extending the scope of ATL with the possibility that some agents have incomplete information about the current state of the world. The research on this subject is far from complete, yet a number of ATL extensions have already been proposed to cope with such systems: from the logics of ATEL [19,20] and "ATL with incomplete information" [4] to more sophisticated approaches like ATOL and ATEL-R* [15], ATL$_{ir}$ and ATL$_{iR}$ [18], and ETSL [21]. Among these, ATL$_{ir}$ seems to stand out for its simplicity and conceptual clarity; also (unlike "ATL with incomplete information", ATEL-R* and ATL$_{iR}$), its model checking procedure is decidable. We believe that ATL$_{ir}$ – while probably *not* the definitive ATL extension for games incomplete information (ATOL, for example, is strictly more expressive with the same model checking complexity) – includes constructs that are indispensable when addressing such games. Thus, we treat ATL$_{ir}$ as a kind of "core" ATL-based language for strategic ability under incomplete information, and present its syntax and semantics in Section 2.3.

2.1 Strategic Abilities with Concurrent Game Structures

Models for ATL, *concurrent game structures* (CGS), can be defined as:

$$M = \langle \mathbb{A}\mathrm{gt}, Q, \Pi, \pi, Act, d, o \rangle$$

where $\mathbb{A}\mathrm{gt} = \{a_1, ..., a_k\}$ is the set of all agents or processes (the "grand coalition" of agents), Q is the set of states, Π the set of atomic propositions, $\pi : \Pi \to \mathcal{P}(Q)$ a valuation of propositions, and Act the set of (atomic) actions; function $d : \mathbb{A}\mathrm{gt} \times Q \to \mathcal{P}(Act)$ defines actions available to an agent in a state, and o is the (deterministic) transition function that assigns the outcome state $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to state q and a tuple of actions $\langle \alpha_1, \ldots, \alpha_k \rangle$ that can be executed by $\mathbb{A}\mathrm{gt}$ in q.[1]

A *strategy* of agent a is a conditional plan that specifies what a is going to do in every possible situation (state).[2] Thus, a strategy can be represented with a function $s_a : Q \to Act$, such that $s_a(q) \in d(a, q)$. A *collective strategy* for a group of agents $A = \{a_1, ..., a_r\}$ is simply a tuple of strategies, one per each agent from A. A *path* in M is an infinite sequence of states that can be effected by subsequent transitions, and refers to a possible course of action (or a possible computation) that may occur in the system. Function $out(q, S_A)$ returns the set of all paths that may result from agents A executing strategy S_A from state q onward. Now, the semantics of ATL formulae can be given via the following clauses:

$M, q \models p$ iff $p \in \pi(q)$ (where $p \in \Pi$);
$M, q \models \neg\varphi$ iff $M, q \not\models \varphi$;

[1] This variant of concurrent game structures differs slightly from the original CGS [4]: we represent agents and their actions with symbolic labels, whereas they are represented with natural numbers in the original version.

[2] This is an important deviation from the original semantics of ATL [4], where strategies assign agents' choices to *sequences* of states, which suggests that agents can recall the whole history of each game. In this paper, on the other hand, we employ "memoryless" strategies. While the choice of one or another notion of strategy affects the semantics of the full ATL*, and most ATL variants for games with incomplete information, it should be pointed out that both types of strategies yield equivalent semantics for "pure" ATL [18].

$M, q \models \varphi \vee \psi$ iff $M, q \models \varphi$ or $M, q \models \psi$;
$M, q \models \langle\langle A\rangle\rangle \bigcirc \varphi$ iff there is a collective strategy S_A such that, for every path $\lambda \in out(S_A, q)$, we have $M, \lambda[1] \models \varphi$;
$M, q \models \langle\langle A\rangle\rangle \Box \varphi$ iff there exists S_A such that, for every $\lambda \in out(S_A, q)$, we have $M, \lambda[i] \models \varphi$ for every $i \geq 0$;
$M, q \models \langle\langle A\rangle\rangle \varphi \mathcal{U} \psi$ iff there exist S_A and $i \geq 0$ such that, for every $\lambda \in out(S_A, q)$, we have that $M, \lambda[i] \models \psi$, and $M, \lambda[j] \models \varphi$ for every $0 \leq j < i$.

It is worth pointing out that the CTL path quantifiers A and E can be expressed in ATL in the following way: $\mathsf{A}\varphi \equiv \langle\langle \emptyset \rangle\rangle \varphi$ and $\mathsf{E}\varphi \equiv \langle\langle \mathbb{A}\mathrm{gt} \rangle\rangle \varphi$.

2.2 Complexity of Model Checking

The model checking problem asks, given model M, state q in M, and formula φ, whether φ holds in M, q. Model checking is usually computationally cheaper than satisfiability checking or theorem proving, while often being at least as useful because the designer, user etc. of a system can come up with a precise model of the system behaviour (e.g. a graph with all the actions that may be effected) in many cases. One of the main results concerning ATL states that its formulae can be model-checked in deterministic polynomial time.

Proposition 1. [3,4] *The complexity of* ATL *model checking problem is* PTIME-*complete, and can be done in time* $\mathbf{O}(ml)$*, where m is the number of transitions in the model and l is the length of the formula.*

While the result is certainly attractive, it should be kept in mind that it is only relative to the size of models and formulae, and these can be very large for most application domains. Indeed, it is well known that the number of states in a model is usually exponential in the size of a higher-level description of the problem domain (Boolean variables, for example) for both CTL and ATL models. Moreover, for higher-lever system descriptions, the computation of $\langle\langle A\rangle\rangle \bigcirc$ may require **PSPACE** or even **NEXPTIME** [7,8]. Finally, as we have already pointed out in [14], the complexity of $\mathbf{O}(ml)$ includes a potential intractability even *on the model level* if the number of agents is a parameter of the problem rather than a fixed value.

Remark 1. [4,14] Let n be the number of states in an ATL model M. The number of transitions in M in *not* bounded by n^2, because transitions are labelled with tuples of agents' choices. Let k denote the number of agents, and d the maximal number of available decisions per agent per state. Then, $m = \mathbf{O}(nd^k)$.

Proposition 2. [14]ATL *model checking is* $\mathbf{\Sigma_2^P}$*-complete when n, k, d, l are parameters of the problem.*

2.3 Strategic Abilities Under Incomplete Information

ATL and its models include no way of addressing uncertainty that an agent or a process may have about the current situation; moreover, strategies in ATL can define different choices for any pair of different states, hence implying that an agent can recognise each (global) state of the system, and act accordingly.

Thus, it can be argued that the logic is tailored for describing and analyzing systems in which every agent/process has *complete and accurate knowledge* about the current state of the system. This is usually not the case for most application domains, where a process can access its *local* state, but the state of the environment and the (local) states of other agents can be observed only partially.

One of the main challenges, when a logic of strategic abilities under incomplete information is addressed, is the question of how agents' knowledge should interfere with the agents' available strategies. It has been argued that only *uniform* strategies should be considered; it was also argued that, when reasoning about what an agent can *enforce*, it seems more appropriate to require the agent to know his winning strategy rather than to know only that such a strategy exists [15,18,16]. In this paper, we treat Schobbens' ATL$_{ir}$ and ATL$_{iR}$ [18] as "core", minimal ATL-based languages for strategic ability under incomplete information. The first logic enables reasoning about agents that have no implicit memory of the game (i.e., they use "memoryless" strategies), while the latter is underlain by the assumption that agents can always memorise the whole game. As agents seldom have unlimited memory, and logics of strategic ability with incomplete information and perfect recall are believed to have undecidable model checking, we use ATL$_{ir}$ as *the* logic of strategic ability under uncertainty here.

ATL$_{ir}$ includes the same formulae as ATL, only the cooperation modalities are presented with a subscript: $\langle\!\langle A \rangle\!\rangle_{ir}$ to indicate that they address agents with imperfect *information* and imperfect *recall.* Models of ATL$_{ir}$, *imperfect information concurrent game structures* (*i*CGS), can be presented as concurrent game structures augmented with a family of indistinguishability relations $\sim_a \subseteq Q \times Q$, one per agent $a \in \mathbb{A}\mathrm{gt}$, that describe agents' uncertainty: $q \sim_a q'$ means that, while the system is in state q, agent a considers it possible that it is in q' now. Every $\sim_a$ is assumed to be an equivalence. It is required that agents have the same choices in indistinguishable states: if $q \sim_a q'$ then $d(a, q) = d(a, q')$.

Again, a *strategy* of agent a is a conditional plan that specifies what a is going to do in every possible state. A feasible (deterministic) plan must prescribe the same choices for indistinguishable states. Therefore ATL$_{ir}$ restricts the strategies that can be used by agents to the set of so called uniform strategies. A *uniform strategy* of agent a is defined as a function $s_a : Q \to Act$, such that: (1) $s_a(q) \in d(a, q)$, and (2) if $q \sim_a q'$ then $s_a(q) = s_a(q')$. A *collective strategy* for a group of agents $A = \{a_1, ..., a_r\}$ is a tuple of strategies $S_A = \langle s_{a_1}, ..., s_{a_r} \rangle$, one per each agent from A. A collective strategy is uniform if it contains only uniform individual strategies. A *path* in M is an infinite sequence of states that can be effected by subsequent transitions, and refers to a possible course of action (or a possible computation) that may occur in the system. Again, function $out(q, S_A)$ returns the set of all paths that may result from agents A executing strategy S_A from state q onward.

$out(q, S_A) = \{\lambda = q_0q_1q_2... \mid q_0 = q$ and for every $i = 1, 2, ...$ there exists a tuple of agents' decisions $\langle \alpha_{a_1}^{i-1}, ..., \alpha_{a_k}^{i-1} \rangle$ such that $\alpha_a^{i-1} = S_A(a)(q_{i-1})$ for each $a \in A$, $\alpha_a^{i-1} \in d(a, q_{i-1})$ for each $a \notin A$, and $o(q_{i-1}, \alpha_{a_1}^{i-1}, ..., \alpha_{a_k}^{i-1}) = q_i\}$.

The semantics of cooperation modalities in ATL$_{ir}$ is defined as follows:

$M, q \models \langle\langle A \rangle\rangle_{ir} \bigcirc \varphi$ iff there is a uniform collective strategy S_A such that, for every $a \in A$, q' such that $q \sim_a q'$, and path $\lambda \in out(S_A, q')$, we have $M, \lambda[1] \models \varphi$;

$M, q \models \langle\langle A \rangle\rangle_{ir} \Box \varphi$ iff there exists a uniform S_A such that, for every $a \in A$, q' such that $q \sim_a q'$, and $\lambda \in out(S_A, q')$, we have $M, \lambda[i]$ for every $i \geq 0$;

$M, q \models \langle\langle A \rangle\rangle_{ir} \varphi \mathcal{U} \psi$ iff there exist a uniform strategy S_A and a natural number $i \geq 0$ such that, for every $a \in A$, q' such that $q \sim_a q'$, and $\lambda \in out(S_A, q')$, we have that $M, \lambda[i] \models \psi$, and $M, \lambda[j] \models \varphi$ for every $0 \leq j < i$.

That is, $\langle\langle A \rangle\rangle_{ir} \varphi$ if A have a uniform strategy, such that for every path *that can possibly result from execution of the strategy according to at least one agent from* A, φ is the case.

Remark 2. The CTL universal path quantifier A can be expressed in ATL$_{ir}$ in the following way: $\mathsf{A}\varphi \equiv \langle\langle \varnothing \rangle\rangle_{ir} \varphi$. The existential path quantifier E, however, is not expressible when cooperation modalities quantify over uniform strategies only (cf. [15], Remarks 4.3 and 4.4).

3 Model Checking Strategic Abilities Under Incomplete Information

Schobbens [18] proved that ATL$_{ir}$ model checking is intractable: more precisely, it is **NP**-hard and $\mathbf{\Delta_2^P}$-easy (i.e., can be solved through a polynomial number of calls to an oracle for some problem in **NP**) when the size of the model is defined in terms of the number of transitions. He also conjectured that the problem is probably $\mathbf{\Delta_2^P}$-complete.

The **NP**-hardness follows from a reduction of the well known SAT problem: we construct an imperfect information concurrent epistemic game structure M with states representing clauses and literals inside those clauses. At every "clause" state, the "clause" agent c chooses a transition to a state that represents literal l_i (i.e., either formula p_i or $\neg p_i$) that appears in this clause. "Literal" states for l_i are governed by agent a_i who can declare the underlying proposition p_i true or false. If it makes l_i false then we end up in a "sink" state q_{lose}; if it makes l_i true then the system proceeds to the next "clause" state (or, after the last clause, to state q_{win}). All the states referring to proposition p_i (or its negation) are indistinguishable for agent a_i, and therefore a_i has to make the same decision in all of them. Now, checking satisfiability of the set of clauses is equivalent to model checking of formula $\langle\langle a_1, ..., a_k \rangle\rangle \Diamond \mathsf{win}$ in the initial state of M, where win is a proposition that holds only in state q_{win}. Note that M is turn-based, i.e. at every state there is a single agent that decides upon the next transition. Moreover, it is easy to see that all the "literal" can be in fact governed by the same "literal" agent a: then the SAT problem reduces to model checking of formula $\langle\langle a\rangle\rangle_{ir}\Diamond\mathsf{win}$. Thus, the model checking problem for ATL$_{ir}$ is **NP**-hard even for turn-based models with at most two agents.

The $\mathbf{\Delta_2^P}$-easiness can be demonstrated through the following observation. If the formula to be model checked is of the form $\langle\langle A\rangle\rangle_{ir}\varphi$ (φ being $\bigcirc\psi$, $\Box\psi$ or $\psi_1\,\mathcal{U}\psi_2$), where φ contains no more cooperation modalities, then it is sufficient to guess a strategy for A, "trim" the model by removing all transitions that will never be executed (according to this strategy), and model check CTL formula $\mathsf{A}\varphi$ in the resulting model. Thus, model checking an arbitrary ATL$_{ir}$ formula can be done by checking the subformulae iteratively, which requires a polynomial number of calls to an **NP** algorithm.

Our contribution is twofold. First, we show that ATL$_{ir}$ model checking is in fact **NP**-complete in the number of transitions in the model and the length of the formula. Second, we prove that the problem is $\mathbf{\Sigma_2^P}$-complete in the number of states, agents and decisions (per agent and state) in the model, and the length of the formula, and therefore sits in the same complexity class as model checking strategic abilities in *perfect* information games with respect to these parameters.

3.1 NP-Completeness: Processing All Transitions

In [18], it was shown that an ATL$_{ir}$ formula can be model-checked via a polynomial number of calls to an **NP** oracle: as the size of a (collective) strategy is $\mathbf{O}(m)$, it is sufficient to process the formula recursively, "guessing" the right strategy every time a cooperation modality is encountered. We use a simple trick to show that it is enough to call the oracle only once: all the necessary strategies can be guessed *beforehand.* Note that the size of the witness is still polynomial in this case: more precisely, it is $\mathbf{O}(ml)$.

Theorem 1. *Model checking* ATL$_{ir}$ *is* **NP**-*complete in the number of transitions in the model and the length of the formula.*

Proof. A nondeterministic algorithm that checks formula φ in model M is presented in Figure 1. Calls to $mcheck_{CTL}$ refer to any established CTL model-checker (e.g. [6]). As for the time necessary to carry out the procedure: guessing the strategies can be done in time $\mathbf{O}(ml)$, while "trimming" the model, checking CTL formulae, and getting rid of the states in which agents may not know that the strategy is successful, can all be done in time $\mathbf{O}(m)$ (recursively for subformulae). Thus, the algorithm terminates in time $\mathbf{O}(ml)$. Combining it with the **NP**-hardness result [18], we obtain the theorem.

Note that the exhaustive deterministic algorithm that checks all possible strategies runs in time $\mathbf{O}(nd^{kn}l) = \mathbf{O}(n(m/n)^n l)$.

3.2 The Complexity Refined

One of the problems with model checking formulae of ATL is that the number of transitions m in a model is not bounded by n^2, and can be very large: more precisely, $m = \mathbf{O}(nd^k)$ where n is the number of states, k the number of agents, and

```
function $mcheck(M, \varphi)$;
  Returns the set of states in $M$, in which formula $\varphi$ holds.

■ assign cooperation modalities in $\varphi$ with subsequent numbers $1, ..., c$;
      // note that $c \leq l$
      // we will denote the coalition from the $i$th cooperation modality in $\varphi$ as $\varphi[i]$
■ for every $i = 1, ..., c$, assign the agents in $\varphi[i]$ with numbers $1, ..., k_c$;
      // note that $k_c \leq k$ and $k_c \leq l$
      // we will denote the $j$th agent in $A$ with $A[j]$
■ guess an array $choice$ such that, for every $i = 1, ..., c$, $q \in Q$, and $j = 1, ..., k_c$, we
  have that $choice[i][q][j] \in d_{\varphi[i][j]}(q)$, and for every $q' \in Q$ such that $q \sim_{\varphi[i][j]} q'$
  we have $choice[i][q][j] = choice[i][q'][j]$;
      // at this point, the optimal choices for all coalitions in $\varphi$ are guessed
      // note that the size of $choice$ is $\mathbf{O}(ml)$
      // by $choice|_i$, we will denote the array $choice$ with rows $1, ..., i-1$ removed
■ return $eval(M, \varphi, choice)$;
```

```
function $eval(M, \varphi, choice)$;
  Returns the states in which $\varphi$ holds, given choices for all the coalitions from $\varphi$.
case $\varphi \in \Pi$ : return $\{q \mid \varphi \in \pi(q)\}$;
case $\varphi = \neg\psi$ : return $Q \setminus eval(M, \psi, choice)$;
case $\varphi = \psi_1 \vee \psi_2$ : return $eval(M, \psi_1, choice) \cup eval(M, \psi_2, choice)$;
case $\varphi = \langle\langle A\rangle\rangle T\psi$, where $T = \bigcirc$ or $\Box$ :
  $Q_1 := eval(M, \psi, choice|_2)$;  $M' := trim(M, choice[1])$;
  add to $M'$ new proposition $\overline{\mathsf{p}}$ with $\pi(\overline{\mathsf{p}}) = Q_1$;
  $Q_2 := mcheck_{CTL}(M', \mathsf{A}T\,\overline{\mathsf{p}})$;
  return $\{q \in Q \mid \forall a, q' \,.\, a \in A \wedge q \sim_a q' \Rightarrow q' \in Q_2\}$;
case $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathcal{U} \psi_2$ :
  $c'$ := the number of cooperation modalities in $\psi_1$;
  $Q_1 := eval(M, \psi_1, choice|_2)$;  $Q_2 := eval(M, \psi_2, choice|_{c'+2})$;
  $M' := trim(M, choice[1])$;
  add to $M'$ new propositions $\overline{\mathsf{p}}_1, \overline{\mathsf{p}}_2$ with $\pi(\overline{\mathsf{p}}_1) = Q_1$, $\pi(\overline{\mathsf{p}}_2) = Q_2$;
  $Q_3 := mcheck_{CTL}(M', \mathsf{A}\overline{\mathsf{p}}_1 \mathcal{U} \overline{\mathsf{p}}_2)$;
  return $\{q \in Q \mid \forall a, q' \,.\, a \in A \wedge q \sim_a q' \Rightarrow q' \in Q_3\}$;
end case
```

```
function $trim(M, thischoice)$;
  Returns the CTL model, which includes exactly the transitions that can occur
  when $A$ execute choices from $thischoice$.

■ $\mathcal{R} := \emptyset$;      // the CTL transition relation (contains pairs of states)
■ for each $q \in Q$ and tuple $resp$ of choices from $\mathbb{A}\mathrm{gt} \setminus A$, such that $resp[a] \in d(a, q)$:
  – $q' := o(q, thischoice[q], resp)$;
  – $\mathcal{R} := \mathcal{R} \cup \{\langle q, q'\rangle\}$;
■ return $\langle Q, \mathcal{R}, \Pi, \pi\rangle$;
```

Fig. 1. Nondeterministic algorithm for model checking formulae of ATL$_{ir}$

d the maximal number of decisions per agent per state. Thus, m is exponential in k unless the model is turn-based or the number of agents is fixed. As it turns out, ATL model checking is $\mathbf{\Sigma_2^P} = \mathbf{NP^{NP}}$-complete when n, k, d, l are considered parameters of the problem [14] (i.e. it can be solved by a nondeterministic algorithm that makes calls to an **NP** oracle).

We observe that ATL$_{ir}$ model checking is also $\mathbf{\Sigma_2^P}$-complete when the number of agents is a parameter. To demonstrate that the problem is $\mathbf{\Sigma_2^P}$-hard, we point out that:

Lemma 1. ATL *is semantically subsumed by* ATL$_{ir}$.

Proof. In order to transform a concurrent game structure M to a corresponding imperfect information concurrent game structure M', we fix the indistinguishability relations as the minimal total reflexive relations, (i.e. $\sim_a = \{\langle q,q\rangle \mid q \in Q\}$ for all $a \in \mathbb{A}\mathrm{gt}$), which means that the agents can distinguish between any two states. Let φ be a formula of ATL, and φ' the result of adding subscript ir in every cooperation modality in φ. Then, $M, q \models \varphi$ iff $M', q \models \varphi'$. Thus, ATL model checking can be seen as a special case of ATL$_{ir}$ model checking.

To show that the problem is $\mathbf{\Sigma_2^P}$-easy, we present a refinement of the algorithm from Section 3.1 in Figures 2 and 3.

Theorem 2. *Function mcheck defines a nondeterministic Turing machine that runs in time* $\mathbf{O}(n^2kl)$*, making calls to an* **NP** *oracle. The oracle itself is a nondeterministic Turing machine that runs in time* $\mathbf{O}(n+k)$*. The size of witnesses is never more than* $\mathbf{O}(nkl)$*.*

Proof. The main idea is as follows. First, we guess nondeterministically *all* the strategies for the cooperation modalities that occur in formula φ (we do it beforehand, as in Section 3.1). The strategies must be uniform, so setting $s_a(q)$ fixes automatically $s_a(q')$ for all $q \sim_a q'$. Then we model check φ recursively: for every subformula $\langle\!\langle A\rangle\!\rangle_{ir}\psi$, we assume the respective strategy and check the formula $\langle\!\langle \varnothing\rangle\!\rangle_{ir}\psi$. To do so, we take ATL formula $\langle\!\langle A\rangle\!\rangle\psi$ as input, and employ the standard ATL model checking algorithm from [4] with one important modification: every time function $Pre(A, Q_1)$ is called, it assumes the respective A's choices, and checks whether $q \in Pre(A, Q_1)$ by calling an **NP** oracle (*"is there a response from the opposition in q that leads to a state outside Q_1?"*) and reversing its answer. Note that the latter amounts to checking $M', q \models \langle\!\langle \varnothing\rangle\!\rangle \bigcirc \mathbf{Q_1}$, where M' is model M with A's actions fixed accordingly, and $\mathbf{Q_1}$ is a formula that holds exactly in states Q_1. Finally, we get rid of the states that have indistinguishable counterparts for which the assumed strategy is not successful. Note that, in the middle part of the algorithm, we use an adaptation of the ATL model checking procedure, which *iterates* over states of the system. This kind of iterative solution is possible because $\langle\!\langle \varnothing\rangle\!\rangle_{ir}\psi \equiv \langle\!\langle \varnothing\rangle\!\rangle\psi$ (although, of course, the analogous property does not hold for $\langle\!\langle A\rangle\!\rangle_{ir}$ in general).

The detailed algorithm is shown in Figures 2 and 3. The procedure is very similar to the ATL model checking algorithm from [14] which was used to demonstrate that the problem was $\mathbf{\Sigma_2^P}$-easy for ATL. Analogous complexity analysis applies: first, the number of iterations *within* one single call of function *eval*, as

```
function $mcheck(M, \varphi)$;
  Returns the set of states in $M$, in which formula $\varphi$ holds.

■ assign cooperation modalities in $\varphi$ with subsequent numbers $1, ..., c$;
      // note that $c \leq l$
      // we will denote the coalition from the $i$th cooperation modality in $\varphi$ as $\varphi[i]$
■ for every $i = 1, ..., c$, assign the agents in $\varphi[i]$ with numbers $1, ..., k_c$;
      // note that $k_c \leq k$ and $k_c \leq l$
      // we will denote the $j$th agent in $A$ with $A[j]$
■ guess an array $choice$ such that, for every $i = 1, ..., c$, $q \in Q$, and $j = 1, ..., k_c$, we
  have that $choice[i][q][j] \in d_{\varphi[i][j]}(q)$, and for every $q' \in Q$ such that $q \sim_{\varphi[i][j]} q'$
  we have $choice[i][q][j] = choice[i][q'][j]$;
      // at this point, the optimal choices for all coalitions in $\varphi$ are guessed
      // note that the size of $choice$ is $\mathbf{O}(nkl)$
      // by $choice|_i$, we will denote the array $choice$ with rows $1, ..., i-1$ removed
■ return $eval(M, \varphi, choice)$;
```

```
function $eval(M, \varphi, choice)$;
  Returns the states in which $\varphi$ holds, given choices for all the coalitions from $\varphi$.

case $\varphi \in \Pi$ : return $\{q \mid \varphi \in \pi(q)\}$;
case $\varphi = \neg\psi$ : return $Q \setminus eval(M, \psi, choice)$;
case $\varphi = \psi_1 \vee \psi_2$ : return $eval(M, \psi_1, choice) \cup eval(M, \psi_2, choice)$;
case $\varphi = \langle\langle A \rangle\rangle \bigcirc \psi$ :
  $Q_1 := pre(A, eval(M, \psi, choice|_2), M, choice[1])$;
  return $\{q \in Q \mid \forall a, q' . a \in A \wedge q \sim_a q' \Rightarrow q' \in Q_1\}$;
case $\varphi = \langle\langle A \rangle\rangle \Box \psi$ : $Q_1 := Q$;   $Q_2 := Q_3 := eval(M, \psi, choice|_2)$;
  while $Q_1 \not\subseteq Q_2$ do $Q_1 := Q_1 \cap Q_2$; $Q_2 := pre(A, Q_1, M, choice[1]) \cap Q_3$ od;
  return $\{q \in Q \mid \forall a, q' . a \in A \wedge q \sim_a q' \Rightarrow q' \in Q_1\}$;
case $\varphi = \langle\langle A \rangle\rangle \psi_1 \mathcal{U} \psi_2$ : $c'$ := the number of cooperation modalities in $\psi_1$;
  $Q_1 := \varnothing$;   $Q_2 := eval(M, \psi_1, choice|_2)$;   $Q_3 := eval(M, \psi_2, choice|_{c'+2})$;
  while $Q_3 \not\subseteq Q_1$ do $Q_1 := Q_1 \cup Q_3$; $Q_3 := pre(A, Q_1, M, choice[1]) \cap Q_2$ od;
  return $\{q \in Q \mid \forall a, q' . a \in A \wedge q \sim_a q' \Rightarrow q' \in Q_1\}$;
end case
```

```
function $pre(A, Q_1, M, thischoice)$;
  Returns the set of states, for which the $A$'s choices from $thischoice$ enforce that
  the next state is in $Q_1$, regardless of what agents from $\mathbb{A}\mathrm{gt} \setminus A$ do.

■ $Q_2 := \varnothing$;
■ for each $q \in Q$: if $oracle(A, Q_1, M, thischoice, q) = yes$ then $Q_2 := Q_2 \cup \{q\}$ fi;
■ return $Q_2$;
```

Fig. 2. The model checking algorithm refined (main part)

well as the number of calls to *Pre*, is $\mathbf{O}(n)$; next, function *Pre* runs in $\mathbf{O}(n)$ steps, including calls to the oracle; removing the states for which a member of the coalition can have any doubts can be done in time $\mathbf{O}(n^2k)$; finally, *eval* is called at most $\mathbf{O}(l)$ times. In consequence, we get a nondeterministic polynomial algorithm that makes calls to an **NP** oracle.

```
function $oracle(A, Q_1, M, thischoice, q)$;
  Returns $yes$ if, and only if, the $A$'s choices from $thischoice$ in $q$ enforce that
  the next state is in $Q_1$, regardless of what agents from $\mathbb{A}\mathrm{gt} \setminus A$ do.

■ guess an array $resp$ such that, for every $a \in \mathbb{A}\mathrm{gt} \setminus A$, we have $resp[a] \in d(a, q)$;
      // at this point, the most dangerous response from the opposition is guessed
      // note that the size of $resp$ is $\mathbf{O}(k)$
■ if $o(q, thischoice[q], resp) \in Q_1$ then return $yes$ else return $no$ fi;
```

Fig. 3. The model checking algorithm refined: the oracle

This gives us the following

Corollary 1. *Model checking* ATL *formulae over* CGS *is* $\mathbf{\Sigma_2^P}$*-complete.*

The result has been somewhat surprising to us, since it turns out that a *finer grained* analysis puts checking strategic abilities of agents under imperfect information in the same complexity class as for perfect information games – while the first case appears *strictly* harder than the latter when we approach it from a more "distant" perspective (i.e. when the input parameters are less detailed). Let us recall from [14] that the hardness of model checking ATL is due to simultaneous actions of agents, and can be demonstrated even for scenarios that consist of a single step. It turns out that restricting agents' strategies to uniform strategies only does not increase model checking complexity *enough* to shift it to a higher complexity class. Even the size of witnesses is the same in both cases.

What is different then, that makes model checking of ATL$_{ir}$ *harder than* ATL *in relation to the number of transitions?*

Definitely *not* the number of transitions itself, since CGS can be seen as a special case of *i*CGS. Comparison of model checking complexity for turn-based structures (i.e. structures in which at every state there is a single agent who decides upon the next transition; this can be modelled by requiring that $d(a, q)$ is a singleton for all but one agent) can give us a hint in this respect. Note that, for such structures, $m = \mathbf{O}(nd)$ and we can use the model checking algorithms from Section 3.1 and from [4] to model-check formulae of ATL$_{ir}$ and ATL, respectively.

Proposition 3. *Model checking* ATL$_{ir}$ *over turn-based i*CGS *is* **NP***-complete, while model checking* ATL *over turn-based* CGS *can be done deterministically in time* $\mathbf{O}(ndl)$*. Since* $d \le n$ *for turn-based structures, the latter bound can be replaced by* $\mathbf{O}(n^2 l)$*.*

The result can be generalised to systems in which only a fixed (or bounded) number of agents is acting in each state: we call such systems *semi-turn-based* concurrent game structures. Note that systems with a fixed (or bounded) number of agents are a special case of semi-turn-based CGS.

Proposition 4. *Model checking* ATL$_{ir}$ *over semi-turn-based i*CGS *is* **NP***-complete, while model checking* ATL *over semi-turn-based* CGS *can be done deterministically in time* $\mathbf{O}(n^2 l)$*.*

Moreover, the exhaustive model checking of ATL formulae can be done in time $\mathbf{O}(nd^k l)$, while, for ATL$_{ir}$ formulae, it can be done in $\mathbf{O}(nd^{kn}l)$ steps. This is due to the fact that $\langle\langle A\rangle\rangle\Box\varphi \equiv \varphi \wedge \langle\langle A\rangle\rangle \bigcirc \langle\langle A\rangle\rangle\Box\varphi$ and $\langle\langle A\rangle\rangle\varphi\mathcal{U}\psi \equiv \psi \vee \varphi \wedge \langle\langle A\rangle\rangle \bigcirc \langle\langle A\rangle\rangle\varphi\mathcal{U}\varphi$ in ATL, whereas analogous properties do not hold for ATL$_{ir}$. Thus, successful ATL strategies can be computed incrementally, state by state. Contrary to this, uniform strategies must be considered *as a whole*, which requires much more backtracking if we check the possibilities exhaustively.

Nevertheless, we believe that the result, presented in this section, sets a glimmer of hope for agent logics that include incomplete information in their scope. If ATL formulae can be feasibly model-checked then agents with incomplete information are not *that* far away. And there already exist running model-checkers for ATL [5,1], based on Ordered Binary Decision Diagrams. Also, new model checking techniques, based on the idea of *Unbounded Model Checking*, are under development [17].

4 Conclusions

This paper contains two main results:

- model checking of ATL$_{ir}$ formulae is in **NP** and hence **NP**-complete (closing a gap in previous work of Schobbens);
- model checking of ATL$_{ir}$ formulae is $\mathbf{\Sigma_2^P}$-complete in the number of states, agents and decisions (per agent and state) in the model, and the length of the formula.

In other words, checking strategic ability under imperfect information falls in the same complexity class as checking strategic ability for *perfect* information agents, when a more refined analysis is conducted – which we consider somewhat surprising. This sets a glimmer of hope for agent logics that include incomplete information in their scope: if ATL formulae can be feasibly model-checked (and there already exist model-checkers for ATL [5,1]), then agents with incomplete information are not *that* far away.

Finally, we point out that the difference between the perfect and imperfect information case lies in modularity of strategies with respect to the property that the agents may want to enforce. For perfect information games, potential successfulness of sub-strategies is more independent and they can be computed (or guessed) incrementally, while imperfect information strategies refuse incremental analysis.

The first author would like to thank Rafał Somla for the discussions about model checking ATL and its various extensions.

References

1. R. Alur, L. de Alfaro, R. Grossu, T.A. Henzinger, M. Kang, C.M. Kirsch, R. Majumdar, F.Y.C. Mang, and B.-Y. Wang. jMocha: A model-checking tool that exploits design structure. In *Proceedings of ICSE*, pages 835–836. IEEE Computer Society Press, 2001.

2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 100–109. IEEE Computer Society Press, 1997.
3. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Lecture Notes in Computer Science*, 1536:23–60, 1998.
4. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
5. R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA user manual. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525, 1998.
6. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
7. L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In *Proceedings of CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 458–473, 2000.
8. L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems, part II. In *Proceedings of CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 566–580, 2001.
9. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.
10. E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. In *Proceedings of the Annual ACM Symposium on Principles of Programming Languages*, pages 151–178, 1982.
11. V. Goranko. Coalition games and alternating temporal logics. In J. van Benthem, editor, *Proceedings of TARK VIII*, pages 259–272. Morgan Kaufmann, 2001.
12. V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139(2):241–280, 2004.
13. W. Jamroga. Some remarks on alternating temporal epistemic logic. In B. Dunin-Keplicz and R. Verbrugge, editors, *Proceedings of FAMAS 2003*, pages 133–140, 2003. Updated version. Available at `http://www.in.tu-clausthal.de/$^\sim$wjamroga/papers/atelremarks03FAMAS.pdf`.
14. W. Jamroga and J. Dix. Do agents make model checking explode (computationally)? In *Proceedings of CEEMAS 2005*, Lecture Notes in Computer Science. Springer Verlag, 2005. To appear.
15. W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.
16. G. Jonker. Feasible strategies in Alternating-time Temporal Epistemic Logic. Master thesis, University of Utrecht, 2003.
17. M. Kacprzak and W. Penczek. Unbounded model checking for Alternating-time Temporal Logic. In *Proceedings of AAMAS-04*, 2004.
18. P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2), 2004.
19. W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 1167–1174. ACM Press, New York, 2002.

20. W. van der Hoek and M. Wooldridge. Cooperation, knowledge and time: Alternating-time Temporal Epistemic Logic and its applications. *Studia Logica*, 75(1):125–157, 2003.
21. S. van Otterloo and G. Jonker. On Epistemic Temporal Strategic Logic. In *Proceedings of LCMAS*, pages 35–45, 2004.

Improved Algorithms for Polynomial-Time Decay and Time-Decay with Additive Error

Tsvi Kopelowitz and Ely Porat

Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
{kopelot, porately}@cs.biu.ac.il

Abstract. We consider the problem of maintaining polynomial and exponential decay aggregates of a data stream, where the weight of values seen from the stream diminishes as time elapses. This type of aggregation was first introduced by Cohen and Strauss in [4]. These types of decay functions on streams are used in many applications in which the relative value of streaming data decreases since the time the data was seen. Some recent work and space efficient algorithms were developed for time-decaying aggregations, and in particular polynomial and exponential decaying aggregations. All of the work done so far has maintained multiplicative approximations for the aggregates. In this paper we present the first O(log N) space algorithm for the polynomial decay under a multiplicative approximation, matching a lower bound. In addition, we explore and develop algorithms and lower bounds for approximations allowing an additive error in addition to the multiplicative error. We show that in some cases, allowing an additive error can decrease the amount of space required, while in other cases we cannot do any better than a solution without additive error.

1 Introduction

In many recent applications the need to manage systems with high speed communication and massive data sets arises. In such applications it is more appropriate to use the data stream model in which data arrives rapidly and needs to be processed by an algorithm that lacks the needed space in order to store all of the streaming data. The algorithm stores a synopsis or summary of the data using space that is much less than the amount of data arriving from the stream. Using the synopsis the algorithm can answer queries regarding the data. However, generally there is a tradeoff between the size of the synopsis and the precision of the answers.

Consider the problem of maintaining the sum of all data seen by a stream. This can be answered exactly using $\Theta(\log S)$ bits, where S is the value of the sum. Morris in [12] showed how to approximate the sum using $O(\log\log S)$ bits.

However, in many applications the weight of data diminishes with time, and therefore should weigh less towards the sum of a stream. In some applications we might be interested in the sum over some recent time frame. We can use a *decay function*, dependent on elapsed time, in order to determine the weight of each item. A *decayed sum* is a weighted sum of the data in accordance to the decay function.

1.1 Applications

We list some applications in which the decayed aggregation or averages (that are essentially the same) are used. Many other applications that use the data stream model can conceivably use the decayed sum scheme for their purpose.

– **Detecting Fraud Transactions by Credit Cards.** Credit card companies use recent behavior of costumers in order to detect whether a particular transaction deviates considerably from a pattern of transaction history of a given costumer [6].

– **Patterns of AT&T Telecom Costumers.** As mentioned in [5], AT&T has an application in which they maintain statistics of approximately 100 million costumers. It is vital to balance the weight of information and available storage space in such an application.

– **The Random Early Detection (RED) Protocol.** Many routers on the internet use the RED protocol for avoiding congestions and controlling paths of data. In order to estimate the congestion, RED uses a weighted average of previous queue lengths [10,7].

– **Selecting Internet Gateways.** In the internet there are many paths to each destination. When selecting a path according to reliability a decayed average of previous reliability measures can be used as an estimate.

1.2 Related and Previous Work

As mentioned by Cohen and Strauss in [4], one of the most intuitive and common decay is the exponential decay that can be easily maintained in $\Theta(\log N)$ space (throughout this paper we assume that N is the amount of time elapsed) by the formula $C \leftarrow (1-w)\,x + wC$ where x is the value of the new data and $0 < w < 1$ determines the decay.

Datar et al in [6] considered sliding window decays where for a given W we are interested in the sum of data seen in the last W time units. They showed an algorithm and a matching lower bound of $\Theta\left(\log^2 N\right)$ bits needed for approximating sliding windows. A more detailed explanation of their techniques is in Sect. 3. Gibbons and Tirthapura in [8] extended sliding windows for distributed streams.

Cohen and Strauss in [4] presented an $O\left(\log^2 N\right)$ space algorithm for maintaining general decay functions. This showed that sliding window decay is the hardest decay function, in some sense. They also presented an $O(\log N \log\log N)$ space algorithm for polynomial decay, and showed a lower bound of $\Omega(\log N)$.

In addition, Cohen and Strauss in [4] argue that the exponential decay and sliding window decay are insufficient for many applications in the sense that they both strongly disregard old data. One expects the weight of data to diminish with time, however, for severe events we don't want the value to diminish entirely. We present an example from [4] that illustrates this.

We wish to measure the availability of network links using a time decay function. Consider two links, L1 and L2. Assume that L1 fails for the duration of five time units between time units 5 and 10, while L2 fails for the duration of half a time unit, starting from time unit 34. In addition assume no more failures occur in either of the links. As time progresses we expect L2 to be considered a more reliable link as opposed to L1.

If we use sliding window decay, we might miss out on the failure of L1 (if the window size is too small), and assume L1 is a more reliable link. If we use exponential decay we notice that the ratio between the weight of L1's failure and the weight of L2's failure remains fixed as time progresses. This means that whatever was considered more reliable at time unit 35 (that can be either one of the links) will be considered more reliable forever.

Ultimately, neither the sliding window decay nor the exponential decay can provide us with a view in which we can guarantee that L1 will be considered more reliable. However, polynomial decay has the property that as time progresses, items seen in the same time vicinity are given approximately the same weight. Using Polynomial decay we can obtain a view in which L1 will be considered more reliable as time progresses. We also add that polynomial decay functions enhance our ability to tune the rate of decay. This gives a strong motivation for studying polynomial decay aggregates

1.3 Our Results

In this paper we present the following. We show the first $O(\log N)$ space algorithm for polynomial decay. This matches the bound obtained by Cohen and Strauss in [4]. We explore another model of approximation that allows an additive error in addition to the multiplicative one. We show that in this model, the amount of space necessary in the Exponential decay can be reduced due to the additional slackness we get from the additive error. Finally, we show that generally, in Polynomial decay, the amount of space required cannot be reduced (we prove a lower bound of $\Omega(\log n)$ space), unless the values of the stream are bounded by a constant, and the sum of the decay function converges.

The paper is structured as follows. In Sect. 2 we provide the definitions for the decay functions that we will use throughout the paper. In Sect. 3 we present some known data structures used for approximating decay sums. In Sect. 4 we present a new data structure that we use to achieve an $O(\log N)$ space algorithm for a polynomial decay. In addition, we reprove in Sect. 4 the lower bound from [4] due to a (very) slight error in the original proof, and in order to introduce it for the lower bound in Sect. 5. In Sect. 5 we define the model that allows more slackness of an additive error, and we analyze the exponential and polynomial decays under the model presented including a lower bound for some cases of the polynomial decay.

2 Preliminaries

We begin by formally defining the problems we will be solving in this paper.

2.1 Decay Functions

We represent the definitions given by Cohen and Strauss in [4] for the decay functions we are interested in. Consider a stream of data where $f(t) \geq 0$ is the item value of the stream obtained at time t. For sake of simplicity we assume our stream only receives values at discrete times, and therefore, t is integral. We define a *decay function* $g(x) \geq 0$ defined for $x \geq 0$ to be a non-increasing function. At time T the weight of the item that arrived at time $t \leq T$ is $g(T-t)$ and the *decayed value* is $f(t)g(T-t)$. We are interested in obtaining the *decayed sum* of $f(t)$ under the decay function $g(x)$ that is defined as follows:

Problem 1. Decayed Sum Problem (DSP). *Given a stream $f(t)$ and a decay function $g(T-t)$ estimate at any time T the decayed sum that is given by:*

$$V_g(T) = \sum_{t \leq T} f(t)g(T-t). \tag{1}$$

When $f(t)$ has binary values, we refer to the DSP as the *Decayed Count Problem* (DCP).

We will be interested in two types of decays. The first one is known as the *Exponential Decay* (ExpD) where for a given parameter $\lambda > 0$ we have $g(t) = 1/\lambda^t$. The other type of decay is the *Polynomial Decay* (PolyD) where for a given parameter $\alpha > 0$ we have $g(t) = 1/t^\alpha$. Cohen and Strauss in [4] argue that often PolyD is a better choice than ExpD because in many applications the decay should not keep a constant ratio between two different decay values as time progresses (as was shown in the example in the introduction) .

2.2 Sliding Windows

Sliding windows are also a family of decay functions introduced by Datar et al in [6]. For a window of size W we have the decay function $g(x)$ such that for $x \leq W$, $g(x) = 1$, and zero otherwise. All of the data in the recent time frame of size W have equal weight, while all of the data not in this time frame has no weight at all. Datar et al in [6] show an algorithm that achieves an approximation of $(1+\epsilon)$ in $O(\log^2 W)$ space. We refer to the decayed sum of a sliding window of size W as V_{SLIWIN_W}.

3 Exponential Histograms for Polynomial Decay

We now present the data structures we will use in this paper. We first review Exponential Histograms that were originally introduced by Datar et al in [6], and then continue on with a review on Weight-Based Merging Histograms, that were introduced in [4].

3.1 The Exponential Histogram

The *Exponential Histogram* (EH) was originally introduced in [6] in order to approximate sliding windows. For a given constant k we define the EH for a binary stream $f(t)$ as a collection of buckets that are formed as follows. When the first non-zero value arrives we open a bucket and insert that value into it. Each bucket is immediately sealed. We continue creating buckets in the same manner until we receive the (k+1)'th non-zero value. At this point we merge the first two buckets arrived into one bucket of size two, and create a new bucket for the newly arrived non-zero value. We continue in this manner so that whenever we have $k+1$ buckets of size 1 we merge the earliest two buckets of size 1 to one bucket of size 2. Recursively, whenever we have $k+1$ buckets of size 2^i we merge the two least recent ones to one bucket of size 2^{i+1}. This gives a total of $O(\log N)$ buckets. For each bucket we maintain a *timestamp* that is the time elapsed since the latest value in the bucket arrived. The *timestamp* requires $O(\log N)$ space per bucket. We also maintain the size (the number of ones) of each bucket using one bit per bucket - because for each bucket we just need to know whether it is the same size as the previous one, or double the size. This gives a total of $O(\log^2 N)$ space.

When approximating sliding window decay, we sum up the counts for all of the buckets with *timestamp* inside the window. In case there is a bucket with some values inside the window and some outside, we add only half the count for that bucket. The absolute error in the estimate for the sliding window decay is half the size of the last bucket. As proven by Datar et al in [6], the approximation for the sliding window decay using the EH is $(1 + 1/k)$, so letting $k = 1/\epsilon$ we can get our desired approximation.

3.2 Weight-Based Merging Histogram

The Weight-Based Merging Histogram (WBMH) was originally introduced by Cohen and Strauss in [4]. Like the EH, the WBMH also aggregates values into buckets. However, the boundaries of the buckets are only dependent on the decay function at hand, and not on a particular stream. This means that the timestamps do not need to be stored. They are set by the decay function that is assumed to be known.

The WBMH utilizes the fact that if a decay function has the property that the ratio $g(x)/g(x + \Delta)$ is non-increasing with x for any time frame Δ then the ratio of two items remains fixed, or approaches one as time advances. The meaning of such a property is that as time progresses, items in larger vicinities have the same value up to a multiplicative factor, so we can group those values together. For convenience we let $b_0 = 1$. We let b_1 be the maximum value such that $(1 + \epsilon')g(b_1 - 1) \geq g(1)$. In the same manner, let b_i be the maximum value such that $(1 + \epsilon')g(b_i - 1) \geq g(b_{i-1})$. The values of b_i are dependent on the decay function, and therefore (as we previously mentioned) they do not need to be stored. The i'th bucket starts at time b_{i-1} and ends at time $b_i - 1$.

One family of decay functions with the above property is PolyD in which we have $s = O(\log N)$ different b_i's for each function. For each bucket we hold an approximate counter of size $O(\log\log N)$ (as described in [12], and mentioned in the introduction) that counts the number of bits in that bucket. Cohen and Strauss in [4] proved that the WBMH gives an approximation for DCP of PolyD using $O(\log N \log\log N)$ space.

4 Polynomial Decay

In this section we present a new algorithm for maintaining PolyD aggregation using $O(\log N)$ space. In addition we reprove the matching lower bound of $\Omega(\log N)$ space (this is done due to a very minor error in the original proof, and for the lower bound in Sect. 5 that utilizes the same idea).

4.1 DCP Under PolyD

Cohen and Strauss in [4] used the EH in order to get an approximation for general decay functions using $O(\log^2 N)$ space. In addition, [4] used another type of histograms – the Weight Based Merging Histograms (WBMH) - in order to maintain PolyD in $O(\log N \log\log N)$ space. We combine both of those methodologies, and alter the EH in order to achieve an upper bound of $O(\log N)$ space. This new data structure will be referred to as the Altered Exponential Histogram (AEH).

We start of with the following equation for general decay functions taken from [4]:

$$V_g(T) = g(N)V_{SLIWIN_N} + \sum_{i=1}^{N-1}(g(N-i) - g(N+1-i))V_{SLIWIN_N \ i}. \quad (2)$$

What this means is that maintaining exact sliding windows allows maintaining exact decayed counts for general decays. However, when allowing an approximation we notice that some decayed functions do not need to be able to calculate all of the T different sliding windows. Instead of holding an exact sliding window for each $t < T$, we can hold sliding windows for all of the times b_i (defined in the previous section). This gives us the following approximation for $V_g(T)$:

$$\bar{V_g}(T) = g(N)V_{SLIWIN_N} + \sum_{i=2}^{s}(g(N-b_i) - g(N-b_{i-1}))V_{SLIWIN_N \ b_i}. \quad (3)$$

Using equation (3) and the same arguments as in [4] for WBMH we get a $(1 \pm \epsilon')$ approximation per sliding window - for a total of $(1 \pm \epsilon')$ approximation for functions that have the *non-increasing ratio* property, such as PolyD.

The problem with this solution is that maintaining the sliding windows with EHs requires $O(\log^2 N)$ space. So, we will now alter the EH in order to save space in approximating each of our $O(\log N)$ sliding windows. For each of the

$O(\log N)$ buckets in the EH that are formed as in [6], we round the *timestamp* down to the nearest b_i. When approximating a sliding window that ends at some b_i, we can sum the sizes of all buckets with a *timestamp* smaller than b_i. However, for the last bucket we count only half of the size. Using the same arguments as in [6] this gives us a $(1 \pm \epsilon'')$ approximation per sliding window. Combining our two approximations, we get a $(1 \pm \epsilon')(1 \pm \epsilon'')$ approximation, so all we need is that $\epsilon \geq \epsilon' + \epsilon'' + \epsilon'\epsilon''$, and we can set $\epsilon' = \epsilon'' = \epsilon/3$ in order to get our desired approximation. In order to save space, instead of maintaining the *timestamp* for each bucket, we can maintain the index of its b_i in $O(\log \log N)$ bits per bucket. This would take $O(\log N \log \log N)$ space. We can further decrease the space if for each bucket we maintain the difference in indexes between its *timestamp* and the *timestamp* from the preceding bucket. We can then find the index of the *timestamp* of a given bucket by summing the differences from the first bucket, till the bucket we are interested in. The total space needed for this is $O(\log N)$ (because we have that the sum of the offsets in $O(\log N)$, and each offset o_j needs $O(\log o_j)$ bits, so the sum of bits needed to store all of the offsets is $\sum \log o_j \leq \sum o_j = O(\log N)$). There are some details regarding how we update the *timestamp*s or offsets that we have skipped. We leave this for the journal version. Finally we have:

Theorem 1. *There exists an algorithm that can estimate DCP within a multiplicative factor using $O(\log N)$ space.*

4.2 DSP Under PolyD

When dealing with general streams (we assume general streams are still restricted to integral values) a reasonable assumption is that for some constant β the values given by the stream are bounded by $O(N^{\beta})$. In such a case, we can use the same method for DCP in order to maintain the decayed aggregation in the following manner.

Assume that at time t we receive the value $m = f(t)$. We can treat the value m as m bits of value one, all arriving at the same time from the stream. Therefore, it is as if we inserted m bits to the AEH at the same time. We can use the method from the previous section that will give the same approximation, and the space required is bounded by $O(\log N^{\beta+1}) = O(\log N)$.

However, when using this method it takes $O(m)$ time to insert the value m to our AEH. For many applications this is far too much time. For example, a router on the internet must be able to process data quickly before the next piece of data arrives, so lowering the amount of time it takes to insert a value m is crucial. One possible solution would be to insert the value of m in $O(\log m)$ time by calculating the presentation of m in the appropriate base, and immediately create the appropriate buckets (this will give us $O(\log m)$ buckets, which we can then insert one by one taking $O(\log m)$ time). However we would like to do better than $O(\log m)$ as well, as we do not want the time to be dependant on the possible values arriving in the data stream, and rather, spend a constant amount of time no matter what is the value arrived. The rest of this subsection shows how we can solve this problem in order to achieve $O(1)$ time for insertion of a value.

In addition to our AEH, we add two WBMHs, referred to as W1 and W2, each for a time frame of size $\log N$. In each one of the WBMHs there are $O(\log \log N)$ buckets. This is because the length of the stream is $k = logN$, so we need $\log k = \log \log N$ buckets. Each bucket has a counter of size $O((\log \log N)^2)$. this is because the total sum of a stream of length k is bounded by $kN^\beta = N^\beta \log N$, so each bucket needs a counter of size $\log kN^\beta = O(\log \log N)$. Therefore, we require another $O((\log \log N)^2)$ space, that we can afford within our $O(\log N)$ upper bound. We note that because the WBMH uses approximate counters, we need to take that extra approximation into consideration when choosing the constants for operation. The first $\log N$ items that arrive are inserted into W1. Each of these inserts takes $O(1)$ time because all of the bits fall in the same bucket (recall that we treat the value m as m bits arriving at the same time). This is because the beginning and end of each bucket does not depend on the values arrived, but rather on the time the value arrived, so each item that arrives at some time goes into only one bucket. This means that we only need to update one counter every time a new item arrives. After $\log N$ items arrive, we insert the next $\log N$ items into W2, and concurrently, we insert the values from W1 to the AEH (we will soon show how to do this efficiently). We continue switching between W1 and W2 as time progresses.

Now we need to show how inserting the values from a WBMH to an AEH can be done efficiently. For sake of convenience, we will assume we want to insert the values from W1. The number of bits in W1 is bounded by $N^{\beta+1}$, hence the amount of buckets to be added to the AEH is $O(\log N)$. We also need to merge the appropriate buckets that were already in the AEH and update their times. Using an $O(\log N)$ sized counter we can exactly count the sum of the counters in W1. From this number we can calculate how many buckets will be added to the AEH, and update it accordingly. In $O(\log N)$ time we can update the buckets already inside the AEH, and in $O(\log N)$ time we can insert the new buckets into the AEH. This gives a total of $O(\log N)$ time to insert the values in W1 into the AEH, which is amortized $O(1)$. In order to obtain a worst case time of $O(1)$ we can take a lazy approach, and spread our operations on the $\log N$ insertions made into W2.

The last thing we need to take care of is answering a query while inserting the values from W1 to AEH. Specifically, in the lazy approach we need to wait till we are done with the whole process of merging W1 with the AEH before answering a query. To avoid this problem we keep two copies of the AEH and two copies of each WBMH. When joining W1 with the AEH, we will join a copy of each, while answering queries from the duplicates. Of course, we also need to query W2 due to items already inserted into it. This completes our data structure and provides the following:

Theorem 2. *There exists an algorithm that can estimate DSP within a multiplicative factor using $O(\log N)$ space.*

4.3 Lower Bound for DCP and DSP Under PolyD

Due to a (very) minor error in [4] for proving the lower bound, we present a corrected version of the proof, similar to the original one. We will reuse the idea in this proof again in order to prove another lower bound in the Sect. 5.

Theorem 3. *A logarithmic number of bits is necessary in order to approximately maintain decay by $g(x) = 1/x^{\alpha}$, within a multiplicative factor (in elapsed time).*

Proof. We will first prove for decayed sums (with non-binary values), and then extend the proof to the binary case using communication complexity and a reduction to the subset problem. We show that if there exists an algorithm A that uses less than $\Omega(\log N)$ bits in order to approximately maintain decay by the given function, then we can use algorithm A in order to distinguish between two subsets of a set of size $\Omega(\log N)$ using $o(\log N)$ space, contradicting the lower bound from [13].

Consider the decay function as stated in the theorem. Let k be some positive value to be determined later. In addition, assume we have two parties, A and B, and a set of size $r = \lfloor (\alpha/(2\log k))\log(N/2) \rfloor$. B wants to know which subset A has, and then determine whether it is the same subset that B has. A sends to B some bits of information in order to determine this. A can use algorithm A on the following stream.

We consider a time interval from -$N/2$ till $N/2$. For $i \leq r$, at time $t = -(kN^{-\alpha/2})^{2i/\alpha}$ we receive a value $C_i = \delta_i(kN^{-\alpha/2})^i$, where $\delta = 1$ if i is a member of the set, and $\delta = 2$ otherwise. No data arrives after $t = -1$. Let $k' = kN^{-\alpha/2}$. At time $t = 0$, A sends the bits currently stored by algorithm A to B. B continues running algorithm A assuming nothing arrives on the stream for the rest of the interval. We will now show how by using algorithm A, B can reproduce C_i for every $i \leq r$, and hence can reproduce the subset that A has.

The decayed value $V_g(t)$ at time $t_i = {k'}^{2i/\alpha}$ is

$$V_g(t_i) = \sum_{j=1}^{r} g({k'}^{2i/\alpha} + {k'}^{2j/\alpha}{k'}^{j}\delta_i). \tag{4}$$

We can therefore compute upper bounds for the contribution of the prefix and suffix of this sum (in the original proof, there was a slight algebraic error that carries on throughout - here we correct this slight error, also showing how the rest of the proof can be made to adhere to the correction). For the prefix we have

$$\sum_{j=1}^{i-1} g({k'}^{2i/\alpha} + {k'}^{2j/\alpha}{k'}^{j}\delta_i) \leq 2\sum_{j=1}^{i-1} g({k'}^{2i/\alpha}){k'}^{j} \tag{5}$$

$$= 2\sum_{j=1}^{i-1} {k'}^{j-2i} = 2{k'}^{-i}\sum_{j=1}^{i-1} {k'}^{-j} \leq 2{k'}^{-i}\sum_{j=1}^{\infty} {k'}^{-j} \leq 2{k'}^{-i}/(k'-1).$$

For the suffix we have

$$\sum_{j=i+1}^{r} g({k'}^{2i/\alpha} + {k'}^{2j/\alpha}{k'}^{j}\delta_i) \leq 2\sum_{j=i+1}^{r} g({k'}^{2j/\alpha}){k'}^{j} \tag{6}$$

$$= 2\sum_{j=i+1}^{r} {k'}^{-j} = 2{k'}^{-i}\sum_{j=1}^{r-i} {k'}^{-j} \leq 2{k'}^{-i}\sum_{j=1}^{\infty} {k'}^{-j} \leq 2{k'}^{-i}/(k'-1).$$

In addition, the difference between the two possible values for the i'th term is

$$2g(2{k'}^{2i/\alpha}){k'}^{i} - g(2{k'}^{2i/\alpha}){k'}^{i} = g(2{k'}^{2i/\alpha}){k'}^{i} = 2^{-\alpha}{k'}^{-i}.$$

The contribution of the prefix and the suffix to the total value is at most $2^{2+\alpha}/(k'-1)$ of a fraction of the difference between the two possible values of the i'th term. We can have this fraction be less than 1 if we have $k' > 2^{2+\alpha}+1$, which would then mean that we can distinguish between the two values of δ_i, as the gap between the case where $\delta_i = 1$ and $\delta_i = 2$ is large enough although the approximation. So we can reproduce whether A has the i'th member in its set or not. This mean that B can reproduce the set held by A, although A sent $o(\log N)$ bits, contradicting [13].

We now extend our proof for a binary stream. We will show that if there exists an algorithm A' for the binary stream scenario that uses $o(\log N)$ bits of space, then in the same problem as above, B can reproduce A's subset using $o(\log N)$ space. We first note that for $\alpha < 2$ the proof is basically the same, because we can simply spread a value of $C_i = \Theta({k'}^{i})$ at time $-k'^{2i/\alpha}$ to C_i ones, contradicting the proof for DSP (the beginning of the proof of this theorem). The reason is that all of the ones that come from some value occur around the same time; therefore they essentially have the same weight (in regard to the decay function). If $\alpha \geq 2$, we cannot spread all of the ones, because we don't have enough room between consecutive values in the stream. However, if we would like to solve the DSP problem using A' that solves the DCP problem, we could divide all of the values in our stream some predetermined value, and in the proof from above, the ratios between the suffix, prefix, and i'th value will remain the same. We divide the values of the C_is by $N^{1/\alpha}$, and use only the slots for which the value given is at least one, and at most $2\epsilon(k')^{2i/\alpha}/\alpha$. We note that when spreading the ones to the past, the values of the suffix and prefix only decrease, and the value of the i'th term (if the value is within the right boundaries) is between $g(2k'^{2i/\alpha})$ and $g(2k'^{2i/\alpha} + 2\epsilon(k')^{2i/\alpha}/\alpha)$. This means that the weight of the i'th term can decrease by a factor of $1/(1+\epsilon/\alpha)^{\alpha}$, which for $\alpha \geq 2$ (as is our case) is approximately $(1-\epsilon)$. This minor loss is one that we can easily afford.

Recall that we only use the slots for which the value given is at least one, and at most $2\epsilon(k')^{2i/\alpha}/\alpha$. This gives us two constraints regarding the possible range for i that we can reproduce. However, we will show that the range is still $\Omega(\log n)$. From the constraint that $k'^{i}/N^{1/\alpha} \geq 1$ we get that $i \geq \log_k N/\alpha$, and from the constraint that $k'^{i}/N^{1/\alpha} \leq 2\epsilon(k')^{2i/\alpha}/\alpha$ we get that $i \leq \log_k N/(\alpha - 2) + \alpha\log(2\epsilon/\alpha)/(\alpha-2) = i \leq \log_k N/(\alpha-2) - \Theta(1)$, being that $\alpha \geq 2$. If we let

k be a small enough value such that it is not much larger than $(4/\epsilon + 1)N^{2/\alpha}$, or is even equal to it, then we have that $\log k \geq log(4/\epsilon + 1) + (\alpha \log N)/2$ and therefore $log_k N = \log N/(\log k - log(4/\epsilon + 1)) = \log N/c$ for some constant c. This means that the number of slots that B can reproduce is $\Omega(\log n)$, as required. □

5 Adding Additive Error – Another Model

Until now all of the approximation models used in maintaining values for the DCP and DSP problems looked into $1 \pm \epsilon$ approximations. We present a new view where we allow an additive error as well. In other words, we want to be able to approximately answer the DCP and DSP problems allowing values of $(1 \pm \epsilon_1)RV \pm \epsilon_2$, where RV stands for the real value of the answers to the problems.

The motivation for such an approximation comes from realizing that when dealing with massive data sets, we are often not interested in answers with small values. This is because small decayed sums generally occur when we have been receiving zero or very small valued items for an extended period of time - a scenario that is uncommon in many applications. Specifically for binary streams there are many such applications. Interestingly, allowing this extra relaxation, we can answer the DCP and DSP problems using much less space under some decay functions, while in other decay functions the extra relaxation doesn't help at all.

We now explore the PolyD and ExpD decay functions, when allowing an additive error.

5.1 Exponential Decay for DSP

In order to maintain ExpD for the DSP problem we note that in ExpD, every time a new value arrives in the stream, the new decayed aggregate has the value of the previous aggregate divided by the base, plus the new arrived value. Using this observation, we can maintain the $O(\log \log N)$ most significant digits of our decayed aggregate. When a new number arrives, we divide the previous aggregate by the base, and add the appropriate most significant $O(\log \log N)$ digits from the new value. Regarding the rest of the least significant digits, we take the value of the ignored digits, and divide that value by our additive approximation. We use the fraction we receive as the probability of adding 1 to our $O(\log \log N)$ bits. The probabilistic analysis is deffered to the journal version.

5.2 Exponential Decay for DCP

For a binary stream it is enough to maintain the last $O(\log 1/\epsilon_2)$ bits seen by the stream, and using the AEH from we can do this in $O(\log \log 1/\epsilon_2)$ space. This of course is a great improvement in space, as it is independent of N.

5.3 Polynomial Decay

When considering the polynomial decay, we differentiate between the case in which $\alpha > 1$ and the stream is binary, to the case in which $1 \geq \alpha > 0$ or the stream can receive general polynomial sized values. When $\alpha > 1$ we show how to further reduce the amount of space used, while in the $1 \geq \alpha > 0$ case we have a lower bound showing that we cannot do better in space used then in the previous model where we allowed only a multiplicative error. The intuitive reason for separating the two cases comes from the well know fact that $\sum_{j \leq T} g(t)$ with $1 \geq \alpha > 0$ diverges, while the sum with $\alpha > 1$ converges. As we will show, the lower bound follows this intuition.

Binary Polynomial Decay with $\boldsymbol{\alpha > 1}$. In this case we only need to maintain the last $1/\epsilon_2$ seen by the stream, and again we can use the AEH to do this in $O(\log 1/\epsilon_2)$. We note that it is easily possible to extend this to any stream with values bounded by some constant.

A Lower Bound for Polynomial Decay

Theorem 4. *A logarithmic number of bits is necessary in order to approximately maintain decay by $g(x) = x^{-\alpha}$, within a multiplicative factor and an additive error (in elapsed time), unless $\alpha > 1$ and the stream's values are bounded by a constant.*

Proof. We provide a sketch of the proof, as the proof here follows the proof of Theorem 3. We also note that by proving that for $\alpha > 1$ and a binary stream one can do better than logarithmic space, we can easily extend this to any stream with values bounded by some constant.

We start off with general stream, and show how we can extend the proof from Theorem 3 to an additional additive error. Let m be some positive valued number to be determined later. We multiply all of the values of the stream that A creates by $m^{\alpha+1}$, and spread the times so that the values arrive at times $-mk'^{2i/\alpha}$ for $i \leq r$. This increases the decay value by a factor of m, while the ratio between the sum of the prefix and suffix, and the i'th term remains unchanged. We can always choose an m large enough to *swallow* the additive error, and because the ratio is unchanged, the multiplicative factor does not affect B's ability to reproduce the set.

For binary streams where either $\alpha \leq 1$, or the values of the stream or not bounded by a constant, we use the same extension as in the proof of Theorem 3. The reason this proof does not work when $\alpha > 1$ for the binary (or constant) case is that we have stream values of size $\delta_i m^\alpha k'^{2i/\alpha}$ that we need to spread between times $-mk'^{2i/\alpha}$ and time $-mk'^{(2i+2)/\alpha}$, and there simply isn't enough room for all of the bits between consecutive times. Therefore, the proof doesn't apply in such a case. □

6 Conclusions

We discussed polynomial and exponential decay in data streams, extending recent work by [6] and [4]. We present the tightest space-efficient algorithm for

PolyD using $O(\log N)$ space when allowing approximations up to a multiplicative factor. We do this by presenting a new data structure, the AEH, that is an extension of the EH from [6] and the WBMH from [4].

In addition, we presented and analyzed our decay functions when allowing an approximation with an additive error in addition to the multiplicative one. We find that in ExpD for general streams $O(\log \log N)$ space suffices, while in binary streams a constant (depending on the approximation) amount of space is all we need. For the PolyD we prove that unless the exponent is larger than one, and the stream values are bounded by some constant, we cannot hope to do better than in the multiplicative only approximation. We also show that one can use the AEH in order to achieve a constant-space algorithm for PolyD when the exponent is larger than one, and the stream values are bounded by some constant.

The need to analyze other decay functions such as the chordal decay, poly-exponential decay, etc. ([4]) still exists. We strongly feel that the AEH can be used in some of the other decay functions, and in many other applications as well.

Acknowledgments

We would like to thank an anonymous referee for some useful comments.

References

1. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. Of the 2002 ACM Symposium on Principles of Database Systems (PODS 2002)*, ACM, 2002.
2. A. Bremler-Barr, E. Cohen, H. Kaplan, and Y. Mansour. Predicting and bypassing internet end-to-end service degradations. In *Proc.* 2^{nd} *ACM-SIGCOMM Internet Measurement Workshop.*, ACM, 2002.
3. E.cohen H.Kaplan, and J.D. Oldham. Managing TCP connections under persistent HTTP. In Computer Networks, 31:1709-1723, 1999.
4. E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. Of the 2003 ACM Symposium on Principles of Database Systems (PODS 2003)*, ACM, 2003.
5. E. Cohen and M. Strauss. Giga-mining. In *Proc. Of KDD*, New York, August 1998.
6. M. Datar, A.Gionis, P.Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc.* 13^{th} *ACM-SIAM Symp. On Discrete Algorithms.* , ACM-SIAM, 2002.
7. S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. In *IEEE/ACM Transactions on Networking*, 1(4), 1993.
8. P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. Of the* 14^{th} *Annual ACM Symp. On Parallel Algorithms and Architectures*, Pages 63-72. ACM, 2002.
9. Warren Gilchrist. Statistical Forecasting. Wiley, 1976.

10. V. Jacobson. Congestion avoidance and control. In *Proc. Of the ACM-SIGCOMM'88 conference*, August 1988.
11. S. Keshav, C.Lund, S. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in IP-over-ATM networks. In *IEEE J. on Selected Areas in Communication*, 13, 1995.
12. R. Morris. Counting large numbers of events in small registers. In *CACM*, 81:840-842, 1978.
13. A. C. Yao. Some complexity questions related to distributed computing. In *Proc. of the 11th ACM STOC*, 209:213, 1979.

A Theoretical Analysis of Alignment and Edit Problems for Trees*

Tetsuji Kuboyama[1], Kilho Shin[2], Tetsuhiro Miyahara[3], and Hiroshi Yasuda[2]

[1] Center for Collaborative Research, University of Tokyo, Japan
4-6-1 Komaba Meguro, Tokyo 153-8505, Japan
kuboyama@ccr.u-tokyo.ac.jp

[2] Research Center for Advanced Science and Technology, University of Tokyo, Japan
4-6-1 Komaba Meguro, Tokyo 153-8904, Japan
{kilho_shin, yasuda}@mpeg2.rcast.u-tokyo.ac.jp

[3] Faculty of Information Sciences, Hiroshima City University, Japan
3-4-1 Ozuka-Higashi, Asaminami, Hiroshima 731-3194, Japan
miyahra@its.hiroshima-cu.ac.jp

Abstract. The problem of comparing two tree structures emerges across a wide range of applications in computational biology, pattern recognition, and many others. A number of tree edit methods have been proposed to find a structural similarity between trees. The alignment of trees is one of these methods, introduced as a natural extension of the alignment of strings, which gives a common supertree pattern of two trees, whereas tree edit gives a common subtree pattern. It is well known that alignment and edit are two equivalent notions for strings from the computational point of view. This equivalence, however, does not hold for trees. The lack of a theoretical formulation of these notions has lead to confusion. In this paper, we give a theoretical analysis of alignment and edit methods, and show an important relationship, which is the equivalence between the the alignment of trees and a variant of tree edit, called less-constrained edit.

1 Introduction

A tree structure plays a significant role in the efficient organization of information. In particular, the problem of comparing tree structures emerges across a wide range of applications in computational biology [1], image analysis [2], pattern recognition [3], natural language processing, information extraction [4] from Web pages, and many others.

Edit-based approaches provide a general framework in comparing trees, measuring similarities, finding common tree patterns, and merging trees. *Tree edit distance* [5,6] and *alignment of trees* [7] were both introduced as natural generalizations of string edit distance [8]. It is well known that alignment and edit are two equivalent notions in strings [9], whereas both are completely different in trees [7].

* This work is partly supported by Grant-in-Aid for Scientific Research No. 17700138, and No. 16016275 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

Although a dozen of tree edit methods have been proposed [10] in various fields, no comprehensive mathematical analysis of these methods has been available. Consequently, the relationship among various tree edit methods has hardly been studied, and essentially equivalent algorithms of tree edit methods have been independently proposed. In fact, an important equivalence between two methods, the alignment of trees and the less-constrained edit has remained unnoticed. We believe that providing a framework in abstract mathematical terms is vital to understand various aspects of edit-based approaches, and to facilitate their efficient implementations.

Towards this goal, in this paper, we propose a new formulation of the notions of alignment and edit for trees, and investigate the relationship among these two notions based on our mathematical formulation. Specifically, our contributions are as follows. **(1)** We show that the alignment of trees is essentially equivalent to a variant of tree edit, called the *less-constrained edit* [11]. **(2)** We give the *mapping* condition for the alignment of trees. The mappings provide definitions of various tree edit methods in a declarative way, in contrast to an operational way of conventional definitions. To the best of our knowledge, the mapping condition for the alignment of trees has been unknown, and the alignment of trees was defined only in an operational way in [7]. **(3)** We show that the condition of the less-constrained mapping given by Lu *et al.* [11] does not relax the condition of the *constrained* mapping due to Zhang [12]. In fact, we show that the condition due to Lu *et al.* [11] is identical to that of the constrained mapping. We revise it and give an originally intended condition of less-constrained mapping.

In Sect. 2, we review the previous work on tree edit methods. In Sect. 3, we give a new formulation of alignment and edit for trees based on the notion of tree mapping. In Sect. 4, we show the main result of this paper that the alignment of trees is equivalent to the less-constrained edit in the mapping condition. In Sect. 5, we conclude this paper.

2 Edit-Based Approach in Trees

In this section, we briefly review some tree edit problems.

Trees we consider in this paper are labeled rooted trees, in which each node is labeled from a finite alphabet Σ. An *ordered tree* is a tree in which the left-to-right order among siblings is given. An *unordered tree* is a tree with no order among siblings. We refer to unordered trees as trees unless otherwise stated.

We denote by $r(T)$ the root of a tree T, and by $T(x)$ the *maximum subtree* of T rooted at a node x. An *ancestor* of a node is recursively defined as follows: an ancestor of a node is either the node itself, or an ancestor of the parent of the node. We denote by $x \leq y$ that a node y is an ancestor of a node x, by $\mathrm{lca}(X)$ the *least*(or *nearest*) *common ancestor* of all nodes in a set of nodes X, and by $x \smile y$ the least common ancestor of x and y.

In an ordered tree, we say that a node x is *to the left* of a node y if $x \smile y$ is a proper ancestor of both x and y, and the child of $x \smile y$ on the path to x is to the left of the child of $x \smile y$ on the path to y.

2.1 Tree Edit Problem

A tree edit problem is the problem of calculating distance between two trees, and finding a common subtree pattern of two trees. A tree edit distance is defined as the minimum cost of elementary edit operations to transform one tree into the other [5,6].

Let l be a labeling function which assigns a label from a set $\Sigma = \{a, b, c, \ldots\}$ to each node. Let λ denote the unique null symbol not in Σ. Let d be a cost function of edit operations.

The *edit operations* on a tree T are the followings:

relabeling of the label of a node x in T with the label of a new node y in T; the cost is denoted by $d(l(x) \to l(y))$,

insertion of a new node x into T as a child of a node y in T, moving a subset (a consecutive subsequence in the case of ordered trees) of y's children and their descendants right under the new node x; note that this is the complementary operation of deletion; the cost is denoted by $d(\lambda \to l(x))$, and

deletion of a non-root node x from T, moving all children of x right under the parent of x; the cost is denoted by $d(l(x) \to \lambda)$.

We assume, without loss of generality, that the root of a tree is not to be deleted or inserted. We refer to each cost factor as its edit operation when there is no confusion; *i.e.*, $\alpha \to \beta, \lambda \to \beta$, and $\alpha \to \lambda$ for $\alpha, \beta \in \Sigma$ are referred to as the edit operations of relabeling, insertion, and deletion, respectively.

The cost function d is defined to be a metric; *i.e.*, for any $\alpha, \beta, \gamma \in \Sigma \cup \{\lambda\}$, (1) $d(\alpha \to \beta) \geq 0$, $d(\alpha \to \alpha) = 0$; (2) $d(\alpha \to \beta) = d(\beta \to \alpha)$; and (3) $d(\alpha \to \gamma) \leq d(\alpha \to \beta) + d(\beta \to \gamma)$.

If a sequence of edit operations E transforms a tree T into a tree U, there exists a sequence of trees $\langle T_0, \ldots, T_n \rangle$ $(n \geq 1)$ such that $T_0 = T$, $T_n = U$, and the i-th edit operation $e_i = (\alpha_i \to \beta_i)$ transforms T_{i-1} into T_i for $i \in \{1, \ldots, n\}$. The cost function d for an edit operation is generalized to that for a sequences of edit operations $E = \langle e_1, \ldots, e_n \rangle$ by letting $d(E) = \Sigma_{i=1}^{n} d(e_i)$.

Let $\mathcal{E}$ be the set of all possible sequences of edit operations to transform T into U. The edit distance δ between two trees T and U is defined as $\delta(T, U) = \min_{E \in \mathcal{E}} \{d(E)\}$.

2.2 Tree Mapping

The effect of a sequence of edit operations is reduced to a structure called *tree mapping*, which is introduced by Tai [5] and simply referred to as *mapping*. We also refer to the *tree mapping* as *mapping* if there is no confusion. A tree mapping depicts node-to-node correspondences between two trees according to the structural similarity, or shows how nodes in one tree are preserved after transformed to the other (See Fig. 2(a)).

Definition 1 (Tai 1979 [5]). A *tree mapping* from a tree T to a tree U is a set $M \subseteq V(T) \times V(U)$ such that, for all (x_1, x_2), $(y_1, y_2) \in M$,

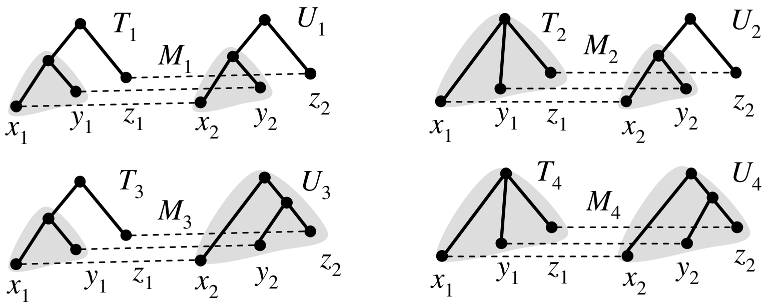


Fig. 1. Examples of tree mappings: each shaded region illustrates how the subtree rooted at $x_1 \smile y_1$ is mapped to the other by each mapping M_i ($i \in \{1, 2, 3, 4\}$). Only M_1 is constrained, and the others are not. M_1, M_2, and M_4 are less-constrained, and M_3 is not.

1. $x_1 \le y_1 \Leftrightarrow x_2 \le y_2$, and
2. (only for ordered trees) x_1 is to the left of $y_1 \Leftrightarrow x_2$ is to the left of y_2.

The original definition of the tree mapping by Tai [5] includes the condition $x_1 = y_1 \Leftrightarrow x_2 = y_2$ for all (x_1, x_2), $(y_1, y_2) \in M$. We omit this condition since it is implied by the condition (1).

For a tree mapping M from T to U, let $V_D = V(T) \setminus \{x|(x, y) \in M\}$, and $V_I = V(U) \setminus \{y|(x, y) \in M\}$. The cost of M is defined as

$$d(M) = \sum_{(x,y)\in M} d(l(x) \to l(y)) + \sum_{x\in V_D} d(l(x) \to \lambda) + \sum_{y\in V_I} d(\lambda \to l(y)).$$

Let $\mathcal{M}$ be the set of all possible tree mappings from T to U. Tai [5] showed that the edit distance δ between T and U is given in the following two ways: $\delta(T, U) = \min_{E\in\mathcal{E}}\{d(E)\} = \min_{M\in\mathcal{M}}\{d(M)\}$. This equation plays the role of a bridge between an operational definition and a declarative definition for the edit distance. For example, Fig. 1 shows examples of tree mappings, in which all nodes connected with dashed lines preserve the tree mapping condition.

2.3 Constrained Mapping

The *constrained mapping* [13] originated from the *structure-preserving mapping* due to Tanaka and Tanaka [14]. Zhang gave a succinct condition of the tree mapping instead of the condition by Tanaka and Tanaka, and presented an efficient algorithm for unordered trees [12].

Definition 2 (Zhang 1996 [12]). A tree mapping M is *constrained* if the following condition holds: for all $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in M$, $z_1 < x_1 \smile y_1$ if and only if $z_2 < x_2 \smile y_2$.

For a tree mapping M from T to U, let M_1 and M_2 be two arbitrary subsets of M. Let $X_i = \{x|(x, y) \in M_i\}$, and $Y_i = \{y|(x, y) \in M_i\}$, for $i \in \{1, 2\}$. An implication of the constrained mapping is that if $T(\mathrm{lca}(X_1))$ and $T(\mathrm{lca}(Y_1))$ are disjoint, then $U(\mathrm{lca}(X_2))$ and $U(\mathrm{lca}(Y_2))$ must be disjoint as well, and vice versa.

As shown in Fig. 1, two disjoint subtrees $T_1(x_1 \smile y_1)$ and $T_1(z_1)$ (note that $\mathrm{lca}(\{z_1\}) = z_1$) are mapped to two disjoint subtrees $U_1(x_2 \smile y_2)$, and $U_1(z_2)$. And the other arbitrary disjoint trees are mapped to disjoint trees by M_1. Thus, M_1 is constrained. On the other hand, two disjoint subtrees $U_2(x_2 \smile y_2)$ and $U_2(z_2)$ are mapped to two non-disjoint subtrees $T_2(x_1 \smile y_1)$ and $T_2(z_1)$. In fact, $T_2(x_1 \smile y_1)$ includes $T_2(z_1)$. Thus, M_2 is not constrained.

2.4 Structure-Respecting Mapping

Richter independently introduced the *structure-respecting mapping* [15] with the same intention of the constrained mapping.

Definition 3 (Richter 1997 [15]). A tree mapping M is *structure-respecting* if the following condition holds: for all $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in M$ such that none of x_1, y_1, and z_1 is an ancestor of the others, $x_1 \smile y_1 = x_1 \smile z_1$ if and only if $x_2 \smile y_2 = x_2 \smile z_2$.

In [11], Lu *et al.* pointed out, without proof, that both the concepts of constrained mapping and structure-respecting mapping are equivalent. We give the proof in Sect. 3.2.

2.5 Less-Constrained Mapping

The less-constrained mapping was introduced by Lu *et al.* [11], which is intended to relax the condition of the constrained mapping [13,12] so that in Fig. 1, both M_1, M_2 and M_4 are less-constrained, whereas M_3 is not less-constrained.

Definition 4 (Lu *et al.* 2001[11]). A tree mapping M is *less-constrained* if the following condition holds: for all $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in M$ such that none of x_1, y_1, and z_1 is an ancestor of the others, $x_1 \smile y_1 \leq x_1 \smile z_1$ and $x_1 \smile z_1 = y_1 \smile z_1$ if and only if $x_2 \smile y_2 \leq x_2 \smile z_2$ and $x_2 \smile z_2 = y_2 \smile z_2$.

This definition, however, does not formulate the concept of the less-constrained mapping correctly. In fact, for example, the definition excludes the case $x_1 \smile y_1 = x_1 \smile z_1 = y_1 \smile z_1$ and $x_2 \smile y_2 > y_2 \smile z_2$, $x_2 \smile y_2 = x_2 \smile z_2$ (See M_4 in Fig. 1). Note that the condition in Definition 4 should hold for any combinations of $(x_1, x_2), (y_1, y_2), (z_1, z_2)$. We give a correct definition in Sect. 3.2.

2.6 Alignment of Trees

The alignment of trees was introduced by Jiang *et al.* [7] as a natural extension of the alignment of strings. In contrast to the tree edit problem, the alignment of trees is viewed as the problem of finding a common supertree pattern of two trees. The definition of the alignment has been given in an operational way [7,16] as follows.

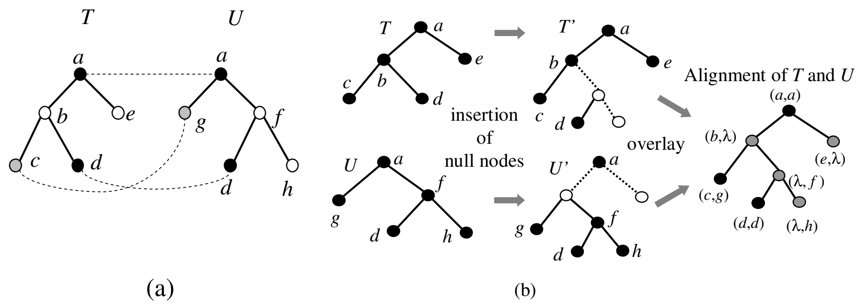


Fig. 2. Example: (a) a tree mapping from T to U: relabeling the node labeled c with g, deleting the two nodes labeled with b and e, and inserting the two node labeled with f and h; (b) An alignment of trees between T and U; the tree mapping in (a) corresponds to this alignment.

Definition 5 (Jiang *et al.* 1995[7]). Let T and U be two trees. An alignment of T and U is obtained by first inserting nodes labeled with the null symbol λ into T and U so that the two resulting trees T' and U' have the same structure (*i.e.*, they are identical if the labels are ignored), and then *overlaying* T' on U'.

Figure 2(b) illustrates an alignment of trees. As shown in Fig. 2(a), we can consider the tree mapping corresponding to the alignment of trees in Fig. 2(b) as well as the tree edit. As for the alignment of trees, however, the tree mapping condition has been unknown in prior work.

From the point of view of tree mapping, it is easy to show that the alignment of tree is different from the tree edit defined by Tai [5]. For example, in Fig. 1, if each pair of nodes in the tree mapping M_3 should be overlaid, then it is impossible to obtain the same tree structure by inserting null nodes into each tree. In fact, the tree mapping M_3 inevitably leads to a directed acyclic graph, not a tree if overlaid.

3 A New Formulation of Tree Edit Problems

We give a new formulation of the tree edit problem to analyze the relationship among edit-based approaches for trees.

3.1 Rooted Trees

We adopt a standard notation $<$ to denote a *strict partial order*, that is, for a non-empty finite set V, (1) $\forall x, y, z \in V \; [x < y \; \wedge \; y < z \Rightarrow x < z]$, and (2) $\forall x \in V \; [x \not< x]$. We denote by $x \leq y$ that $x < y$ or $x = y$ for all $x, y \in V$. We say that two elements $x, y \in V$ are *comparable* if $x < y$, $x = y$ or $y < x$ holds.

Definition 6. A *rooted tree* $T = (V, <)$ is a nonempty, finite, and strict partially ordered set with the maximum element $r(T) \in V$ called the *root*, and such that $\{y \in V | x \leq y\}$ is a totally ordered set for every $x \in V$.

Unless otherwise stated, all trees we consider in this paper are labeled, rooted and unordered trees. Although all the definitions, propositions, lemmas and theorems stated in this paper also hold for the ordered tree with no or slight modification, this paper does not state all of them.

We call the elements of V the *nodes* of T, and denote the set of all nodes in T by $V(T)$. An *ancestor* of x is a node y such that $x \leq y$. In particular, if $x < y$, then y is called a *proper ancestor*. The *parent* of a node x is the minimum node of the proper ancestors of x in T, and denoted by $p(x)$. For a node $x \in V(T)$, we denote by $\mathrm{ch}(x)$ the set of nodes $\{y \in V(T) | y < x$ and $\nexists z \in V(T)$ such that $y < z < x\}$, and refer to the elements of $\mathrm{ch}(x)$ as the *children* of x. A *leaf* of a tree T is a minimal node in $V(T)$.

We redefine the notion of least common ancestor as follows.

Definition 7. For any tree $T = (V, <)$, a *common ancestor* of a set of nodes $V' \subseteq V$ is an element $x \in V$ such that $y \leq x$ for all $y \in V'$. A common ancestor x of V' is the *least common ancestor* of V' if, for any common ancestor x' of V', $x \leq x'$ holds. We denote the least common ancestor of V' by $\mathrm{lca}(V')$, and $\mathrm{lca}(\{x, y\})$ by $x \smile y$.

Lemma 1. *The following properties hold in terms of the least common ancestor:*
1. $x \smile x = x$, *2.* $x \smile y = y \smile x$,
3. $(x \smile y) \smile z = x \smile (y \smile z)$, *4.* $x \leq y \Leftrightarrow x \smile y = y$,
5. $x \smile y < x \smile z \Rightarrow y \smile z = x \smile z$, *and* *6.* $x \smile y = x \smile z \Rightarrow y \smile z \leq x \smile y$.

Proof. (1) to (4) are all easy to prove, and we omit these proofs.
(5): Since $y < x \smile z$ by the premise, we have $y \smile z \leq x \smile z$. On the other hand, if $x \smile y < y \smile z$, then we have $x < y \smile z$, therefore, $y \smile z \geq x \smile z$. If $y \smile z \leq x \smile y$, then $z \leq x \smile y$, therefore, $x \smile z \leq x \smile y$, as is contradictory to the premise.
(6): The assertion immediately follows from $x \leq x \smile z$ and $y \leq y \smile z$. □

3.2 Less-Constrained Mapping Revised

Definition 4 due to Lu *et al.* [11] does not relax that of the constrained mapping. In fact, it is easy to show that the definition ends up with that of the constrained mapping as follows.

Definition 4 is reduced to a more succinct form of condition by Lemma 1 (6); *i.e.*, $x_i \smile y_i \leq x_i \smile z_i$ is implied by $x_i \smile z_i = y_i \smile z_i$ for $i \in \{1, 2\}$ in Definition 4. Therefore, it is shown that the condition due to Lu *et al.* is equivalent to that of the structure-respecting mapping in Definition 3. Hence, by the following proposition, the condition due to Lu *et al.* ends up with that of the constrained mapping.

Proposition 1. *For any tree mapping M, M is structure-respecting if and only if M is constrained.*

Proof. **(only-if part)** Assume that $z_1 < x_1 \smile y_1$. If z_2 and $x_2 \smile y_2$ are comparable, then $z_2 < x_2 \smile y_2$ holds by Definition 1. (i) If any two of x_1, y_1, z_1 are comparable, *i.e.*, z_1 is comparable to x_1 or y_1 (because if $x_1 \leq y_1$, then

$z_1 < x_1 \smallsmile y_1 = y_1$), z_2 and $x_2 \smallsmile y_2$ are also comparable by Definition 1. (ii) Suppose that any of x_1, y_1, z_1 is not an ancestor of any of the others. Since we may assume that $x_1 \smallsmile y_1 = x_1 \smallsmile z_1$ without loss of generality, $x_2 \smallsmile z_2 = x_2 \smallsmile y_2$ holds by Definition 3. Therefore, z_2 and $x_2 \smallsmile y_2$ are comparable, too.

(if part) We show the contraposition of $x_2 \smallsmile y_2 = x_2 \smallsmile z_2 \Rightarrow x_1 \smallsmile y_1 = x_1 \smallsmile z_1$. If $x_1 \smallsmile y_1 \neq x_1 \smallsmile z_1$, we may assume $x_1 \smallsmile y_1 < x_1 \smallsmile z_1$ since $x_1 \smallsmile y_1$ and $x_1 \smallsmile z_1$ are comparable. If $z_2 = x_1 \smallsmile y_1$, then $x_2 \leq z_2$. Therefore $x_1 \leq z_1$ holds. Moreover, if $z_2 < x_2 \smallsmile y_2$, then $z_1 < x_1 \smallsmile y_1$ by Definition 2. Hence, we have $z_2 \not\leq x_2 \smallsmile y_2$. Since $x_2 \smallsmile y_2 < x_2 \smallsmile z_2$ follows, we have the contraposition. By symmetry, $x_1 \smallsmile y_1 = x_1 \smallsmile z_1 \Rightarrow x_2 \smallsmile y_2 = x_2 \smallsmile z_2$ also holds. □

We give a correct definition of the less-constrained mapping as follows.

Definition 8. A tree mapping M is *less-constrained* if the following condition holds: $\forall (x_1, x_2), (y_1, y_2), (z_1, z_2) \in M$ $[x_1 \smallsmile y_1 < x_1 \smallsmile z_1 \Rightarrow y_2 \smallsmile z_2 = x_2 \smallsmile z_2]$.

Proposition 2. *The condition in Definition 8 is equivalent to the condition:* $\forall (x_1, x_2)$, (y_1, y_2), $(z_1, z_2) \in M$ $[x_2 \smallsmile y_2 < x_2 \smallsmile z_2 \Rightarrow y_1 \smallsmile z_1 = x_1 \smallsmile z_1]$.

3.3 Tree Homomorphism

The rest of this section is devoted to define the alignment of trees in a formal manner. We first introduce the notion of tree homomorphism to represent structural similarities between trees. Most of proofs are omitted because of the space limitation.

Definition 9 (Homomorphism). Let T and U be two trees. A *homomorphism* from T to U is a mapping $\varphi : V(T) \to V(U)$ such that $\varphi(x) \leq \varphi(y)$ if $x < y$ for all $x, y \in V(T)$. When a mapping $\varphi : V(T) \to V(U)$ yields a homomorphism of trees, we simply denote it by $\varphi : T \to U$.

Definition 10. For a homomorphism $\varphi : T \to U$, the *image* of φ is a tree $\Im(\varphi) = (V(\Im(\varphi)), <_{\Im(\varphi)})$ such that $V(\Im(\varphi)) = \{x \in V(U) | x \leq \varphi(r(T))\}$, and for all $x, y \in V(\Im(\varphi))$, $x <_{\Im(\varphi)} y \Leftrightarrow x < y$.

Definition 11 (Isomorphism). Let T and U be two trees. An *isomorphism* from T to U is a bijection φ from $V(T)$ to $V(U)$ such that (x, y) is an edge of T if and only if $(\varphi(x), \varphi(y))$ is an edge of U.

It is obvious from these definitions that a composition of homomorphisms is a homomorphism, and an isomorphism φ and its inverse φ^{-1} are both homomorphisms.

Proposition 3. *Let T and U be two trees. Suppose that a homomorphism φ is a bijection from $V(T)$ to $V(U)$. Then the following two properties are equivalent: (1) φ is an isomorphism, and (2) $\forall x, y \in V(T)$ $[\varphi(x) < \varphi(y) \Rightarrow x < y]$.*

3.4 Embedding

We introduce an important subclass of the tree homomorphism, called embedding, which is a mapping from a tree T to a tree U such that it preserves the tree mapping condition, and $V(T) \subseteq V(U)$.

Definition 12 (Embedding). Let T and U be two trees. A homomorphism $\varphi : T \to U$ is an *embedding* if the following conditions are satisfied: φ is injective, and $\forall x, y \in V(T)$ $[\varphi(x) < \varphi(y) \Rightarrow x < y]$.

We refer to $\mathrm{red}(\varphi) = |V(\Im(\varphi)) \setminus \varphi(V(T))|$ as the *redundancy* of $\varphi : T \to U$. Figure 3(a) shows an example of an embedding.

Proposition 4. *For an embedding $\varphi : T \to U$ and $x, y \in V(T)$, the minimum node $\varphi(z)$ in U such that $\varphi(x) \smile \varphi(y) < \varphi(z)$ is identical to $\varphi(x \smile y)$. Furthermore, the following are equivalent: (1) $\varphi(x) \smile \varphi(y) < \varphi(x \smile y)$, and (2) $\varphi(x) \smile \varphi(y) \notin \varphi(V(T))$.*

Proof. Suppose that $\varphi(x) \smile \varphi(y) \leq \varphi(z)$. By Definition 12, we have $x \smile y \leq z$. Hence $\varphi(x \smile y) \leq \varphi(z)$. This implies that $\varphi(x \smile y)$ is the minimum $\varphi(z)$ such that $\varphi(x) \smile \varphi(y) \leq \varphi(z)$. The equivalence between (1) and (2) follows immediately from this property. □

An embedding is uniquely determined except for the isomorphism as shown in the following.

Corollary 1. *Let S, T, and U be three trees. Let $\varphi : S \to U$ and $\psi : T \to U$ be two embeddings with $\varphi(V(S)) = \psi(V(T))$. There exists a unique isomorphism $\eta : T \to S$ such that $\psi = \varphi \circ \eta$.*

Corollary 2. *For an embedding $\varphi : T \to U$, if $x \smile y < x \smile z$, then $\varphi(x) \smile \varphi(y) < \varphi(x) \smile \varphi(z)$.*

Proof. $\varphi(x) \smile \varphi(y) \leq \varphi(x \smile y) < \varphi(x \smile z)$ holds. $\varphi(x \smile y)$ and $\varphi(x) \smile \varphi(z)$ are comparable since both are ancestors of $\varphi(x)$. If $\varphi(x) \smile \varphi(z) = \varphi(x \smile z)$, then there is nothing to prove. If $\varphi(x) \smile \varphi(z) \leq w < \varphi(x \smile z)$, then $w \notin \varphi(V(T))$ by Proposition 4. Therefore, we have $\varphi(x \smile y) < \varphi(x) \smile \varphi(z)$. □

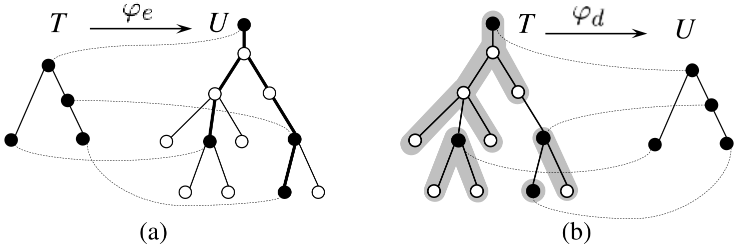


Fig. 3. Example: (a) an embedding φ_e, and (b) a degeneration φ_d

We introduce a useful expression for filtering nodes of trees. For a tree T, let $\pi(\mathbf{x}) : V(T) \to \{\mathbf{true}, \mathbf{false}\}$ denote a unary predicate with a predicate variable $\mathbf{x}$. By $T[\pi(\mathbf{x})] = (V[\pi(\mathbf{x})], <_\pi)$, we denote that $V[\pi(\mathbf{x})] = \{x | x \in V(T)$ and $\pi(x) = \mathbf{true}\}$, and $\forall x, y \in V[\pi(\mathbf{x})]\ [x <_\pi y \Leftrightarrow x < y]$. For example, $T[\mathbf{x} \leq x]$ is equivalent to $T(x)$. Note that $T[\pi(\mathbf{x})]$ is not necessarily a tree since it may not have a root.

By $\mathbf{E}_{\pi(\mathbf{x})}$, we denote a natural inclusion $\mathbf{E}_{\pi(\mathbf{x})} : V(T[\pi(\mathbf{x})]) \to V(T)$.

Proposition 5. *For $x, y \in V(T[\pi(\mathbf{x})])$, $x < y$ if and only if $\mathbf{E}_{\pi(\mathbf{x})}(x) < \mathbf{E}_{\pi(\mathbf{x})}(y)$.*

Note that if $T[\pi(\mathbf{x})]$ is a tree, then $\mathbf{E}_{\pi(\mathbf{x})}$ is an embedding with $\mathrm{red}(\mathbf{E}_{\pi(\mathbf{x})}) = |\{x \in V(T) | \pi(x) = \mathbf{false}\}|$.

Now we are ready to give a formal definition of the insertion operation.

Definition 13 (Insertion). Let T and U be two trees. An embedding $\varphi : T \to U$ with $\mathrm{red}(\varphi) = 1$ is called an *insertion.* In particular, if $\varphi(V(T)) = V(U) \setminus \{x\}$ for $x \neq r(U)$, the insertion φ is called an x-insertion.

Proposition 6. *For any $x \in V(T)$ such that $x \neq r(T)$, there exists an x-insertion φ into T. Furthermore, an x-insertion is unique up to an isomorphism.*

Proof. Let $\pi(\mathbf{x})$ be $\mathbf{x} \neq x$. Then, $\mathbf{E}_{\pi(\mathbf{x})} : T[\pi(\mathbf{x})] \to T$ is an x-insertion into T by Proposition 5. By Corollary 1, an x-insertion is uniquely determined up to an isomorphism. □

We denote the unique x-insertion by I_x.

Proposition 7. *Let T and U be two trees. For $x \in V(U)$, $I_x : T \to U$ satisfies the following properties:*

1. *for any $y \in ch(x)$, $I_x : T(I_x^{-1}(y)) \to U(y)$ is an isomorphism, and*
2. *$I_x : T[\bigwedge_{y \in ch(x)} \mathbf{x} \not\leq I_x^{-1}(y)] \to U[\mathbf{x} \not\leq x]$ is an isomorphism.*

Proof. Without loss of generality, we may assume that $T = U[\mathbf{x} \neq x]$. It follows from Proposition 5 that $U[\mathbf{x} \neq x \wedge \mathbf{x} \leq y] = U(y)$, and $U[\mathbf{x} \neq x \wedge \bigwedge_{y \in \mathrm{ch}(x)} \mathbf{x} \not\leq I_x^{-1}(y)] = U[\mathbf{x} \not\leq x]$. Hence, we obtain the assertions. □

The following theorem shows that the insertion in Definition 13 is equivalent to the operational definition of the insertion.

Theorem 1 (Decomposition of embedding). *Let φ be an embedding from T to U with $V(\Im(\varphi)) \setminus \varphi(V(T)) = \{x_1, \ldots, x_n\}$. There exist a sequence of trees $T_0, T_1, \ldots, T_n$, and a sequence of insertions $\varphi_i : T_i \to T_{i-1}$ ($i \in \{1, \ldots, n\}$) such that $T_0 = U$, $T_n = T$, $\varphi_1 \circ \cdots \circ \varphi_i(V(T_i)) = V(\Im(\varphi)) \setminus \{x_1, \ldots, x_i\}$, and $\varphi = \varphi_1 \circ \cdots \circ \varphi_n$;*

$$\begin{tikzcd}
T_n \arrow[r, "\varphi_n", "I_{x_n}"'] \arrow[d, equal] & T_{n-1} \arrow[r, "\varphi_{n-1}", "I_{x_{n-1}}"'] & \cdots \arrow[r, "\varphi_2", "I_{x_2}"'] & T_1 \arrow[r, "\varphi_1", "I_{x_1}"'] & T_0 \arrow[d, equal] \\
T \arrow[rrrr, "\varphi"] & & & & U .
\end{tikzcd}$$

3.5 Degeneration

We introduce the complementary notion of embedding, called degeneration as follows.

Definition 14 (Degeneration). Let T and U be two trees. A homomorphism $\varphi : T \to U$ is a *degeneration* if the following conditions are satisfied: φ is surjective onto $V(\Im(\varphi))$, $\forall x, y \in V(T)$ $[\varphi(x) = \varphi(y) \Rightarrow \varphi(x \smile y) = \varphi(x)]$, and $\forall x, y \in V(T)$ $[\varphi(x) < \varphi(y) \Rightarrow \exists z \in V(T)$ $[\varphi(y) = \varphi(z) \ \wedge \ x < z]]$.

We refer to $\mathrm{Dup}(\varphi) = \{x \in V(T) | \varphi(x) = \varphi(p(x))\}$ as the *duplication* of the degeneration $\varphi : T \to U$. Figure 3(b) shows an example of a degeneration.

Lemma 2. *Let T and U be two trees. For any degeneration $\varphi : T \to U$, there exists a unique embedding $\psi : \Im(\varphi) \to T$ such that $\varphi \circ \psi$ is the identity mapping on $V(\Im(\varphi))$ and $\psi \circ \varphi$ is the identity mapping on $V(T) \setminus Dup(\varphi)$.*

Proposition 8. *For any degeneration $\varphi : T \to U$, $\varphi(x \smile y) = \varphi(x) \smile \varphi(y)$.*

Definition 15 (Deletion). Let T and U be two trees. A degeneration $\varphi : T \to U$ is called a *deletion* from T if $|\mathrm{Dup}(\varphi)| = 1$. In particular, if a deletion φ is surjective and $\mathrm{Dup}(\varphi) = \{x\}$, φ is called an x-deletion and denoted by D_x.

Proposition 9. *Let T and U be two trees. For $x \in V(T)$, $D_x : T \to U$ satisfies the following properties:*

1. *for any $y \in ch(x)$, $D_x : T(y) \to U(D_x(y))$ is an isomorphism, and*
2. *$D_x : T[\mathbf{x} \not\leq x] \to U[\bigwedge_{y \in ch(x)} \mathbf{x} \not\leq D_x(y)]$ is an isomorphism.*

The following theorem shows that Definition 15 of the deletion is equivalent to the operational definition of the deletion.

Theorem 2 (Decomposition of degeneration). *Let φ be a degeneration from T to U with $Dup(\varphi) = \{x_1, \dots, x_n\}$. There exist a sequence of trees $T_0, T_1, \dots, T_n$, and a sequence of deletions $\varphi_i : T_i \to T_{i+1}$ ($i \in \{0, \dots, n-1\}$) such that $T_0 = T$, $T_n = U$, $Dup(\varphi_{i-1} \circ \cdots \circ \varphi_0) = \{x_1, \dots, x_i\}$, and $\varphi = \varphi_{n-1} \circ \cdots \circ \varphi_0$.*

3.6 Duality Between Embedding and Degeneration

In Lemma 2, we see that, for a given degeneration φ, there exists an embedding ψ such that $\varphi \circ \psi$ is an identity mapping. In fact, its reverse also holds.

Theorem 3. *Let T and U be two trees. The following two properties hold:*

1. *For any degeneration $\varphi : T \to U$, there exists a unique embedding $\psi : \Im(\varphi) \to T$ such that $\varphi \circ \psi$ is the identity mapping on $V(\Im(\varphi))$ and $\psi \circ \varphi$ is the identity mapping on $V(T) \setminus Dup(\varphi)$.*
2. *For any embedding $\psi : U \to T$, there exists a unique degeneration $\varphi : \Im(\psi) \to U$ such that $\varphi \circ \psi$ is the identity mapping on $V(U)$ and $\psi \circ \varphi$ is the identity mapping on $V(\Im(\psi)) \setminus Dup(\varphi)$.*

This theorem is to the effect that there exists a unique degeneration $\bar{\psi}$ (an embedding $\bar{\varphi}$, resp.) if an embedding ψ (a degeneration φ, resp.) is given.

3.7 Characterization of Alignment of Trees

Now we are ready to give a definition of the alignment of trees in a formal manner.

Throughout in this section, S and T are trees, and $M \subseteq V(S) \times V(T)$ is a tree mapping from S to T.

Definition 16. A tree mapping M from S to T is an *alignable* if and only if there exists a triplet (U, φ, ψ) such that

1. $\varphi : S \to U$ is an embedding,
2. $\psi : T \to U$ is an embedding, and
3. $\varphi(x) = \psi(y)$ for all $(x, y) \in M$;

$$\begin{tikzcd} & U & \\ S \arrow[ur, "\varphi"] \arrow[rr, "M"] & & T. \arrow[ul, "\psi"'] \end{tikzcd}$$

We call (U, φ, ψ) a *union* on M.

Lemma 3. *Let M be an alignable mapping with a union (U, φ, ψ). For any $(x, y) \in M$, the equation $(x, y) = (x, \bar{\psi}(\varphi(y)))$ holds, where $\bar{\psi}$ is the degeneration such that $\bar{\psi} \circ \psi$ is the identity mapping of T.*

Proof. The assertion is obvious since $\bar{\psi} \circ \psi$ is the identity mapping of T. □

Proposition 10. *Let S and T be two trees. Any singleton tree mapping $M = \{(x, y)\}$ from S to T is alignable.*

Proof. By Definition 16, this assertion is intuitively obvious. (We omit this proof due to the space limitation.) □

Lemma 4. *Let $\eta : S \to \bar{S}$ is an embedding. For a tree mapping M, M is alignable if and only if $\bar{M} = \{(\eta(x), y) | (x, y) \in M\}$ is alignable.*

Proof. By Definition 16, the if-part of this lemma is obvious, and the proof of the only-if-part is omitted due to the space limitation. □

This lemma implies that if M is alignable after inserting nodes, it is also alignable without the insertion.

Lemma 5. *Let M' be a subset of M. If M is alignable, then M' is also alignable.*

Proof. A union on M is also a union on M'. □

By the definition of tree mapping, for $(s, t) \in M$, if $s = r(S)$, then $t = r(T)$.

Lemma 6. *Let (U, φ, ψ) be a union on M. Then, there exist φ' and ψ' such that (U, φ', ψ') is also a union on M and $\varphi'(r(S)) = \psi'(r(T))$. In particular, M is alignable if and only if $M \cup \{(r(S), r(T))\}$ is also alignable.*

Proof. Let $(s, t) \in M$. $\varphi(r(S))$ and $\psi(r(T))$ are comparable, since they are ancestors of $\varphi(s) = \psi(t)$. If $\varphi(r(S)) = \psi(r(T))$, there is nothing to prove. Without loss of generality, we may assume that $\varphi(r(S)) < \psi(r(T))$. Define $\varphi' : V(S) \to V(U)$ by $\varphi'(x) = \varphi(x)$ if $x \neq r(S)$ and $\varphi'(r(S)) = \psi(r(T))$. In the following, we see that φ' is an embedding. First, let $x, y \in V(S)$ satisfy

$x < y$. If $y \neq r(S)$, $\varphi'(x) < \varphi'(y)$ holds since φ is a homomorphism. If $y = r(S)$, $\varphi'(x) = \varphi(x) < \varphi(r(S)) < \varphi'(r(S))$ holds. Thus, φ' is a homomorphism. The property $x < y$ if $\varphi'(x) < \varphi'(y)$ is also easily proved. Consequently, we see that φ' is an embedding.

Since the if-part of this lemma follows from Lemma 5, it suffices to show the only-if-part. As shown in the first part, if (U, φ, ψ), we have another union (U, φ', ψ') such that $\varphi'(r(S)) = \psi'(r(T))$. Therefore, $M \cup \{(r(S),\ r(T))\}$ is alignable. □

Lemma 7. *Let S_i denote the tree $S(\sigma_i)$ for ch$(r(S)) = \{\sigma_1, \dots, \sigma_m\}$, T_i the tree $T(\tau_i)$ for ch$(r(T)) = \{\tau_1, \dots, \tau_n\}$. By symmetry, we assume that $m \leq n$. By $M_i \subset V(S_i) \times V(T_i)$ for $i \in \{1, \dots, m\}$, we denote the tree mapping $\{(s,t) \in M | s \in V(S_i)$ and $t \in V(T_i)\}$. If $M = \bigcup_{i=1}^{m} M_i$ and each M_i is alignable, then M is also alignable.*

4 Equivalence Between Alignable Mapping and Less-Constrained Mapping

Now we are ready to prove our main theorem.

Theorem 4. *For any tree mapping M, M is alignable if and only if M is less-constrained.*

Proof. **(only-if part):** Let (U, φ, ψ) be a union on M. Hence, $\varphi : S \to U$ and $\psi : U \to T$ are embeddings such that $\varphi(s) = \psi(t)$ for any $(s,t) \in M$. Further, $\bar{\psi}$ denote the degeneration such that $\bar{\psi} \circ \psi$ is the identity mapping of T (Theorem 3). Suppose that (s_1, t_1), (s_2, t_2), and (s_3, t_3) are any three elements of M such that $s_1 \smile s_2 < s_1 \smile s_3$. We have $\varphi(s_1) \smile \varphi(s_2) < \varphi(s_1) \smile \varphi(s_3)$ by Corollary 2, and therefore $\varphi(s_2) \smile \varphi(s_3) = \varphi(s_1) \smile \varphi(s_3)$. Also, we have $\bar{\psi}(\varphi(s_2)) \smile \bar{\psi}(\varphi(s_3)) = \bar{\psi}(\varphi(s_2) \smile \varphi(s_3)) = \bar{\psi}(\varphi(s_1) \smile \varphi(s_3)) = \bar{\psi}(\varphi(s_1)) \smile \bar{\psi}(\varphi(s_3))$ by Proposition 8. Since $\bar{\psi}(\varphi(s_1)) = t_1$, $\bar{\psi}(\varphi(s_2)) = t_2$ and $\bar{\psi}(\varphi(s_3)) = t_3$ hold by Lemma 3, we conclude that $t_2 \smile t_3 = t_1 \smile t_3$. Derivation of $s_2 \smile s_3 = s_1 \smile s_3$ from $t_1 \smile t_2 < t_1 \smile t_3$ is shown in the same way.

(if part): The assertion in the case of $|M| = 1$ directly follows from Proposition 10.

Let $|M| \geq 2$ for the induction step. Let M be the set of node pairs $\{(s_1, t_1), \dots, (s_n, t_n)\}$, $X \subseteq V(S)$ denote the set of nodes $\{s_1, \dots, s_n\}$, and $Y \subseteq V(T)$ denote the set of nodes $\{t_1, \dots, t_n\}$.

It suffices to prove the assertion of the theorem under the hypothesis that $\mathrm{lca}(X) = r(S)$ and $\mathrm{lca}(Y) = r(T)$. In fact, for the embeddings $\alpha = \mathbf{E}_{\mathbf{x} \leq \mathrm{lca}(X)} : S[\mathbf{x} \leq \mathrm{lca}(X)] \to S$ and $\beta = \mathbf{E}_{\mathbf{x} \leq \mathrm{lca}(Y)} : T(\mathrm{lca}(Y)) \to T$, by Lemma 4, if $M' = \{(\alpha^{-1}(s), \beta^{-1}(t)) | (s,t) \in M\}$ is alignable, then M is alignable. We may also assume that M does not contain $(r(S), r(T))$, since, if M contains it, it suffices to eliminate it by Lemma 6.

We now choose $X_k = \{s_1, \dots, s_k\}$, by reordering s_i's if necessary, such that (1) $k \geq 1$, (2) $\mathrm{lca}(X_k)$ is not the root of S, and (3) for any $x \in X \setminus X_k$, $\mathrm{lca}(X_k \cup \{x\}) = r(S)$.

Note that $k < n$. Let us denote by Y_k the set of nodes $\{t_1, \ldots, t_k\}$ corresponding to X_k.

Claim 1. For any $i \leq k$ and $j > k$, $s_i \smile s_j$ is the root of S.

Proof. The two nodes $s_i \smile s_j$ and $\mathrm{lca}(S_k)$ are comparable since $s_i \in X_k$. Now assume that $s_i \smile s_j \leq \mathrm{lca}(X_k)$. It follows that $\mathrm{lca}(S_k \cup \{s_j\}) = \mathrm{lca}(X_k)$. This contradicts the definition of X_k. Hence $\mathrm{lca}(X_k) < s_i \smile s_j$, and in particular $s_i \smile s_j = \mathrm{lca}(X_k \cup \{s_j\})$. This implies that $s_i \smile s_j$ is the root of S. □

Let $A = \{x \in \mathrm{ch}(r(S)) | \exists i[1 \leq i \leq k \ \wedge \ s_i \leq x]\}$ and $B = \{x \in \mathrm{ch}(r(S)) | \exists j[k < j \leq n \ \wedge \ s_j \leq x]\}$. We have $A \cap B = \emptyset$, since, if $x \in A \cap B$, we have $s_i \smile s_j \leq x$ for $1 \leq i \leq k$ and $k < j \leq n$, as is contradictory to Claim 1.

Thus, by inserting nodes as children of $r(S)$ if necessary, we may assume the following properties: (1) the children of $r(S)$ are only two nodes a and b, (2) $\mathrm{lca}(S_k) \leq a$, and (3) $\mathrm{lca}(X \setminus S_k) \leq b$.

Now, to apply similar proof to Y_k, we claim the following.

Claim 2. For any $i \leq k$ and $j > k$, $t_i \smile t_j$ is the root of T.
(This proof is similar to Claim 1. So, we omit this proof.)

Therefore, in the same way as the case of S, by inserting nodes as children of $r(T)$ if necessary, we may assume the following properties by Lem 4: (1) the children of $r(T)$ are only two nodes α and β, (2) $\mathrm{lca}(Y_k) \leq \alpha$, and (3) $\mathrm{lca}(Y \setminus Y_k) \leq \beta$.

By the induction hypothesis, M_k=$\{(s_1, t_1), \ldots, (s_k, t_k)\}$ is an alignable mapping from $S(a)$ to $T(\alpha)$, and $M \setminus M_k$ is alignable from $S(b)$ to $T(\beta)$. Then, by Lemma 7, M is alignable from S to T. □

The size of a tree T is the number of nodes in T, denoted by $|T|$. We denote the maximum number of children for all nodes in a tree T by $\deg(T)$. For ordered trees, an algorithm for computing a less-constrained edit distance was presented by Lu *et al.*[11]. The time complexity of the algorithm is, for two trees T and U, $O(|T| \cdot |U| \cdot \deg(T)^3 \cdot \deg(U)^3 \cdot (\deg(T) + \deg(U)))$. By Theorem 4, we can immediately improve this algorithm because there is a more efficient algorithm for computing an alignment of trees by [7]. The time complexity is $O(|T| \cdot |U| \cdot (\deg(T) + \deg(U))^2)$.

Recall that Jiang *et al.* showed that the alignment problem for two unordered trees is MAX-SNP hard [7]. Furthermore, we obtain a more negative result for the alignment of trees because Lu *et al.* showed that the less-constrained edit distance problem for unordered trees has no polynomial-time absolute approximation algorithm [11], *i.e.*, the solution is not within an additive constant of the optimum, unless P = NP. Then we immediately have the next corollary.

Corollary 3. *The alignment problem for two unordered trees has no polynomial-time absolute approximation algorithm, unless P = NP.*

5 Conclusion

In this paper, we have presented a new theoretical formulation of tree edit problems in approximate tree matching as a unifying framework. This framework enables us to describe distinct semantics for approximate tree matching, and study the matching properties of tree edit methods. We have mainly focused on two tree edit methods, the alignment of trees and the less-constrained edit, which have been independently proposed, but the relationship between them remained unnoticed. By using our formulation, we have clarified the semantics of these notions. We then revised the definition of less-constrained edit in the original work. Finally, we have proved an important relationship between these two notions. That is, we have showed that the alignment of trees is essentially equivalent to the less-constrained edit. This result implies that finding a common supertree pattern is a subclass problem of finding a common subtree pattern.

References

1. Sakakibara, Y.: Pair hidden markov models on tree structures. Bioinformatics **19** (2003) 232–240
2. Torsello, A., Hancock, E.R.: Matching and embedding through edit-union of trees. In: Proc. of ECCV 2002. Volume 2352 of LNCS. (2002) 822–836
3. Ferraro, P., Godin, C.: A distance measure between plant architectures. Annals of Forest Science **57** (2000) 445–461
4. Hogue, A., Karger, D.: Thresher: Automating the unwrapping of semantic content from the world wide web. In: Proc. of WWW 2005. (2005) 86–95
5. Tai, K.C.: The tree-to-tree correction problem. Journal of the ACM **26** (1979) 422–433
6. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing **18** (1989) 1245–1262
7. Jiang, T., Wang, L., Zhang, K.: Alignment of trees — an alternative to tree edit. Theoretical Computer Science **143** (1995) 137–148
8. Wagner, R., Fischer, M.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–173
9. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
10. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science **337** (2005) 217–239
11. Lu, C.L., Su, Z.Y., Tang, G.Y.: A new measure of edit distance between labeled trees. In: Proc. of COCOON 2001. Volume 2108 of LNCS. (2001) pp. 338–348
12. Zhang, K.: A constrained edit distance between unordered labeled trees. Algorithmica **15** (1996) 205–222
13. Zhang, K.: Algorithms for the constrained editing distance between ordered labeled trees and related problems. Pattern Recognition **28** (1995) 463–474
14. Tanaka, E., Tanaka, K.: The tree-to-tree editing problem. International Journal of Pattern Recognition and Artificial Intelligence **2** (1988) 221–240
15. Richter, T.: A new measure of the distance between ordered trees and its applications. Technical Report 85166-CS, Dept. of Computer Science, Univ. of Bonn (1997)
16. Wang, J.L., Zhang, K.: Finding similar consensus between trees: an algorithm and a distance hierarchy. Pattern Recognition **34** (2001) 127–137

A Complete Formulation of Generalized Affine Equivalence

Marco Macchetti, Mario Caironi, Luca Breveglieri, and Alessandra Cherubini

Politecnico di Milano, Milan, Italy
{macchett, caironi, brevegli}@elet.polimi.it, aleche@mate.polimi.it

Abstract. In this paper we present an extension of the generalized linear equivalence relation, proposed in [7]. This mathematical tool can be helpful for the classification of non-linear functions $f : F_p^m \rightarrow F_p^n$ based on their cryptographic properties. It thus can have relevance in the design criteria for substitution boxes (S-boxes), the latter being commonly used to achieve non-linearity in most symmetric key algorithms. First, we introduce a simple but effective representation of the cryptographic properties of S-box functions when the characteristic of the underlying finite field is odd; following this line, we adapt the linear cryptanalysis technique, providing a generalization of Matsui's lemma. This is done in order to complete the proof of Theorem 2 in [7], also by considering the broader class of generalized affine transformations. We believe that the present work can be a step towards the extension of known cryptanalytic techniques and concepts to finite fields with odd characteristic.

Keywords: Boolean functions, generalized linear equivalence, linear cryptanalysis, S-boxes.

1 Introduction

Symmetric key cryptographic algorithms play a crucial role in today's secure communication protocols and secure storage applications, due to their high efficiency and key-agility. The class of block ciphers has recently known a flourishing of proposals, also due to the Advanced Encryption Standard establishment process.

Block ciphers are usually characterized by an iterative nature: a constant set of transformations, called *round* or more generically step, is applied several times to the plaintext block in order to obtain the corresponding ciphertext. The round transformation must necessarily possess several properties in order to enforce the robustness of the whole algorithm, and to maximize efficiency: it must be key-dependent, it should be highly non-linear and it should guarantee a high level of diffusion of information.

These constraints must be satisfied regardless of the block cipher structural scheme, e.g. they are valid for Feistel networks [1], Lai-Massey [13] and Substitution-permutation networks [8]. A common and well-studied method to implement the non-linear step is to use bricklayer functions composed by S-boxes. These are usually defined over the binary domain, i.e. $S : F_2^m \rightarrow F_2^n$ and they are chosen such that their cryptographic characteristics are optimal.

Several works have been focused on the characterization of the non-linear properties of S-boxes, some examples being [3],[16],[17],[18], and on the possibility of partitioning them into equivalence classes [14],[11],[12]; recent papers propose efficient algorithms that can be used to decide if two S-boxes [6] or two Boolean functions [10] are linearly equivalent. This research activity is motivated by the relevant link between the properties of S-boxes and the security and efficiency of block ciphers.

Two attacks that are particularly relevant for block ciphers are linear cryptanalysis [15],[4] and differential cryptanalysis [5], thus most of the efforts have been directed to the problem of finding S-boxes with optimal differential and linear characteristics.

In [7], Breveglieri, Cherubini and Macchetti proposed an extension of the criterion of functional linear equivalence called *generalized linear equivalence*; it has been shown that generalized equivalence classes result from merging of classical equivalence classes and that linear and differential characteristics are indeed invariant under this broader class of transformation.

The aim of this paper is double fold. First, we elaborate more on the theoretical basis of the original formulation of generalized equivalence; in fact, although it has been proved that the linear characteristics of S-boxes are invariant in the context of these transformations, the proof, as it is, is rigorously valid only for fields of even characteristic. No formalization of the linear cryptanalysis technique over fields with odd characteristic can be found in the scientific literature. This is provided in this paper, along with the corresponding generalization of Matsui's lemma. The original proof of invariance is then completely and coherently derived.

A second contribution is the introduction of generalized *affine* transformations; these are indeed the natural extension of classical affine transformations and complete the results of [7]. The proof of invariance for S-box cryptographic robustness is thus obtained within the largest possible context.

This paper is organized as follows: in Section 2 we define and analyze the linear characteristics of S-boxes defined over finite fields of odd characteristic. In Section 3 we extend the linear cryptanalysis technique, providing a generalization of Matsui's lemma. In Section 4 we give a complete proof for the invariance of cryptographic robustness of S-boxes, also considering generalized affine transformations. Section 5 concludes the paper.

2 Linear Biases over F_p^m

Broadly speaking, the goal of symmetric-key cryptanalysis is to distinguish a block cipher from a set of random permutations, and to get information on key material faster than it could be done via a trivial brute-force attack.

In the case of differential cryptanalysis[1], the suggested distinguisher is the probability of finding certain differences in the ciphertext blocks, given certain

[1] We assume here that the reader has some basic familiarity with the ideas beyond differential and linear cryptanalysis.

differences in the corresponding plaintexts. If this probability deviates significantly from what would be expected from a random permutation, the attack is successful and information about the key can be found.

Differential characteristics for N rounds of a block cipher can be constructed starting from differential characteristics of the single S-boxes; these in fact are usually the only non-linear component of a symmetric key algorithm. Conventional algorithms are defined over the finite field F_2 for evident reasons of efficiency. However, it is easy to extend the differential cryptanalysis technique to finite fields of odd characteristic; this extension is of theoretical interest as it may be used to test the cryptographic robustness of basic arithmetic operations, such as multiplication, inversion, and power functions defined over a field F_{p^m}. Recent work has been done by Dobbertin [9] regarding the problem of finding power monomials in such fields with optimal differential characteristics.

For a given S-box function $f : F_p^m \to F_p^n$ with p prime and $m, n > 1$ the Difference Distribution Table (DDT) is built by computing the number $\delta_f(a, b)$ of solutions x of the equation

$$f(x \oplus a) \ominus f(x) = b \qquad a \in F_p^m, b \in F_p^n \tag{1}$$

where $\oplus$ and $\ominus$ respectively indicate sum over F_{p^m}, the finite field associated with the vector space F_p^m, and difference over F_{p^n}. The lower the value of the maximum entry in the table, $\Delta_f = \max_{a \neq 0, b}(\delta_f(a, b))$, the more robust function f is versus differential cryptanalysis, since the differential characteristics $\delta_f(a, b)$ of S-boxes located in different rounds of the cipher are, roughly speaking, connected to form a multi-round characteristic. The amount of plaintext-ciphertext block pairs needed to highlight a hypothetical differential bias is inversely proportional to its magnitude.

The linear cryptanalysis technique is built upon a very similar concept, namely that of linear distinguisher. The objective of linear cryptanalysis is to build linear (over F_2) equations involving plaintext, ciphertext and key bits that hold with a probability significantly different from 50%. If this is possible for a high number of rounds, then the attack may reveal information about the key bits faster than brute-force attacks (this indeed is the case for the DES cipher). Again, linear characteristics for a full cipher are built starting from those of the single S-boxes.

More formally, the Linear Approximation Table (LAT) of an S-box function $f : F_2^m \to F_2^n$ is built by counting the number $\lambda_f(a, b)$ of solutions x of the equation

$$a \bullet x = b \bullet f(x) \qquad a \in {F_2}^m, b \in {F_2}^n \tag{2}$$

where the inner product over F_2^m and F_2^n is indicated with $\bullet$ and gives a value in F_2. The robustness to linear cryptanalysis is measured with the maximum value $\Lambda_f = \max_{a, b \neq 0}(||\lambda_f(a, b) - 2^{m-1}||)$. In fact, the event when the number of solutions of (2) is always very near to 2^{m-1} is the best case for the designer and the worst case for the attacker; this is because a random Boolean function is expected to be equal to any linear Boolean function for roughly half of the points of its domain, i.e. 2^{m-1} times in the specific context[2]. The attacker can then infer nothing about the function, apart from the fact that it behaves randomly, a thing which does not help in distinguishing the block cipher from a random permutation.

Several extensions of linear cryptanalysis are known, see for instance [2] which contains also a very good survey. An extension of linear cryptanalysis to finite fields of odd characteristic may be beneficial for the same motivations outlined above for the differential case.

We start by defining the affine biases for an S-box over F_p. Let $f : F_p^m \to F_p^n$ be an S-box function, then we introduce the Affine Approximation Table (AAT) of f, which is built by counting the number $\lambda_f(a, b, c)$ of solutions x of the equation

$$a \bullet x \oplus b \bullet f(x) = c \qquad a \in {F_p}^m, b \in {F_p}^n, c \in F_p \tag{3}$$

The constant c in (3) is introduced to take into account the fact that the inner product now gives a value in F_p, and for this reason it is not sufficient to compare f to all the linear functions, and in fact all affine functions must be taken into consideration. Another way of looking at (3) is to say that we count the number of times that function $b \bullet f(x)$ is equal to the affine function $(-a) \bullet x \oplus c$, hence the name AAT.

Each cell of the AAT of f is indexed by the triplet $\{a, b, c\}$ and the AAT is indeed a three-dimensional array rather than a table, still we keep the old terminology for clearness. The value of interest to the cryptanalyst becomes in this case $\lambda_f(a, b, c) - p^{m-1}$; the reason is that a random function over the range F_p is expected to be equal to any affine function in roughly one case out of p, and since the cardinality of the domain of f is p^m a simple division gives the expected result p^{m-1}. The overall robustness of the function can then be characterized with the parameter $\Lambda_f = \max_{a,b\neq 0,c}(||\lambda_f(a, b, c) - p^{m-1}||)$.

In the binary case there is no need to consider affine functions, because the number of solutions of equation

$$a \bullet x \oplus b \bullet f(x) = c \qquad a \in {F_2}^m, b \in {F_2}^n, c \in F_2 \tag{4}$$

is equal to the number of solutions for the pair $\{a, -b\}$ in (2) when $c = 0$ and is equal to the difference between 2^m and the previous number when $c = 1$. In a sense, in fields of even characteristic, affine distinguishers are totally redundant and give no advantage to the attacker in addition to linear distinguishers.

Even in the odd characteristic case a partial redundancy is present, because it must hold that

$$\sum_{c=0}^{p-1} \lambda_f(a, b, c) = p^m \tag{5}$$

and this implies that one value out of p in the AAT is redundant. At this point it is useful to give a visual representation of the AAT, since this will also be

[2] The functions for which $\Lambda_f = 2^{\frac{m}{2}-1}$ are called Bent, and exist only under additional hypotheses on the number of input/output variables.

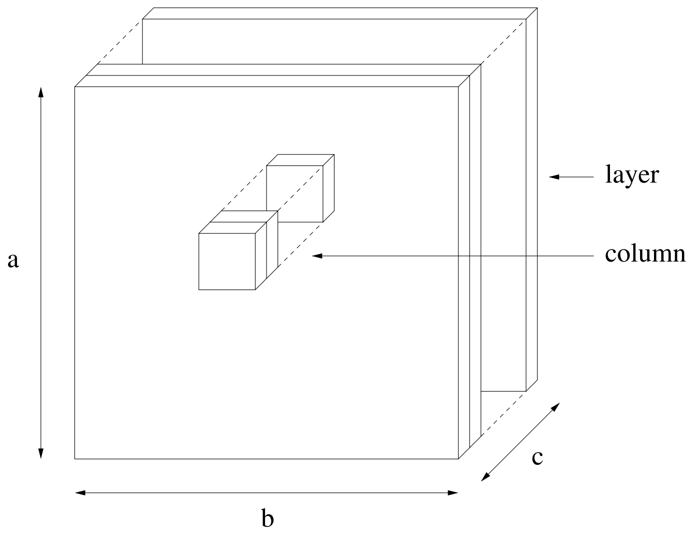


Fig. 1. A graphical representation of the Affine Approximation Table

useful in the following Sections to understand what is the concrete effect of generalized linear and affine transformations. Figure 1 depicts the AAT of a generic S-box function f; the values of the three parameters a, b, c vary along the three different axes of the table. We call *layers* the set of cells with constant index c, and *columns* the set of cells with constant a, b indexes; these structures are highlighted in the Figure.

Elaborating on these definitions, we can say that one out of the p layers is then redundant, because essentially all the information is already contained in the remaining $p - 1$ ones. The choice of the redundant layer is arbitrary, thus in the following discussion we will assume that all the layers are maintained. It is clear that the number of possible biases for S-boxes defined over F_p is higher than that of S-boxes defined over the field F_2; the latter only have two layers in the AAT. This means that more computational effort is in general required to calculate the AAT versus the LAT, but also that the cryptanalyst may have more freedom in the choice of the biases to be used in an *affine* attack.

In the next Section we will see how the AATs can be used in such a generalization of the linear cryptanalysis attack.

3 Extending Linear Cryptanalysis

The next step towards a complete formulation of linear cryptanalysis on fields with odd characteristic is the extension of Matsui's Piling-Up Lemma [15]; this is classically used to obtain the bias of a sum of linearly biased variables, even if it is rigorously valid only if the variables are strictly uncorrelated.

An extension of Matsui's Lemma has been proposed in [2], in the context of a specific variant of linear cryptanalysis: the input/output Boolean sums are substituted with linear (and non-linear) projections over F_2^l. The formulation is indeed quite complex; our goal here is rather to obtain a simple formula, involving only the affine biases, which is the direct extension of that of Matsui.

Let X_1 be a variable with values over F_p; the affine bias ϵ_1^i is defined via the following equation:

$$\Pr(X_1 = i) = \frac{1}{p} + \epsilon_1^i \qquad i \in F_p \tag{6}$$

Thus a vector of affine biases $\Xi_1 =< \epsilon_1^0, \ldots, \epsilon_1^{p-1} >$ is associated with X_1. Now, we want to be able to compute the affine bias vector of a sum of two such variables starting from the two single bias vectors; this is done in the following calculations, where all the sums performed over F_p are indicated with $\oplus$.

$$\begin{aligned}\Pr(X_1 \oplus X_2 = k) &= \sum_{i \oplus j = k} (\frac{1}{p} + \epsilon_1^i)(\frac{1}{p} + \epsilon_2^j) = \\ &= \sum_{i \oplus j = k} \frac{1}{p^2} + \frac{1}{p} \sum_{i \oplus j = k} \epsilon_1^i + \frac{1}{p} \sum_{i \oplus j = k} \epsilon_2^j + \sum_{i \oplus j = k} \epsilon_1^i \epsilon_2^j = \\ &= \frac{1}{p} + \sum_{i \oplus j = k} \epsilon_1^i \epsilon_2^j \end{aligned} \tag{7}$$

By re-writing the last passage using only the affine biases we obtain:

$$\epsilon_{1,2}^k = \sum_{i \oplus j = k} \epsilon_1^i \epsilon_2^j \tag{8}$$

which is a direct generalization of Matsui's formula for the sum of two variables; moreover, the link between the affine bias vectors is given by:

$$\Xi_{1,2} = \Xi_1 \star \Xi_2 \tag{9}$$

where $\star$ stands for a variant of the discrete convolution operation where the sum and differences of vector indexes are computed over F_p. We note that both (8) and (9) could be easily extended to a number $n > 2$ of variables, and that they reduce to the well-known formula for linear cryptanalysis if $p = 2$.

In the proposed extension of linear cryptanalysis the variables X_i are indeed approximations of the non-linear components of the algorithm under consideration, typically the active S-boxes, i.e. they will have the form of (3). Thus, the affine bias vector Ξ_i is indeed nothing but a column of the AAT of f, indexed by the specific values of a, b. We think that the name *affine cryptanalysis* could be effectively used to identify the extension of linear cryptanalysis to fields of odd characteristic.

An outline of Matsui's algorithm 2 targeting a cryptographic algorithm which operates on the base field F_p is roughly as follows:

1. The attacker chooses an affine characteristic over $N-1$ rounds of the cipher; the affine distinguishers of the active S-boxes are calculated and stored in the AATs.

2. The attacker chooses the affine approximations for all active S-boxes, i.e. the values of the parameters a, b are selected for each active S-box.
3. Equation (9) can then be used to calculate the affine bias vector of the global characteristic, Ξ_T.
4. Last-round decryptions of the ciphertexts will eventually reveal the bias of the affine approximation under the correct key hypothesis.

We underline a difference with regard to the case of even characteristic. The maximum affine bias inside Ξ_T must always be searched for and identified; this happens because the position of the ϵ_T^i with maximum (minimum) value depends on the key material which is added along the affine trail. This has the effect of changing the i in a key-dependent way, and roughly increases the complexity of the attack by a factor of p compared to standard linear cryptanalysis.

An anonymous referee has pointed out that in the case of composite fields $F_{p^{mn}}$, the field F_{p^m} can be taken as a base field in place of F_p and all equations could be re-written properly to obtain affine approximations at a higher level. The constant c would in this case belong to F_{p^m}, and the inner product in (3) would be modified accordingly. This leads to an interesting formulation of the affine biases that may have practical applications in the case of composite fields with even characteristic.

4 A Complete Formulation of Generalized Equivalence

4.1 An Extended Proof

Given the previous background, it is now possible to give a coherent proof of Theorem 2 in [7]. The need for a formal extension basically derives from the fact the original formulation of generalized linear equivalence does not take into account the differences between linear cryptanalysis and affine cryptanalysis. The part about differential characteristics remains unchanged and will not be repeated here (it will be expanded when generalized affine transformation are considered).

We summarize here the basics of the approach outlined in [7]. It is possible to associate a particular geometric representation to any completely specified function $f : F_p^m \rightarrow F_p^n$. Let S be a linear space of dimension $k = m + n$, where the vector components are defined over F_p; consider the set $\mathcal{F}$ of p^m vectors, belonging to S, formed by the rows of the truth-table of f (each vector belonging to $\mathcal{F}$ is the concatenation of an input vector of f and its corresponding output vector). We refer to $\mathcal{F}$ as the implicit embedding of f in the linear space S.

If an invertible linear transformation of coordinates is applied to S, the essential information contained in $\mathcal{F}$ is not changed. Every such invertible linear transformation is governed by a non-singular $(m + n) \times (m + n)$ matrix over F_p. The non-singularity of this matrix, while providing the possibility to invert the transformation, also assures that we do not loose information while transforming the coordinates. The extended Theorem follows.

Theorem 1. *Given two functions $f, g : {F_p}^m \rightarrow {F_p}^n$ and a non-singular $(m+n) \times (m+n)$ matrix T over F_p, if $g = T(f)$ then the distributions of values in the AATs of f and g are equal.*

Proof. A cell of the AAT table of f indexed by $\{a, b, c\}$ contains the number of input vectors x such that $a^T \bullet x \oplus b^T \bullet f(x) = c$, where, for sake of clearness, the transposed of vector v is indicated with v^T.
Thus, if we consider the geometric representation for function f we have that the cell contains the number of vectors w belonging to the implicit embedding of f such that $k^T \bullet w = c$ where $k = (a)|(b)$ (the concatenation of vectors a and b); note that $a \in {F_p}^m$, $b \in {F_p}^n$ and $k \in {F_p}^{m+n}$. The merged index k is unique for every column in the AAT of the two functions.
These vectors will be transformed by the change of basis into other vectors w' belonging to the implicit embedding of function g such that $w' = Tw$. We can rewrite the equation as:

$$k^T \bullet Tw = c \quad \Leftrightarrow \quad (T^T k)^T \bullet w = c \quad \Leftrightarrow \quad (k')^T \bullet w = c$$

Since matrix T is non-singular, there is a bijection between the values of k and those of $k' = T^T k$, i.e. the cells of the AAT of g are just a (linear) rearrangement of the cells of the AAT of f. □

Note that the cells belonging to a given layer cannot be shifted to different layers; indeed the cells of all the layers are reordered in a uniform way, given only by matrix T. The question if there are even more general transformations can thus arise, and we positively answer in the following Section.

4.2 Generalized Affine Transformations

We introduce the following functional equivalence relation.

Definition 1. *Two functions $f, g : F_p^m \rightarrow F_p^n$ are called generally affine equivalent if and only if the implicit embedding of g can be obtained from the implicit embedding of f as*

$$\mathcal{G} = T(\mathcal{F}) \oplus e \tag{10}$$

where T is a non-singular $(m+n) \times (m+n)$ matrix over F_p and e is a vector belonging to F_p^{m+n}.

This is the natural and most elegant definition of generalized equivalence. A proof of invariance for the cryptographic characteristics of functions f, g is given in the following theorem.

Theorem 2. *Given two functions $f, g : F_p^m \rightarrow F_p^n$, a non-singular $(m+n) \times (m+n)$ matrix T and a constant vector $e \in F_p^{m+n}$, if $g = T(f) \oplus e$ then the distributions of values in the AATs and DDTs of f and g are equal.*

Proof. We first prove the relation regarding the DDTs of f and g.

A cell of the DDT of f located in the i-th row and in the j-th column contains the number of the input vector pairs (x, y) such that $y = x \oplus i$ and $f(y) = f(x) \oplus j$.

Thus, if we consider the geometric representation for function f we have that the cell contains the number of vector pairs (w, z) belonging to the implicit embedding of f such that $w = z \oplus k$ where $k = (i)|(j)$; note that $i \in F_p^m$, $j \in F_p^n$ and $k \in F_p^{m+n}$. These pairs will be transformed by the change of basis into other pairs (w', z') belonging to the implicit embedding of function g such that $w' = Tw \oplus e$, $z' = Tz \oplus e$. If we define k' equal to $w' - z'$, we obtain that $k' = Tk$ and the relation $w' = z' \oplus k'$ holds. Since matrix T is non-singular, there is a bijection between the values of k and those of k': this means that the cells of the DDT of g are just a rearrangement of the cells of the DDT of f.

Now we prove the relation between the AATs.

A cell of the AAT of f located in the column indexed by a, b and in the $c-th$ layer contains the number of input vectors x such that $a^T \bullet x \oplus b^T \bullet f(x) = c$. Thus, if we consider the geometric representation of function f we have that the cell contains the number of vectors w belonging to the implicit embedding of f such that $k^T \bullet w = c$ where $k = (i)|(j)$; note that $a \in F_p^m$, $b \in F_p^n$, $k \in F_p^{m+n}$ and $c \in F_p$. These vectors will be transformed by the change of basis into other vectors w' belonging to the implicit embedding of function g such that $w' = Tw \oplus e$. We can rewrite the equation as:

$$k^T \bullet (Tw \oplus e) = c \iff (T^T k)^T \bullet w = c \ominus k^T \bullet e \iff (k')^T \bullet w = c' \tag{11}$$

Given the non-singularity of matrix T, we have a bijection between the values of k and k'; moreover, for fixed k the values of c and those of c' are bound by a permutation. Thus (11) states that the cells of the AAT of g are an (affine) rearrangement of the cells of the AAT of f. □

The reordering of the cells in the AAT is now more complex than in the preceding case, because here the cells can also migrate among the different layers due to the presence of e; actually the columns of the AAT of f are permuted, and the cells inside each column are also re-ordered in a column-specific way.

The nature of the transformation defined in (10) is quite general; an open question is if this is indeed the most general instance of affine functional equivalence relation.

5 Conclusions

In this paper we have given an extension of the generalized equivalence relation between functions defined over finite fields; the generalized affine transformations are the most general instance of equivalence relations proposed so far in the scientific literature.

As a side result, we have derived an extension of the linear cryptanalysis technique that is applicable to finite fields of odd characteristic; this may be practically useful to test the cryptographic robustness of arithmetic operations (and cryptographic algorithms) defined over such fields. The extension has been named affine cryptanalysis.

References

1. Announcing the Standard for DATA ENCRYPTION STANDARD (DES). FIPS Publication 46-2, NIST, 1993.
2. Baignéres, T., Junod, P., Vaudenay, S.: How Far Can We Go Beyond Linear Cryptanalysis? Proceedings of ASIACRYPT 2004, 432-450, 2004.
3. Beth, T., Ding, C.: On Almost Perfect Nonlinear Permutations. Proceedings of EUROCRYPT '93, 65–76, 1994.
4. Biham, E.: On Matsui's Linear Cryptanalysis. Proceedings of EUROCRYPT '94, 341–355, 1994.
5. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, 4(1):3–72, 1991.
6. Biryukov, A., De Canniere, C., Braeken, A., Preneel, B.: A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. Proceedings of EUROCRYPT 2003, 33–50, 2003.
7. Breveglieri, L., Cherubini, A., Macchetti, M.: On the Generalized Linear Equivalence of Functions over Finite Fields. Proceedings of ASIACRYPT 2004, 79–91, 2004.
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer-Verlag, 2002.
9. Dobbertin, H., Mills, D., Muller, E.N., Pott, A., Willems, W.: APN functions in odd characteristic. Discrete Mathematics, 267(1-3):95–112, 2003.
10. Fuller, J., Millan, W.,: Linear Redundancy in S-Boxes. Proceedings of FSE 2003, 74–86, 2003.
11. Harrison, M.A.: The Number of Classes of Invertible Boolean Functions. Journal of ACM, 10:25–28, 1963.
12. Harrison, M.A.: On Asymptotic Estimates in Switching and Automata Theory. Journal of ACM, 13(1):151–157, 1966.
13. Junod, P., Vaudenay, S.: FOX : A New Family of Block Ciphers. Proceedings of SAC 2004, 114–129, 2004.
14. Lorens, C.S.: Invertible Boolean Functions. IEEE Transactions on Electronic Computers, EC-13:529–541, 1964.
15. Matsui, M.: Linear Cryptanalysis method for DES cipher. Proceedings of EUROCRYPT '93, 386–397, 1994.
16. Nyberg, K.: Differentially Uniform Mappings for Cryptography. Proceedings of EUROCRYPT '93, 55–64, 1994.
17. Nyberg, K.: Perfect Nonlinear S-Boxes. Proceedings of EUROCRYPT '91, 378–386, 1991.
18. Nyberg, K., Knudsen, L. R.: Provable security against differential cryptanalysis. Proceedings of CRYPTO '92, 566–574, 1992.

A New Combinatorial Approach to Sequence Comparison

S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino

University of Palermo, Dipartimento di Matematica ed Applicationi,
Via Archirafi 34, 90123 Palermo, Italy
{sabrina, restivo, mari}@math.unipa.it, giovyros@virgilio.it

Abstract. In this paper we introduce a new alignment-free method for comparing sequences which is combinatorial by nature and does not use any compressor nor any information-theoretic notion. Such a method is based on an extension of the Burrows-Wheeler Transform, a transformation widely used in the context of Data Compression. The new extended transformation takes as input a multiset of sequences and produces as output a string obtained by a suitable rearrangement of the characters of all the input sequences. By using such a transformation we define a measure to compare sequences that takes into account how the characters coming from different input sequences are mixed in the output string. Such a method is tested on a real data set for the whole mitochondrial genome phylogeny problem. However, the goal of this paper is to introduce a new and general methodology for automatic categorization of sequences.

Introduction

The recent developments in sequencing genomes have given a new direction to bioinformatic research. Actually, the possibility of sequencing the whole genome has raised the question of discovering common features between biological sequences corresponding to different species, reflecting on common evolutionary and functional mechanisms. This reason has led researchers to look for a definition of a distance measure on sequences able to capture these common mechanisms. Most of the traditional methods for comparing biological sequences were based on the technique of sequence alignment. Nevertheless, sequence alignment considers only local mutations of the genome, therefore it is not suitable to measure events like segment rearrangements, that involve longer genomic sequences. For this reason some alignment-free distance measures have been recently introduced (see [25] for a survey). Most of them are based on the concept of information theory and data compression (cf. [23,13,4,6,1]). Such measures are more suitable to deal with the problem of whole genome phylogeny. The intuitive idea is that the more similar two sequences are, the more effective their joint compression is than their independent compression.

We introduce a new alignment-free method for comparing sequences that, differently from other ones, is combinatorial by nature and does not make use of any compressor nor any information theoretic notion. Our method is based on an extension of the Burrows-Wheeler Transform recently given in [18,19]. The Burrows-Wheeler Transform (denoted by BWT) is a well founded mathematical transformation on sequences introduced in 1994 (cf. [2]), widely used in the context of Data Compression ([20,7]) and recently studied also from a combinatorial point of view ([17]). It has been remarked (cf. [5]) that there exists a close relation between BWT and a technique, used by Gessel and Reutenauer in [8] for stating a correspondence between finite words and a set of permutations with a given cyclic structure and a given descent set (cf. also Chapter 11 of [14]).

Loosely speaking, BWT is a transformation that produces a permutation $BWT(w)$ of an input sequence w, such that we can easily retrieve w from $BWT(w)$, i.e. the transformation is reversible, and, at the same time, $BWT(w)$ is much easier to compress than w.

The new transformation introduced in [18,19] and denoted by E, works analogously to BWT, but takes as input a multiset S of sequences. Such a transformation has been also inspired by the above mentioned technique of Gessel and Reutenauer.

A fundamental step in the computation of $E(S)$ consists in sorting all the symbols occurring in the sequences in S, using as a sort key for each symbol its context, i.e. the segment following it in the sequence. Such a step is realized by sorting the conjugates of all sequences in S according to an order relation different from the lexicographical one.

We use the transformation E in order to define a new method for comparing sequences. Such a method is based on the following idea: when E is applied to $S = \{u, v\}$, if the same segment s occurs both in u and v, then the conjugates of u and v starting by s are likely to be close in the sorted list of conjugates. This implies that the greater is the number of segments shared by u and v, the greater is the mixing of the conjugates of u and v in the sorted list. The comparison method based on transformation E will measure how similar u and v are, by taking into account how their conjugates are mixed. This intuition has different possible formalizations. In [16] we introduced a distance measure that computes the number of alternations in the above list between the conjugates of u and those of v. In this paper we propose a new measure that takes into account also the characters in the output of the transformation.

The computation of our distance is simple and efficient, and it is particularly advantageous in the case of a multiple sequences comparison. We also test our method by applying the distance here introduced to a data set for the whole mitochondrial genome phylogeny problem. The results we have obtained are very close to the ones derived, with other approaches, in most of the papers in which the considered species are the same. Remark that, however, the goal of this paper is not to confirm or refute previous phylogenetic studies but rather to introduce a new methodology for automatic categorization of generic sequences of characters. Actually, such a methodology can also be applied to natural language processing, for instance in order to obtain languages phylogenies, authorship attributions, classifications of texts.

1 Preliminaries: the *BWT* and Its Extension

1.1 The Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT from now on) was introduced in 1994 by Burrows and Wheeler [2] and represents an extremely useful tool for textual lossless data compression. The idea is to apply a reversible transformation in order to produce a permutation $BWT(w)$ of an input sequence w, defined over an ordered alphabet A, so that the sequence becomes easier to compress. Actually the transformation leads to group characters together so that the probability of finding a character close to another instance of the same character is substantially increased. BWT transforms a sequence $w = a_0 a_1 \cdots a_{n-1}$ by constructing all n cyclic shifts of w, sorting them lexicographically and extracting the last character of each cyclic shift. The sequence $BWT(w)$ consists of the sequence of these characters. Moreover the transformation computes the index I, that is the row containing the original sequence in the sorted list of cyclic shifts.

For instance, suppose we want to compute $BWT(w)$ where $w = abraca$. Consider the matrix M in Figure 1, which consists of all cyclic shifts of w, lexicographically sorted.

$$\begin{array}{rrcccccc}
 & & F & & & & & L \\
 & & \downarrow & & & & & \downarrow \\
 & 0 & a & a & b & r & a & c \\
I \rightarrow & 1 & a & b & r & a & c & a \\
 & 2 & a & c & a & a & b & r \\
 & 3 & b & r & a & c & a & a \\
 & 4 & c & a & a & b & r & a \\
 & 5 & r & a & c & a & a & b
\end{array}$$

Fig. 1. The matrix M of all cyclic rotations of the sequence $w = abraca$

The last column L of the matrix represents $BWT(w) = caraab$ and $I = 1$ since the original sequence w appears in row 1. The first column F, instead, contains the sequence of the characters of w lexicographically sorted.

In [2] the following properties concerning BWT have been proved:

1. For all $i = 0, \ldots, n-1$, $i \neq I$, the character $L[i]$ is followed in the original string by $F[i]$;
2. for each character z, the i-th occurrence of z in F corresponds to the i-th occurrence of z in L.

From above properties it follows that the Burrows-Wheeler transform is reversible in the sense that, given $BWT(w)$ and the index I, it is possible to recover the original string w. Actually, according to Property 2, we can define a function $\tau\colon \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n-1\}$ giving the correspondence between the positions of characters of the first and the last column of the matrix M. The function τ represent also the order in which we have to rearrange the elements of F to reconstruct the original sequence w. Hence, starting from the position I, we can recover the sequence w as follows:

$$a_i = F[\tau^i(I)] \ , \text{ where } \tau^0(x) = x, \text{ and } \tau^{i+1}(x) = \tau(\tau^i(x)).$$

We show, for instance, how the reconstruction works for the example in Figure 1:

$$\tau = \begin{pmatrix} 0\,1\,2\,3\,4\,5 \\ 1\,3\,4\,5\,0\,2 \end{pmatrix},$$

$$\begin{array}{l} a_0 = F[1] = a \\ a_1 = F[3] = b \\ a_2 = F[5] = r \\ a_3 = F[2] = a \\ a_4 = F[4] = c \\ a_5 = F[0] = a. \end{array}$$

1.2 An Extension of the BWT to k Sequences

In [18,19] it was defined a new transformation that works analogously to the BWT, but it takes as input a multiset of k sequences, with $k \geq 1$. Such a transformation has also been inspired by a Gessel and Reutenauer result on Combinatorics of Permutations (cf. [8]). We need to introduce a new order relation between sequences that differs from the usual lexicographical order when one sequence is a prefix of the other one.

Let A be a finite ordered alphabet. We denote by A^* the set of sequences over A.

A sequence $v \in A^*$ is *primitive* if $v = w^n$ implies $v = w$ and $n = 1$. In this paper we take into account only primitive sequences. Note that this hypothesis is not restrictive, since any string can be transformed into a primitive string just by adding an end-of-string symbol. Recall that two sequences $x, y \in A^*$ are *conjugate* if x is a cyclic shift of y i.e. $x = uv$ and $y = vu$ for some $u, v \in A^*$. Note that a sequence v is primitive if and only if all its conjugates are distinct. If u is a sequence in A^*, we denote by u^ω the infinite sequence obtained by infinitely iterating u, i.e. $u^\omega = uuuuu\ldots$. Remark that if u and v are two distinct primitive sequences, then $u^\omega \neq v^\omega$. On infinite sequences, the lexicographic ordering is naturally defined , that is, given two infinite sequences $x = x_1x_2\ldots$ and $y = y_1y_2\ldots$, with $x_i, y_i \in A$, we say that $x <_{lex} y$ if either $x = y$, or there exists an index $j \in \mathbb{N}$ such that $x_i = y_i$ for $i = 1, 2, \ldots, j-1$ and $x_j < y_j$.

Let u, v be two primitive sequences. We say that

$$u \preceq_\omega v \Longleftrightarrow u^\omega <_{lex} v^\omega$$

It is easy to verify that $\preceq_\omega$ is a total order. We also remark that this order relation is different from the lexicographic one. In fact for instance $ab <_{lex} aba$ but $aba \preceq_\omega ab$. Although the $\preceq_\omega$ order of u and v is defined by using infinite sequences, the following proposition shows that it is possible to decide their mutual $\preceq_\omega$-ordering by extending them up to the length $|u| + |v| - \gcd(|u|, |v|)$.

Such a bound is a consequence of a well known result of Periodicity on Words, the Fine and Wilf theorem. We will denote by $pref_k(u)$ the prefix of length k of a finite or infinite sequence u. In [18,19] the following proposition is proved.

Proposition 1. *Given u and v two primitive sequences,*

$$u \preceq_\omega v \Longleftrightarrow pref_k(u^\omega) <_{lex} pref_k(v^\omega),$$

where $k = |u| + |v| - \gcd(|u|, |v|)$.

We can also remark that the bound given in Proposition 1 is tight. This is a consequence of the tightness of such a bound in the Fine and Wilf Theorem (cf. [22]).

Example 1. We can consider the sequences $u = abaab$ and $v = abaababa$. One can see that $v \preceq_\omega u$ and u^ω and v^ω differ for the character in position 12=5+8-1. However remark that $u <_{lex} v$.

$$\begin{array}{l}\overbrace{abaab}^{u}\overbrace{abaab}^{u}\overbrace{a\boldsymbol{b}\cdots}^{u}\\ \underbrace{abaababa}_{v}\underbrace{aba\boldsymbol{a}\cdots}_{v}\end{array}$$

Since we are interested in the $\preceq_\omega$-sorting of k sequences, it could be useful to have a common bound for all strings to be sorted. This can be derived from an extension of the Fine and Wilf Theorem to more than two periods (cf. [24,10]).

Let $S = \{u_1, \dots u_k\}$ be a multiset of k primitive sequences of A^*. We define the Extended Burrows-Wheeler Transform (denoted by E) as follows:

- Let $w_1, w_2, \dots, w_m$ be the list of conjugates of elements of S, sorted according to the order $\preceq_\omega$, that is $w_i \preceq_\omega w_j$ for $1 \le i < j \le m$.
- We denote by $\mathcal{I}$ the set of indices representing the positions in the list $\{w_i\}_{i=1}^m$ of the original sequences $u_1, \dots, u_k$ in S.
- We denote by $E(S)$ the word obtained by concatenating the last character of the each sequence w_i, for $i = 1, \dots, m$.
- The output of the transformation is the couple $(E(S), \mathcal{I})$,

If we arrange the sorted list of the conjugates of elements of S in a table, the sequence $E(S)$ is obtained by concatenating the last elements of each row in the table.

Example 2. Let $S = \{abac, cbab, bca, cba\}$. We represent below on the left side the $<_{lex}$-ordered list of all w^ω, where the w's are the conjugates of elements in S, and on the right side the final table with the $\preceq_\omega$-ordered rows:

Remark 1. Notice that in case of $k = 1$, that is $S = \{u\}$, one has that $E(S) = BWT(u)$. Moreover, remark that if $S = \{u_1, \dots, u_k\}$, then if $E(S)$ is an element of the shuffle of $BWT(u_1), BWT(u_2), \dots, BWT(u_k)$, since the conjugates of a single word in S are relatively sorted in the sorted list of all conjugates. For the same reason, if $S = X \cup Y$, $E(S)$ belongs to the shuffle of $E(X)$ and $E(Y)$.

$$\begin{array}{lcl}
a\,b\,a\,c\,a\,b\cdots & & 1\ a\,b\,a\,\mathbf{c} \\
a\,b\,c\,a\,b\,c\cdots & & 2\ a\,b\,\mathbf{c} \\
a\,b\,c\,b\,a\,b\cdots & & 3\ a\,b\,c\,\mathbf{b} \\
a\,c\,a\,b\,a\,c\cdots & & 4\ a\,c\,a\,\mathbf{b} \\
a\,c\,b\,a\,c\,b\cdots & & 5\ a\,c\,\mathbf{b} \\
b\,a\,b\,c\,b\,a\cdots & & 6\ b\,a\,b\,\mathbf{c} \\
b\,a\,c\,a\,b\,a\cdots & \Longrightarrow & 7\ b\,a\,c\,\mathbf{a} \\
b\,a\,c\,b\,a\,c\cdots & & 8\ b\,a\,\mathbf{c} \\
b\,c\,a\,b\,c\,a\cdots & & 9\ b\,c\,\mathbf{a} \\
b\,c\,b\,a\,b\,c\cdots & & 10\ b\,c\,b\,\mathbf{a} \\
c\,a\,b\,a\,c\,a\cdots & & 11\ c\,a\,b\,\mathbf{a} \\
c\,a\,b\,c\,a\,b\cdots & & 12\ c\,a\,\mathbf{b} \\
c\,b\,a\,b\,c\,b\cdots & & 13\ c\,b\,a\,\mathbf{b} \\
c\,b\,a\,c\,b\,a\cdots & & 14\ c\,b\,\mathbf{a}
\end{array}$$

Fig. 2. The output is the couple $(E(S), \mathcal{I})$ where $E(S) = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

Let us denote by M the table of the $\preceq_\omega$-sorted conjugates of the elements of the multiset S. Analogously to the case of BWT, in [16] the following properties have been proved:

1. For every $i \notin \mathcal{I}$, the first character of the i-th row of M, follows the last character of the same row in one of the sequences in S.
2. For any character, its occurrences in the first column of M appear in the same order as in $E(S)$.

As a consequence of the above properties, in [16] it has been proved that the transformation E is reversible in the sense that given an output $(E(S), \mathcal{I})$ of E, the original multiset S of primitive sequences over A can be recovered.

From an algorithmic point of view, as well as for the Burrows-Wheeler transformation (cf. [20,2]), the time complexity of the extended transformation depends on the sorting step. The problem of sorting the list of all conjugates can be reduced to the sorting of all suffixes of these sequences. This problem has been widely studied and it can be solved in linear time by building a suitable data structure (cf. [9,21,15,11]).

2 New Sequence Distance Measures

The transformation E can be used in order to define a class of distance measures between two sequences. Such measures are based on the remark that, when E is applied to a pair of sequences u, v, if the same segment s appears both in u and in v, then the conjugates of u and v starting by s are likely to be close in the $\preceq_\omega$-sorted list of conjugates. This implies that the greater is the number of segments shared by u and v, the greater is the mixing in the sorted list of the conjugates of u and v. A distance measure based on transformation E will take into such a mixing. This intuition has different possible formalizations. In this section we first recall a distance measure introduced in [16] that computes the number of alternations in the above list between the conjugates of u and those of

v. Then, we propose a new measure that takes into account also the characters in the output of the transformation.

Let $S = \{u, v\}$ and let $w_1, w_2, \ldots, w_m$ be the sorted list of the conjugates of u and v obtained in the first step of the computation of $E(u, v)$. Consider the new alphabet $\Sigma = \{U, V\}$: the *coloring* of $E(u, v)$ is the map $\gamma \colon \{1, 2, \cdots, m\} \to \{U, V\}$ defined as:

$$\gamma(i) = \begin{cases} U \text{ if } w_i \text{ is a conjugate of } u \\ V \text{ if } w_i \text{ is a conjugate of } v \end{cases}$$

If we say that U and V are the colors associated to u and v respectively, then γ associates to each conjugate of u the color U.

The distance measure δ introduced in [16] uses the map γ. We report here the definition:

Definition 1. *Let $u, v \in A^*$ be two sequences and let $\Gamma(u, v) = \gamma(1)\gamma(2)\cdots\gamma(m) = U^{n_1}V^{n_2}U^{n_3}\cdots V^{n_k}$, for some $n_1, n_2, \ldots, n_k \in \mathbb{N}$. Then*

$$\delta(u, v) = \sum_{\substack{i=1, \\ n_i \neq 0}}^{k} (n_i - 1)$$

In the Example 3 below, the computation of the distance δ on two words is shown. Notice that distance $\delta(u, v)$ considers the size of the blocks in $\Gamma(u, v)$ having the same color, independently from the corresponding letter in $E(u, v)$. That is, it takes into account the alternation of conjugates of u and v in the output of the extended transformation, but does not consider the characters in the same colored block. In [16] several properties of such a distance measure are proved. In particular, if u and v are conjugates, then $\delta(u, v) = 0$, but the converse is not always true, that is $\delta(u, v) = 0$ does not imply that u and v are conjugates. We note also that the triangle inequality does not hold for δ.

In this section we are going to define a different distance measure, that captures a different aspect of similarity between strings. Such a new distance takes into account at the same time the output of the transformation E and the coloring γ of its elements.

In order to introduce the new measure we need some notations.

Given a sequence $x = x_1x_2\cdots x_n \in A^*$, we denote by $x_{i,j}$, with $1 \leq i \leq j \leq n$, the subsequence of x starting at position i and ending at position j, that is $x_{i,j} = x_ix_{i+1}\cdots x_j$. A *monotonic block* of x is a subsequence $x_{i,j}$ constituted by equal characters and that is maximal with respect to this property, that is $x_{i,j} = a^{j-i+1}$ for some $a \in A$ and $x_{i-1}, x_{j+1} \neq a$. Notice that every word can be decomposed in a unique way as concatenation of monotonic blocks.

Let $E(u, v)$ be the output of E on the words u and v and let γ be its coloring. Let $B_1B_2\cdots B_k$ be the monotonic block decomposition of $E(u, v)$. Let us suppose that for some $h \in \{1, 2, \ldots, k\}$, $B_h = E(u, v)_{i,j}$. Then we define $c_h(u)$ the number of characters colored by U in the block B_h, that is

$$c_h(u) = card\{l \mid i \leq l \leq j \text{ and } \gamma(l) = U\}.$$

Analogously $c_h(v)$ is the number of characters colored by V in the block B_h, that is

$$c_h(v) = card\{l \mid i \leq l \leq j \text{ and } \gamma(l) = V\}.$$

The distance we are going to define takes into account how many symbols colored by U and V can be found inside each monotonic block in $E(u, v)$. Formally we define our distance measure as follows:

Definition 2. *Let u and v be two words over A, $E(u, v)$ the output of E, γ its coloring and $B_1 B_2 \cdots B_k$ its monotonic block decomposition. We define:*

$$\varrho(u, v) = \sum_{i=1}^{k} |c_i(u) - c_i(v)|.$$

Example 3. Consider the words $u = aaabbbb$ and $v = abaabbb$. Then

l	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$E(u,v)$	$\mathbf{b}$	$\mathbf{b}$	a	$\mathbf{b}$	a	a	$\mathbf{b}$	a	$\mathbf{b}$	$\mathbf{b}$	$\mathbf{b}$	$\mathbf{b}$	a	a
γ	$\mathbf{U}$	$\mathbf{V}$	U	$\mathbf{V}$	V	U	$\mathbf{U}$	V	$\mathbf{V}$	$\mathbf{U}$	$\mathbf{V}$	$\mathbf{U}$	V	U
$\lvert c_i(u) - c_i(v)\rvert$		0	1	1		0	1	1				0		0

Here we have distinguished the different monotonic blocks by writing in boldface the ones corresponding to the character b and leaving with the normal face the ones corresponding to character a. We write the value $|c_i(u) - c_i(v)|$ at the end of the corresponding i-th monotonic block. Then we can see that $\varrho(u, v) = \sum_{i=1}^{8} |c_i(u) - c_i(v)| = 4$. Regarding to distance δ, since $\Gamma(u, v) = UVUV^2U^2V^2UVUVU$ then $\delta(u, v) = 3$.

Remark 2. Notice that ϱ is a distance measure for conjugacy classes. Actually, if u' is a conjugate of u and v' is a conjugate of v, then $\varrho(u, v) = \varrho(u', v')$.

Proposition 2. *Let $u, v \in A^*$. Then $\varrho(u, v) = 0$ if and only if u is a conjugate of v.*

Proof. If u is a conjugate of v, then in the sorted list $w_1, w_2, \cdots w_m$, for all odd i, w_i and w_{i+1} are equal and they are conjugates of u and v respectively. That is, for each monotonic block B_h of $E(u, v)$ we have an even number of elements, alternatively colored U and V, that is $c_h(u) = c_h(v)$. Then $\varrho(u, v) = 0$.

Conversely, if $\varrho(u, v) = 0$, then for each monotonic block B_h of w, $c_h(u) = c_h(v)$. Since in each monotonic block we have the same number of characters coming from u and v and since $E(u, v)$ is an element of the shuffle of $BWT(u)$ and $BWT(v)$ (cf. Remark 1), we derive that $BWT(u) = BWT(v)$. So, we can deduce that u is a conjugate of v.

Nevertheless, we can not state that ϱ is a distance under the mathematical point of view. In fact ϱ does not satisfy the triangle inequality, as shown in the following example:

Example 4. Consider the words $u = aaabbbb$, $v = abaabbb$ and $z = abababb$. We have that $\varrho(u, z) = 4$, $\varrho(u, v) = 12$ and $\varrho(v, z) = 6$.

2.1 Multiple Sequence Comparison

A particular feature of our method is that it can also be applied in order to compare k different sequences, with $k > 2$. More formally, let $S = \{u_1, u_2, \ldots, u_k\}$ be a set of sequences and let $w_1, w_2, \ldots, w_m$ be the sorted list of the conjugates of the elements of S obtained in the first step of the computation of $E(S)$. Consider the new alphabet $\Sigma = \{U_1, U_2, \ldots, U_k\}$. The *coloring* of $E(S)$ is the map $\gamma \colon \{1, 2, \ldots, m\} \to \Sigma$ defined, for as:

$$\gamma(i) = U_j \qquad \text{if } w_i \text{ is a conjugate of } u_j$$

Notice that the coloring γ allows to recover the transformation of a subset of S. In fact, suppose that $X = \{x_1, x_2, \ldots, x_h\} \subset S$ and let $X_1, \ldots, X_h$ be the colors associated to $x_1, \ldots, x_h$, respectively. Then $E(X)$ is obtained from $E(S)$ by deleting the characters having colors not associated to elements of X.

Example 5. Let $S = \{u, v, z\}$, where $u = aaabbbb$, $v = abaabbb$ and $z = abababb$. Let U, V, Z be the colors associated, by the map γ, to u, v, z, respectively:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$E(u,v,z)$	b	b	a	b	b	b	b	a	a	b	a	b	b	a	a	b	b	a	b	a	a
γ	U	V	U	V	Z	Z	Z	V	U	U	V	V	Z	Z	Z	U	V	Z	U	V	U

It is easy to see how this output allows us to compute the transformation applied to all pairs of words in S. In fact $E(u, v) = bbabaabababbbbaa$, $E(u, z) = babbbabbaababa$ and $E(v, z) = bbbbbaabbaabaaa$. Moreover we can also easily compute $\varrho(u, z) = 4$, $\varrho(u, v) = 12$ and $\varrho(v, z) = 6$

The previous example shows how we can compute the distance ϱ of all pairs taken out of a set S of k sequences of length n by simultaneously applying the transformation E to the entire set S. Such a technique is very useful from a computational point of view, for instance in order to construct phylogenetic trees (see Section 3). Actually, in order to obtain the $k \times k$ distance matrix, we can compute $E(S)$ and its coloring γ by performing a single sorting of kn sequences of length n instead of $O(k^2)$ sortings of $2n$ sequences of length n.

From a theoretical point of view, such a method allows to define, in a natural way, a notion of distance between multisets of sequences. Actually, given two multisets S and T, we denote by $w_1, w_2, \ldots, w_m$ the sorted list of the conjugates of the elements of S and T obtained in the first step of the computation of $E(S \cup T)$. We can consider the new alphabet $\Sigma = \{U, V\}$ and the *coloring* map $\gamma \colon \{1, \ldots, m\} \to \Sigma$ defined as follows:

$$\gamma(i) = \begin{cases} U \text{ if } w_i \text{ is a conjugate of an element of } S \\ V \text{ if } w_i \text{ is a conjugate of an element of } T \end{cases}$$

Let $E(S \cup T)$ be the output of E, γ its coloring map and $B_1 B_2 \cdots B_p$ its monotonic block decomposition. We denote by $c_i(S)$ the number of characters in the monotonic block B_i colored by U. Then we can define

$$\varrho(S,T) = \sum_{i=1}^{p} |c_i(S) - c_i(T)|.$$

Such a measure provides a further tool for the classification of data.

3 Experiment on Biological Sequences

The distance introduced in this paper measures the dissimilarity between two conjugacy classes of sequences, i.e. two circular sequences (cf. Remark 2)[1]. So, in order to test our method, we applied a normalized version of our distance to the whole mitochondrial genome phylogeny, since a mitochondrial DNA sequence can be considered as a circular sequence. We used several groups of mtDNA genomes and the results we have obtained are very close to the ones derived, with other approaches, in most of the papers in which the same dataset is considered. Actually, in this section we report an experiment in which we construct a phylogeny of the Eutherian orders using complete unaligned mitochondrial genomes. We choose our group of sequences by using the mtDNA genomes of the following 20 mammals from *GenBank*: human (Homo sapiens, V00662), chimpanzee (Pan troglodytes, D38116), pigmy chimpanzee (Pan paniscus, D38113),

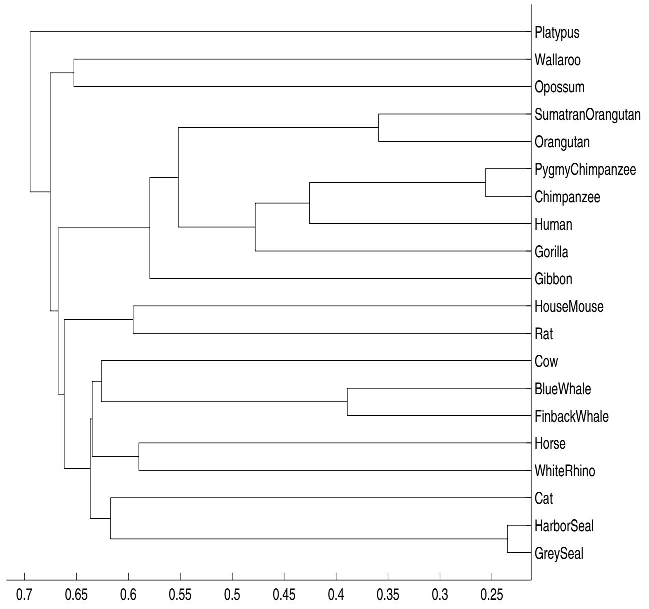


Fig. 3. The evolutionary tree built from complete mammalian mtDNA sequences of the 20 species analyzed in (Cao et al., 1998)

[1] Recall that in order to consider not circular sequences, it suffices to add an end-of-string symbol # to the sequences.

gorilla (Gorilla gorilla, D38114), orangutan (Pongo pygmaeus, D38115), gibbon (Hylobates lar, X99256), sumatran orangutan (Pongo pygmaeus abelii, X97707), horse (Equus caballus, X79547), white rhino (Ceratotherium simum, Y07726), harbor seal (Phoca vitulina, X63726), gray seal (Halichoerus grypus, X72004), cat (Felis catus, U20753), finback whale (Balenoptera physalus, X61145), blue whale (musculus, X72204), cow (Bos taurus, V00654), rat (Rattus norvegicus, X14848), house mouse (Mus musculus, V00711), opossum (Didelphis virginiana, Z29573), wallaroo (Macropus robustus, Y10524), and platypus (Ornithorhyncus anatinus, X83427). Note that the rodent species are kept to murids only and that marsupials and monotremes are also added. The dendrogram (see Figure 3) is generated by using a single linkage clustering.

The phylogeny here obtained is very close to the ones obtained in most of the papers in which the species considered are almost the same (cf. [23,3,12,13]). Our resulting phylogeny proposes the following grouping of the placental mammals: (Primates, (Ferungulates, Rodents)).

Nevertheless, the goal of this experiment is not to confirm or refute previous phylogenetic studies but rather to introduce new methods and tools to the comparative genomics research community. Actually, the phylogeny here obtained by using the proposed distance shows that our method can successfully construct evolutionary trees using whole genome sequences.

Acknowledgments

We wish to thank Cenk Sahinalp for his useful suggestions that led to the definition of the ϱ distance.

References

1. D. Benedetto, E. Caglioti, and V. Loreto. Zipping out relevant information. *Computing in Science and Engineering*, pages 80–85, 2003.
2. M. Burrows and D.J. Wheeler. A block sorting data compression algorithm. Technical report, DIGITAL System Research Center, 1994.
3. Y. Cao, A. Janke, P. J. Waddell, M. Westerman, O. Takenaka, S. Murata, N. Okada, S. Pääbo, and M. Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *J. Mol. Evol.*, 47:307–322, 1998.
4. R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Trans. Information Theory*, 51(4):1523–1545, April 2005.
5. M. Crochemore, J. Désarménien, and D. Perrin. A note on the Burrows-Wheeler transformation. *Theoret. Comput. Sci.*, 332:567–572, 2005.
6. F. Ergun, S. Muthukrishnan, and C. Sahinalp. Comparing sequences with segment rearrangements. *Lecture Notes in Comput. Sci*, pages 183–194, 2003. Proc. of the FSTTCS'03, Bombay, India.
7. P. Fenwick. The Burrows-Wheeler transform for block sorting text compression: principles and improvements. *The Computer Journal*, 39(9):731–740, 1996.
8. I. M. Gessel and C. Reutenauer. Counting permutations with given cycle structure and descent set. *J. Combin. Theory Ser. A*, 64(2):189–215, 1993.

9. D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.
10. L. Ilie and S. Constantinescu. Fine and Wilf's theorem for any number of periods. *TUCS (Turku Center for Computer Science) General Pubblication*, 25:65–74, 2003. proc. WORDS 2003.
11. N. J. Larsson and K. Sadakane. Faster suffix sorting. Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1-43/(1999), Department of Computer Science, Lund University, Sweden, 1999.
12. M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17:149–154, 2001.
13. M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi. The similarity metric. *IEEE Trans. Inform. Th.*, 12(5):3250–3264, 2004.
14. M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.
15. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
16. S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In Alberto Apostolico, Maxime Crochemore, and Kunsoo Park, editors, *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, volume 3537 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2005.
17. S. Mantaci, A. Restivo, and M. Sciortino. Burrows-Wheeler transform and Sturmian words. *Informat. Proc. Lett.*, 86:241–246, 2003.
18. S. Mantaci, A. Restivo, and M. Sciortino. An extension of the Burrows-Wheeler Transform to k words. Technical Report 267, University of Palermo, Dipartimento di Matematica ed Appl., December 2004.
19. S. Mantaci, A. Restivo, and M. Sciortino. An Extension of the Burrows Wheeler Transform to k Words (Extended Abstract). In *2005 Data Compression Conference (DCC 2005), 29-31 March 2005, Snowbird, UT, USA*, page 469. IEEE Computer Society, 2005.
20. G. Manzini. The Burrows-Wheeler transform: Theory and practice. In *Proc. of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS '99)*, pages 34–47. Springer-Verlag LNCS n. 1672, 1999.
21. E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.
22. F. Mignosi and A. Restivo. Periodicity. In *M. Lothaire, Algebraic Combinatorics on Words*, chapter 8, pages 237–274. Cambridge University Press, 2002.
23. H.H. Otu and K. Sayood. A new sequence distance measure for phylogenetic tree construction. *Bioinformatics*, 19(16):2122–2130, 2003.
24. R. Tijdeman and L.Zamboni. Fine and Wilf words for any periods. *Indag. Math.*, 14(1):135–147, 2003.
25. S. Vinga and J. Almeida. Alignment-free sequence comparison – a review. *Bioinformatics*, 19(4):513–523, 2003.

A Typed Assembly Language for Non-interference

Ricardo Medel[1], Adriana Compagnoni[1], and Eduardo Bonelli[2]

[1] Stevens Institute of Technology, Hoboken NJ 07030, USA
{rmedel, abc}@cs.stevens.edu
[2] LIFIA, Fac. de Informática, Univ. Nac. de La Plata, Argentina
eduardo@sol.info.unlp.edu.ar

Abstract. Non-interference is a desirable property of systems in a multilevel security architecture, stating that confidential information is not disclosed in public output. The challenge of studying information flow for assembly languages is that the control flow constructs that guide the analysis in high-level languages are not present. To address this problem, we define a typed assembly language that uses pseudo-instructions to impose a stack discipline on the control flow of programs. We develop a type system for checking that assembly programs enjoy non-interference and its proof of soundness.

1 Introduction

The confidentiality of information handled by computing systems is of paramount importance. However, standard perimeter security mechanisms such as access control or digital signatures fail to address the enforcement of information-flow policies. On the other hand, language-based strategies offer a promising approach to information flow security. In this paper, we study confidentiality for an assembly language using a language-based approach to security via type-theory.

In a multilevel security architecture information can range from having low (public) to high (confidential) security level. Information flow analysis studies whether an attacker can obtain information about the confidential data by observing the output of the system. The non-interference property states that any two executions of the same program, where only the high-level inputs differ in both executions, does not exhibit any observable difference in the program's output.

In this paper we define SIF, a typed assembly language for secure information flow analysis with security types. This language contains two pseudo-instructions, `cpush` L and `cjmp` L, for handling a stack of code labels indicating the program points where different branches of code converge, and the type system enforces a stack policy on those code labels. Our development culminates with a proof that well-typed SIF programs are assembled to untyped machine code that satisfy non-interference.

The type system of SIF detects explicit illegal flows as well as implicit illegal flows arising from the control structure of a program. Other covert channels such as those based on termination, timing, and power consumption, are outside the scope of this paper.

2 SIF, a Typed Assembly Language

In information flow analysis, a security level is associated with the program counter (pc) at each program execution point. This security level is used to detect implicit information flow from high-level values to low-level values. Moreover, control flow analysis is crucial in allowing this security level to decrease where there is no risk of illicit flow of information.

Sec. level of pc

```
low     if x=0
high      then y:=1
high      else y:=2
low     z:=3
```

(a) High-level program

L1 : `bnz` $r_1, L2$ % `if x≠0 goto` *L2*
     `move` $r_2 \leftarrow 1$ % `y:= 1`
     `jmp` $L3$
L2 : `move` $r_2 \leftarrow 2$ % `y:= 2`
L3 : `move` $r_3 \leftarrow 3$ % `z:= 3`

(b) Assembly program

Fig. 1. Example of implicit illegal information flow

Consider the example in Figure 1(a), where `x` has high security level and `z` has low security level. Notice that `y` cannot have low security level, since information about `x` can be retrieved from `y`, violating the non-interference property. Since the execution path depends on the value stored in the high-security variable `x`, entering the branches of the `if-then-else` changes the security level of the pc to high, indicating that only high-level variables can be updated. On the other hand, since `z` is modified after both branches, there is no leaking of information from either `y` or `x` to `z`. Therefore, the security level of the pc can be safely lowered.

A standard compilation of this example to assembly language may produce the code shown in Figure 1(b). Note that the block structure of the `if-then-else` is lost, and it is not clear where it is safe to lower the security level of the pc. We address this problem by including in our assembly language a stack of code labels accessed by two pseudo-instructions, `cpush` L and `cjmp` L, to simulate the block structure of high-level languages.

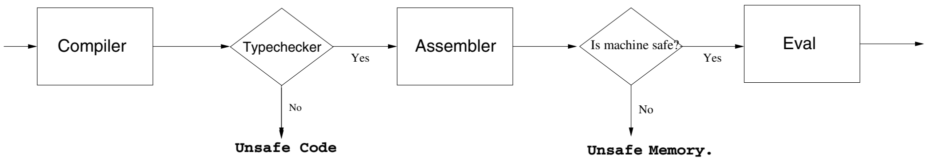
The instruction `cpush` L pushes L onto the stack while `cjmp` L first pops L from the stack if L is already at the top, and then jumps to the instruction labelled by L. The extra label information in `cjmp` L allows us to statically control that the intended label is removed, thereby preventing ill structured code.

The SIF code for the example in Figure 1(a) is shown below. The code at *L1* pushes the label *L3* onto the stack. The code at *L3* corresponds to the instructions following the `if-then-else` in the source code. Observe that the code at *L3* can only be executed once, because the instruction `cjmp` *L3* at the end of the code pointed to by *L1* (`then` branch), or at the end of *L2* (`else` branch), removes the top of the stack and jumps to the code pointed to by *L3*. At this point it is safe to lower the security level of the pc, since updating the low-security register r_3 does not leak any information about r_1.

Moreover, as in HBAL [1], the type-checking of the program is separated from the verification of the safety of the machine configuration where the program is assembled. Thus, following the schema shown below, a type-checker can verify if a program is safe

```
L1 : {r0 : int^⊥, r1 : int^⊤, r2 : int^⊤, r3 : int^⊥, pc : ⊥} || ϵ
     cpush L3                         % set junction point L3
     bnz r1,L2                        % if x≠ 0 goto L2
     arithi r2 ← r0 + 1               % y := 1, with r0=0
     cjmp L3
L2 : {r0 : int^⊥, r2 : int^⊤, r3 : int^⊥, pc : ⊤} || L3 · ϵ
     arithi r2 ← r0 + 2               % y := 2
     cjmp L3
L3 : {r0 : int^⊥, r3 : int^⊥, pc : ⊥} || ϵ
     arithi r3 ← r0 + 3               % z := 3
     halt
     eof
```

for execution on *any safe* memory configuration, and the runtime environment only needs to check that the initial machine configuration is safe before each run.



The assembler removes `cpush` L and translates `cjmp` L into `jmp` L, an ordinary unconditional jump, leaving no trace of these pseudo-instructions in the executable code (See the definition of the assembly function $\mathsf{Asm}(-)$ in section 2.4).

2.1 The Type System

We assume given a lattice $\mathfrak{L}_{\mathbf{sec}}$ of *security labels* [8], with an ordering relation $\sqsubseteq$, least ($\bot$) and greatest ($\top$) elements, and join ($\sqcup$) and meet ($\sqcap$) operations. These labels assign security levels to elements of the language through types. The type expressions of SIF are given by the following grammar:

security labels	$l \in \mathfrak{L}_{\mathbf{sec}}$
security types	$\sigma ::= \omega^l$
word types	$\omega ::= int \mid [\tau]$
memory location types	$\tau ::= \sigma \times \ldots \times \sigma \mid \mathsf{code}$

Security types (σ) are word types annotated with a security label. The expression $\mathrm{LABL}(\sigma)$ returns the security label of a security type σ. A *word type* (ω) is either an integer type (int) or a pointer to a memory location type ($[\tau]$). *Memory location types* (τ) are tuples of security types, or a special type code. We use $\tau[c]$, with c a positive integer, to refer to the c^{th} word type of the product type τ. Since the type code indicates the type of an assembly instruction, our system distinguishes code from data.

A *context* ($\Gamma \parallel \Lambda$) contains a register context Γ and a junction points stack Λ. A *junction points stack* (Λ) is a stack of code labels, each representing the convergence point of a fork in the control flow of a program. The empty stack is denoted by ϵ. A *register context* Γ contains type information about registers, mapping them to security types. We assume a finite set of registers $\{r_0, \ldots, r_n\}$, with two dedicated registers: r_0, that always holds zero, and pc, the program counter.

We write $Dom(\Gamma)$ for the domain of the register context Γ. The empty context is denoted by $\{\}$. The register context obtained by eliminating from Γ all pairs with r as first component be denoted by $\Gamma_{/r}$, while Γ, Γ' denotes the union of register contexts with disjoint domains. We use $\Gamma, r : \sigma$ as a shorthand for $\Gamma, \{r : \sigma\}$, and $\Gamma[r := \sigma]$ as a shorthand for $\Gamma_{/r}, \{r : \sigma\}$.

Since the program counter is always a pointer to code, we usually write $\mathsf{pc} : l$ instead of $\mathsf{pc} : [\mathtt{code}]^l$, and $\Gamma(\mathsf{pc}) = l$ if $\mathsf{pc} : l \in \Gamma$.

2.2 Syntax of SIF Programs

A *program* (P) is a sequence of instructions and code labels ended by the directive `eof`. SIF has standard assembly language instructions such as arithmetic operations, conditional branching, load, and store, plus pseudo-instructions `cpush` and `cjmp` to handle the stack of code labels.

$$
\begin{array}{lrl}
\textit{program} & P ::= & \mathtt{eof} \;\mid\; L; P \;\mid\; p; P \\
\textit{instructions} & p ::= & \mathtt{halt} \;\mid\; \mathtt{jmp}\ L \;\mid\; \mathtt{bnz}\ r, L \\
 & \mid & \mathtt{load}\ r \leftarrow r[c] \;\mid\; \mathtt{store}\ r[c] \leftarrow r \\
 & \mid & \mathtt{arith}\ r \leftarrow r \odot r \;\mid\; \mathtt{arithi}\ r \leftarrow r \odot i \\
 & \mid & \mathtt{cpush}\ L \;\mid\; \mathtt{cjmp}\ L \\
\textit{operations} & \odot ::= & + \;\mid\; - \;\mid\; * \;\mid\; /
\end{array}
$$

We use c to indicate an offset, and i to indicate integer literals. We assume an infinite enumerable set of code labels. Intuitively, the instruction `cpush` L pushes the junction point represented by the code label L onto the stack, while the instruction `cjmp` L behaves as a pop and a jump. If L is at the top of the stack, it pops L and then jumps to the instruction labeled L.

2.3 Typing Rules

A *signature* (Σ) is a mapping assigning contexts to labels. The context $\Sigma(L)$ contains the typing assumptions for the registers in the program point pointed to by the label L. The judgment $\Gamma \parallel \Lambda \vdash_\Sigma P$ is a typing judgment for a SIF program P, with signature Σ, in a context $\Gamma \parallel \Lambda$. We say that a program P is *well-typed* if $\mathsf{Ctxt}(P) \vdash_\Sigma P$, where $\mathsf{Ctxt}(P)$ is the partial function defined as: $\mathsf{Ctxt}(L; P) = \Sigma(L)$, $\mathsf{Ctxt}(\mathtt{eof}) = \{\} \parallel \epsilon$.

The typing rules for SIF programs, shown in Figures 2 and 3, are designed to prevent illegal flows of information. The directive `eof` is treated as a `halt` instruction. So, rules T_Eof and T_Halt ensure that the stack is empty.

Rule T_Label requires that the current context be compatible with the context expected at the position of the label, as defined in the signature (Σ) of the program. Jumps and conditional jumps are typed by rules T_Jmp and T_CondBrnch. In both rules the current context has to be compatible with the context expected at the destination code. In T_CondBrnch, both the code pointed to by L and the remaining program P are considered destinations of the jump included in this operation. In order to avoid implicit

$$\frac{\Gamma' \subseteq \Gamma \quad l \sqsubseteq l'}{(\Gamma, \mathsf{pc} : l \parallel \Lambda) \leq (\Gamma', \mathsf{pc} : l' \parallel \Lambda)}\ \mathsf{ST\_RegBank}$$

$$\frac{\mathsf{Ctxt}(P) \vdash_\Sigma P}{\Gamma \parallel \epsilon \vdash_\Sigma \mathtt{halt}\,; P}\ \mathsf{T\_Halt} \qquad \frac{}{\Gamma \parallel \epsilon \vdash_\Sigma \mathtt{eof}}\ \mathsf{T\_Eof}$$

$$\frac{(\Gamma \parallel \Lambda) \leq \Sigma(L) \quad \Sigma(L) \vdash_\Sigma P}{\Gamma \parallel \Lambda \vdash_\Sigma L; P}\ \mathsf{T\_Label}$$

$$\frac{(\Gamma \parallel \Lambda) \leq \Sigma(L) \quad \mathsf{Ctxt}(P) \vdash_\Sigma P}{\Gamma \parallel \Lambda \vdash_\Sigma \mathtt{jmp}\ L; P}\ \mathsf{T\_Jmp}$$

$$\frac{(\Gamma, r : \mathit{int}^l\ , \mathsf{pc} : l \sqcup l' \parallel \Lambda) \leq \Sigma(L) \quad \Gamma, r : \mathit{int}^l\ , \mathsf{pc} : l \sqcup l' \parallel \Lambda \vdash_\Sigma P}{\Gamma, r : \mathit{int}^l\ , \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \mathtt{bnz}\ r, L; P}\ \mathsf{T\_CondBrnch}$$

Fig. 2. Subtyping for contexts and typing rules for programs (1^{st} part)

flows of information, the security level of the pc in the destination code should not be lower than the current security level and the security level of the register (r) that controls the branching.

In T_Arith the security level of the source registers and the pc should not exceed the security level of the target register to avoid explicit flows of information. The security level of r_d can actually be lowered to reflect its new contents, but, to avoid implicit information flows, it cannot be lowered beyond the level of the pc. Similarly for T_Arithi, T_Load and T_Store. In T_Load, an additional condition establishes that the security level of the pointer to the heap has to be lower than or equal to the security level of the word to be read.

The rule T_Cpush controls whether `cpush` L can add the code label L to the stack. Since L is going to be consumed by a `cjmp` L instruction, its security level should not be lower than the current level of the pc. The `cjmp` L instruction jumps to the junction point pointed to by label L. Furthermore, to prevent ill structured programs the rule T_Cjmp forces the code label L to be at the top of the stack, and the current context has to be compatible with the one expected at the destination code. However, since a `cjmp` instruction allows the security level to be lowered, there are no conditions on its security level.

2.4 Type Soundness of SIF

In this section we define a semantics for the untyped assembly instructions operating on a machine model, we give an interpretation for SIF types which captures the way types are implemented in memory, and finally we prove that the execution of a well-typed SIF program modifies a type-safe configuration into another type-safe configuration.

Let $\mathsf{Reg} = \{0, 1, \ldots, \mathsf{R}_{\max}\}$ be the register indices, with two dedicated registers: $R(0) = 0$, and $R(\mathsf{pc})$ is the program counter. Let $\mathsf{Loc} \subseteq \mathbf{Z}$ be the set of memory lo-

$$\frac{\begin{array}{ll} \Gamma(r_d) = \omega^{l_d} & r_d, r_s, r_t \neq \mathsf{pc} \\ \Gamma(r_s) = int^{l_s} & l \sqcup l_s \sqcup l_t \sqsubseteq l_d \\ \Gamma(r_t) = int^{l_t} & \Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma P \end{array}}{\Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \texttt{arith}\ r_d \leftarrow r_s \odot r_t; P} \ \mathsf{T\_Arith}$$

$$\frac{\begin{array}{ll} & r_d, r_s \neq \mathsf{pc} \\ \Gamma(r_d) = \omega^{l_d} & l \sqcup l_s \sqsubseteq l_d \\ \Gamma(r_s) = int^{l_s} & \Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma P \end{array}}{\Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \texttt{arithi}\ r_d \leftarrow r_s \odot i; P} \ \mathsf{T\_Arithi}$$

$$\frac{\begin{array}{ll} \Gamma(r_s) = [\tau]^{l_s} & r_d, r_s \neq \mathsf{pc} \\ \Gamma(r_d) = \omega^{l_d} & l \sqcup l_s \sqsubseteq l_c \sqsubseteq l_d \\ \tau[c] = \omega_c^{l_c} & \Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma P \end{array}}{\Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \texttt{load}\ r_d \leftarrow r_s[c]; P} \ \mathsf{T\_Load}$$

$$\frac{\begin{array}{ll} \Gamma(r_d) = [\tau]^{l_d} & r_d, r_s \neq \mathsf{pc} \\ \Gamma(r_s) = \tau[c] = \omega^{l_s} & l \sqcup l_d \sqsubseteq l_s \\ \tau \text{ is } \mathsf{code\text{-}free} & \Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma P \end{array}}{\Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \texttt{store}\ r_d[c] \leftarrow r_s; P} \ \mathsf{T\_Store}$$

$$\frac{l \sqsubseteq \Sigma(L)(\mathsf{pc}) \quad \Gamma, \mathsf{pc} : l \parallel L \cdot \Lambda \vdash_\Sigma P}{\Gamma, \mathsf{pc} : l \parallel \Lambda \vdash_\Sigma \texttt{cpush}\ L; P} \ \mathsf{T\_Cpush}$$

$$\frac{\Sigma(L) = \Gamma' \parallel \Lambda \quad \Gamma'_{/\mathsf{pc}} \subseteq \Gamma_{/\mathsf{pc}} \quad \mathsf{Ctxt}(P) \vdash_\Sigma P}{\Gamma \parallel L \cdot \Lambda \vdash_\Sigma \texttt{cjmp}\ L; P} \ \mathsf{T\_Cjmp}$$

Fig. 3. Typing rules for programs (2^{nd} part)

cations on our machine, Wrd be the set of machine words that can stand for integers or locations, and Code be the set of machine words which can stand for machine instructions. To simplify the presentation, we assume that Wrd is disjoint from Code; so, our model keeps code separate from data.

A *machine configuration* M is a pair (H, R) where $H : \mathsf{Loc} \rightharpoonup \mathsf{Wrd} \uplus \mathsf{Code}$ is a partial function defining a heap configuration, and $R : \mathsf{Reg} \to \mathsf{Wrd}$ is a register configuration.

Given a program P, a *machine assembled for* P is a machine configuration which contains a representation of the assembly program, with machine instructions stored in some designated contiguous portion of the heap. Supposing $P = p_1; \ldots; p_n$, the assembly process defines a function $\mathsf{PAdr} : 1, \ldots, n \to \mathsf{Loc}$ which gives the destination location for the code when assembling the typed instruction p_u, where $1 \leq u \leq n$. For each of the locations ℓ where P is stored, $H(\ell) \in \mathsf{Code}$. The assembly process also defines the function $\mathsf{LAdr}(L)$, which assigns to each label in P the heap location where the code pointed to by the label was assembled.

Given a machine configuration $M = (H, R)$, we define a *machine transition* relation $M \longrightarrow M'$, as follows: First, M' differs from M by incrementing $R(\mathsf{pc})$ according to the length of the instruction in $H(R(\mathsf{pc}))$; then, the transformation given in the table below is applied to obtain the new heap H', or register bank R'. The operations on r_0 have no effect.

`jmp` L	$R' = R[\mathsf{pc} := \mathsf{LAdr}(L)]$
`bnz` r, L	$R' = \begin{cases} R, \text{ if } R(r) = 0 \\ R[\mathsf{pc} := \mathsf{LAdr}(L)], \text{ otherwise} \end{cases}$
`arith` $r_d \leftarrow r_s \odot r_t$	$R' = R[r_d := R(r_s) \odot R(r_t)]$
`arithi` $r_d \leftarrow r_s \odot i$	$R' = R[r_d := R(r_s) \odot i]$
`load` $r_d \leftarrow r_s[c]$	$R' = R[r_d := H(R(r_s) + c)]$
`store` $r_d[c] \leftarrow r_s$	$H' = H[R(r_d) + c := R(r_s)]$

$\mathsf{Asm}(p_u)$ stands for the sequence of untyped machine instructions which is the result of assembling a typed assembly instruction p_u:

$$\begin{array}{ll} \mathsf{Asm}(L) = \epsilon & \mathsf{Asm}(\texttt{eof}\ ) = \texttt{halt} \\ \mathsf{Asm}(\texttt{cpush}\ L) = \epsilon & \mathsf{Asm}(\texttt{cjmp}\ L) = \texttt{jmp}\ L \\ \mathsf{Asm}(p_u) = p_u, \text{ otherwise} & \end{array}$$

Notice that the sequence has at most one instruction. We write $M \overset{\mathsf{Asm}(p_u)}{\longrightarrow} M'$, if M executes to M' through the instructions in $\mathsf{Asm}(p_u)$, by zero or one transitions in M. The reflexive and transitive closure of this relation is defined by the following rules.

$$\frac{}{M \Longrightarrow M}\ \mathsf{Refl} \qquad \frac{M_1 \overset{\mathsf{Asm}(p_u)}{\longrightarrow} M_2}{M_1 \Longrightarrow M_2}\ \mathsf{Incl} \qquad \frac{M_1 \Longrightarrow M_2 \quad M_2 \Longrightarrow M_3}{M_1 \Longrightarrow M_3}\ \mathsf{Trans}$$

2.5 Imposing Types on the Model

A *heap context* ψ is a function that maps heap locations to security types. A heap context contains type information about the heap locations required to type the registers. $Dom(\psi)$ denotes the domain of the heap context ψ. The empty context is denoted by $\{\}$. We write $\psi[\ell := \tau]$ for the heap context resulting from updating ψ with $\ell : \tau$. Two heap contexts ψ and ψ' are *compatible*, denoted $\mathsf{compat}(\psi, \psi')$, if for all $\ell \in Dom(\psi) \cap Dom(\psi')$, $\psi(\ell) = \psi'(\ell)$. The following rules assign types to heap locations:

$$\frac{H(\ell) \in \mathsf{Code}}{H; \{\ell : \mathsf{code}\} \models \ell : \mathsf{code}\ \mathsf{hloc}}\ \mathsf{T\_HLocCode} \qquad \frac{H(\ell) \in \mathsf{Wrd}}{H; \{\ell : int^l\} \models \ell : int^l\ \mathsf{hloc}}\ \mathsf{T\_HLocInt}$$

$$\frac{H(\ell) \in \mathsf{Wrd} \quad \mathsf{compat}(\psi, \{\ell : [\tau]^l\}) \quad H; \psi \models H(\ell) : \tau\ \mathsf{hloc}}{H; \psi \cup \{\ell : [\tau]^l\} \models \ell : [\tau]^l\ \mathsf{hloc}}\ \mathsf{T\_HLocPtr}$$

$$\frac{\mathsf{compat}(\psi, \psi') \quad H; \psi \models \ell : \tau\ \mathsf{hloc}}{H; \psi \cup \psi' \models \ell : \tau\ \mathsf{hloc}}\ \mathsf{W\_HLoc}$$

$$\frac{\begin{array}{ll} m_i = size(\sigma_0) + \ldots + size(\sigma_{i-1}) & \\ H; \psi \models \ell + m_i : \sigma_i\ \mathsf{hloc} & \text{for all } 0 \leq i \leq n \end{array}}{H; \psi \models \ell : \sigma_0 \times \ldots \times \sigma_n\ \mathsf{hloc}}\ \mathsf{T\_HLocProd}$$

In order to define the notion of satisfiability of contexts by machine configurations, we need to define a satisfiability relation for registers.

$$\frac{r \neq \mathsf{pc}}{M \models_{\{\}} r : int^{l}\ \mathsf{reg}}\ \mathsf{T\_RegInt} \qquad \frac{H;\psi \models R(r) : \tau\ \mathsf{hloc}}{(H,R) \models_{\psi} r : [\tau]^{l}\ \mathsf{reg}}\ \mathsf{T\_RegPtr}$$

$$\frac{(H,R) \models_{\psi} r : \sigma\ \mathsf{reg} \quad \mathsf{compat}(\psi,\psi')}{(H,R) \models_{\psi \cup \psi}\ r : \sigma\ \mathsf{reg}}\ \mathsf{W\_Reg}$$

A machine configuration M *satisfies* a typing assignment Γ with a heap typing context ψ (written $M \models_{\psi} \Gamma$) if and only if for each register $r_i \in Dom(\Gamma)$, M satisfies the typing statement $M \models_{\psi_i} r_i : \Gamma(r_i)$ reg, the heap contexts ψ_i are pairwise compatible, and $\psi = \cup_{\forall i}\psi_i$.

A machine configuration $M = (H, R)$ is in *final state* if $H(R(\mathsf{pc})) = \mathtt{halt}$. We define an approximation to the execution of a typed program $P = p_1; \ldots; p_n$ by relating the execution of the code locations in the machine M with the control paths in the program by means of the relation $p_u \rightsquigarrow p_v$, which holds between pairs of instructions indexed by the set:

$$\begin{array}{l}\{(i, i+1) \mid p_i \neq \mathtt{jmp},\ \mathtt{cjmp},\ \text{ and } i < n\} \\ \cup \\ \{(i, j+1) \mid p_i = \mathtt{jmp}\ L, \mathtt{bnz}\ r, L, \text{ or } \mathtt{cjmp}\ L, \text{ and } p_j = L\}.\end{array}$$

We use $p_u \overset{*}{\rightsquigarrow} p_v$ to denote the reflexive and transitive closure of $p_u \rightsquigarrow p_v$.

2.6 Type Soundness

In this section we show that our type system ensures that the reduction rules preserve type safety. The soundness results imply that if the initial memory satisfies the initial typing assumptions of the program, then each memory configuration reachable from the initial memory satisfies the typing assumptions of its current instruction.

The typing assumptions of each instruction of a program can be obtained from the initial context by the typechecking process. The derivation $\mathsf{Ctxt}(P) \ \vdash_{\Sigma} \ P$ of a well-typed program $P = p_1; \ldots p_u; \ldots; p_n$ determines a sequence of contexts $\Gamma_1 \parallel \Lambda_1, \ldots, \Gamma_n \parallel \Lambda_n$ from sub-derivations of the form $\Gamma_u \parallel \Lambda_u \ \vdash_{\Sigma}\ p_u; p_{u+1}; \ldots; p_n$.

A machine configuration is considered type-safe if it satisfies the typing assumptions of its current instruction. Given a well-typed program $P = p_1; \ldots p_u; \ldots; p_n$ and a heap context ψ, we say $M = (H, R)$ is *type safe at* u *for* P *with* ψ if M is assembled for P; $R(\mathsf{pc}) = \mathsf{PAdr}(u)$; and $M \models_{\psi} \Gamma_u$.

We prove two meta-theoretic results Progress and Subject Reduction. Progress (Theorem 1) establishes that a non-final-state type safe machine can always progress to a new machine by executing a well-typed instruction, and Subject Reduction (Theorem 2) establishes that if a type safe machine progresses to another machine, the resulting machine is also type safe.

Theorem 1 (Progress). *Suppose a well-typed program* $P = p_1; \ldots p_u; \ldots; p_n$ *and a machine configuration* M *type safe at* u. *Then there exists* M' *such that* $M \xrightarrow{\mathsf{Asm}(p_u)} M'$, *or* M *is in* final state.

Theorem 2 (Subject Reduction). *Suppose $P = p_1; \ldots p_u; \ldots; p_n$ is a well-typed program and (H, R) is a machine configuration type safe at u, and $(H, R) \overset{\mathsf{Asm}(p_u)}{\longrightarrow} M'$. Then there exists $p_v \in P$ such that $p_u \rightsquigarrow p_v$ and M' is type safe at v.*

The proof of this theorem proceeds by case analysis on the current instruction p_u, analyzing each of the possible instructions that follow p_u, based on the definition of program transitions. See the companion technical report [13] for details.

3 Non-interference

Given an arbitrary (but fixed) security level ζ of an *observer*, non-interference states that computed low-security values ($\sqsubseteq \zeta$) should not be affected by high-security input values ($\not\sqsubseteq \zeta$). In order to prove that a program P satisfies non-interference one must show that any two terminating executions fired from indistinguishable (from the point of view of the observer) machine configurations yield indistinguishable configurations of the same security observation level.

In order to establish what it means for machine configurations to be indistinguishable from an observer's point of view whose security level is ζ, we define a ζ-indistinguishability relation for machine configurations.

The following definitions assume a given security level ζ, two machine configurations $M_1 = (H_1, R_1)$ and $M_2 = (H_2, R_2)$, two heap contexts ψ_1 and ψ_2, and two register contexts Γ_1 and Γ_2, such that $M_1 \models_{\psi_1} \Gamma_1$ and $M_2 \models_{\psi_2} \Gamma_2$.

Two register banks are ζ-indistinguishable if the observable registers in one bank are also observable in the other, and the contents of these registers are also ζ-indistinguishable.

Definition 1 (ζ-indistinguishability of register banks). *Two register banks R_1 and R_2 are ζ-indistinguishable, written $\rhd_{H_1:\psi_1, H_2:\psi_2} R_1 : \Gamma_1 \approx_\zeta R_2 : \Gamma_2$ $\mathsf{regBank}$, if for all $r \in Dom_{\cup}(\Gamma_1, \Gamma_2)$[1], with $r \neq \mathsf{pc}$:*

$$\textsc{Labl}(\Gamma_1(r)) \sqsubseteq \zeta \textit{ or } \textsc{Labl}(\Gamma_2(r)) \sqsubseteq \zeta \textit{ implies} \begin{cases} r \in Dom_{\cap}(R_1, R_2, \Gamma_1, \Gamma_2), \\ \Gamma_1(r) = \Gamma_2(r), \textit{ and} \\ \rhd_{H_1:\psi_1, H_2:\psi_2} R_1(r) \approx_\zeta R_2(r) : \Gamma_1(r)\ \mathsf{val} \end{cases}$$

Two word values v_1 and v_2 of type ω^l are considered ζ-indistinguishable, written $\rhd_{H_1:\psi_1, H_2:\psi_2} v_1 \approx_\zeta v_2 : \omega^l$ val, if $l \sqsubseteq \zeta$ implies that both values are equal. In case of pointers to heap locations, the locations have to be also ζ-indistinguishable.

Two heap values ℓ_1 and ℓ_2 of type τ are considered ζ-indistinguishable, written $\rhd_{H_1:\psi_1, H_2:\psi_2} \ell_1 \approx_\zeta \ell_2 : \tau$ hval, if $\ell_1 \in H_1$, $\ell_2 \in H_2$, and either the type τ is $\mathtt{code}$ and $\ell_1 = \ell_2$, or $\tau = \sigma_1 \times \ldots \times \sigma_n$ and each pair of offset locations $\ell_1 + m_i$ and $\ell_2 + m_i$ (with m_i as in rule T_HLocProd) are ζ-indistinguishable, or τ is a word type with a security label l and $l \sqsubseteq \zeta$ implies that both values are equal.

The proof of our main result, the Non-Interference Theorem 3, requires two notions of indistinguishability of stacks (Low and High). If one execution of a program branches on a condition while the other does not, the junction points stacks may differ in each of the paths followed by the executions. If the security level of the pc is low in one

[1] We use $Dom_{\oplus}(A_1, \ldots, A_n)$ as an abbreviation for $Dom(A_1) \oplus \ldots \oplus Dom(A_n)$.

$$\frac{}{\rhd_{\Sigma}\epsilon \approx_{\zeta} \epsilon\ \mathsf{Low}}\ \mathsf{LowAxiom} \qquad \frac{\Sigma(L)(\mathsf{pc}) \sqsubseteq \zeta \quad \rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{Low}}{\rhd_{\Sigma} L \cdot \Lambda_1 \approx_{\zeta} L \cdot \Lambda_2\ \mathsf{Low}}\ \mathsf{LowLow}$$

$$\frac{\Sigma(L_1)(\mathsf{pc}) \not\sqsubseteq \zeta \quad \Sigma(L_2)(\mathsf{pc}) \not\sqsubseteq \zeta \quad \rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{Low}}{\rhd_{\Sigma} L_1 \cdot \Lambda_1 \approx_{\zeta} L_2 \cdot \Lambda_2\ \mathsf{cstackLow}}\ \mathsf{LowHigh}$$

$$\frac{\rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{Low}}{\rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{High}}\ \mathsf{HighAxiom} \qquad \frac{\Sigma(L)(\mathsf{pc}) \not\sqsubseteq \zeta \quad \rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{High}}{\rhd_{\Sigma} L \cdot \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{High}}\ \mathsf{HighLeft}$$

$$\frac{\Sigma(L)(\mathsf{pc}) \not\sqsubseteq \zeta \quad \rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2\ \mathsf{High}}{\rhd_{\Sigma} \Lambda_1 \approx_{\zeta} L \cdot \Lambda_2\ \mathsf{High}}\ \mathsf{HighRight}$$

Fig. 4. ζ-indistinguishability of junction points stacks

execution, then it has to be low in the other execution as well, and the executions must be identical. The first three rules of Figure 4 define the relation of low-indistinguishability for stacks. In low-security executions the associated stacks mus be of the same size, and each code label in the stack of the first execution must be indistinguishable from that of the corresponding element in the second one.

If the security level of the pc of one of the two executions is high, then the other one must be high too. The executions are likely to be running different instructions, and thus the associated stacks may have different sizes. However, we need to ensure that both executions follow branches of the same condition. This is done by requiring that both associated stacks have a common (low-indistinguishable) sub-stack. The second three rules of Figure 4 define the relation of high-indistinguishability for stacks. Also note that, as imposed by the typing rules, the code labels added to the stack associated to high-security branches are of high-security level.

Finally, we define the relation of indistinguishability of two machine con from the point of view of an observer of level ζ.

Definition 2. *Two machine configurations $M_1 = (H_1, R_1)$ and $M_2 = (H_2, R_2)$ are* ζ-indistinguishable*, denoted by the judgment*

$$\rhd_P M_1 : \Gamma_1, \Lambda_1, \psi_1 \approx_{\zeta} M_2 : \Gamma_2, \Lambda_2, \psi_2\ \mathsf{mConfig},$$

if and only if

1. $M_1 \models_{\psi_1} \Gamma_1$ *and* $M_2 \models_{\psi_2} \Gamma_2$*,*
2. M_1 *and* M_2 *are assembled for* P *at the same addresses,*
3. $\rhd_{H_1:\psi_1, H_2:\psi_2} R_1 : \Gamma_1 \approx_{\zeta} R_2 : \Gamma_2$ regBank*, and*
4. *either*
   (a) $\Gamma_1(\mathsf{pc}) = \Gamma_2(\mathsf{pc}) \sqsubseteq \zeta$ *and* $R_1(\mathsf{pc}) = R_2(\mathsf{pc})$ *and* $\rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2$ Low*, or*
   (b) $\Gamma_1(\mathsf{pc}) \not\sqsubseteq \zeta$ *and* $\Gamma_2(\mathsf{pc}) \not\sqsubseteq \zeta$ *and* $\rhd_{\Sigma} \Lambda_1 \approx_{\zeta} \Lambda_2$ High.

Note that both machine configurations must be consistent with their corresponding typing assignments, and they must be executing the code resulting from assembling P.

We may now state the non-interference theorem establishing that starting from two indistinguishable machine configurations assembled for the same program P, if each execution terminates, the resulting machine configurations remain indistinguishable.

In the following theorem and lemmas, for any instruction p_i in a well-typed program $P = p_1; \ldots; p_n$, the context $\Gamma_i \parallel \Lambda_i$ is obtained from the judment $\Gamma_i \parallel \Lambda_i \vdash_\Sigma p_i; p_n$, which is derived by a sub-derivation of $\mathsf{Ctxt}(P) \vdash_\Sigma P$.

Theorem 3 (Non-interference). *Let $P = p_1; \ldots; p_n$ be a well-typed program, $M_1 = (H_1, R_1)$ and $M_2 = (H_2, R_2)$ be machine configurations such that both are* type safe *at 1 for P with ψ and*

$$\rhd_P M_1 : \Gamma_1, \epsilon, \psi \approx_\zeta M_2 : \Gamma_1, \epsilon, \psi \; \mathsf{mConfig}.$$

If $M_1 \Longrightarrow M_1'$ and $M_2 \Longrightarrow M_2'$, with M_1' and M_2' in final state, *then*

$$\rhd_P M_1' : \Gamma_v, \epsilon, \psi_1 \approx_\zeta M_2' : \Gamma_w, \epsilon, \psi_2 \; \mathsf{mConfig}.$$

The technical challenge that lies in the proof of this theorem is that the ζ-indistinguishability of configurations holds after each transition step. The proof is developed in two stages. First it is proved that two ζ-indistinguishable configurations that have a low (and identical) level for the pc can reduce in a *lock step fashion* in a manner invariant to the ζ-indistinguishability property. This is stated by the following lemma.

Lemma 1 (Low-PC Step). *Let $P = p_1; \ldots; p_n$ be a well-typed program, such that p_{v_1} and p_{v_2} are in P, $M_1 = (H_1, R_1)$ and $M_2 = (H_2, R_2)$ be machine configurations. Suppose*

1. *M_1 is type safe at v_1 and M_2 is type safe at v_2, for P with ψ_1 and ψ_2, respectively,*
2. $\rhd_P M_1 : \Gamma_{v_1}, \Lambda_{v_1}, \psi_1 \approx_\zeta M_2 : \Gamma_{v_2}, \Lambda_{v_2}, \psi_2 \; \mathsf{mConfig}$,
3. *$\Gamma_{v_1}(\mathsf{pc}) \sqsubseteq \zeta$ and $\Gamma_{v_2}(\mathsf{pc}) \sqsubseteq \zeta$,*
4. *$M_1 \xrightarrow{\mathsf{Asm}(p_{v_1})} M_1'$, and*
5. *there exists p_{w_1} in P such that $p_{v_1} \rightsquigarrow p_{w_1}$, and M_1' is type safe at w_1 with ψ_3.*

Then, there exists a configuration M_2' such that:

(a) $M_2 \xrightarrow{\mathsf{Asm}(p_{v_2})} M_2'$,
(b) *there exists p_{w_2} in P such that $p_{v_2} \rightsquigarrow p_{w_2}$, and M_2' is type safe at w_2 with ψ_4, and*
(c) $\rhd_P M_1' : \Gamma_{w_1}, \Lambda_{w_2}, \psi_3 \approx_\zeta M_2' : \Gamma_{w_2}, \Lambda_{w_2}, \psi_4 \; \mathsf{mConfig}$.

When the level of the pc is low, the programs execute the same instructions (with possibly different heap and register bank). They may be seen to be *synchronized* and each reduction step made by one is emulated with a reduction of the same instruction by the other. The resulting machines must be ζ-indistinguishable.

However, a conditional branch (`bnz`) may cause the execution to fork on a high value. As a consequence, both of their pc become high and we must provide proof that there are some ζ-indistinguishable machines to which they reduce. Then, the second stage of the proof consists of showing that every reduction step of one execution whose pc has a high-security level can be met with a number of reduction steps (possibly none) from the other execution such that they reach indistinguishable configurations. The High-PC Step Lemma states such result.

Lemma 2 (High-PC Step). *Let $P = p_1; \ldots; p_n$ be a well-typed program, such that p_{v_1} and p_{v_2} are in P, and $M_1 = (H_1, R_1)$ and $M_2 = (H_2, R_2)$ be machine configurations. Suppose*

1. *M_1 is type safe at v_1 and M_2 is type safe at v_2, for P with ψ_1 and ψ_2, respectively.*
2. *$\rhd_P M_1 : \Gamma_{v_1}, \Lambda_{v_1}, \psi_1 \approx_\zeta M_2 : \Gamma_{v_2}, \Lambda_{v_2}, \psi_2$ $\mathsf{mConfig}$,*
3. *$\Gamma_{v_1}(\mathsf{pc}) \not\sqsubseteq \zeta$ and $\Gamma_{v_2}(\mathsf{pc}) \not\sqsubseteq \zeta$,*
4. *$M_1 \overset{\mathsf{Asm}(p_{v_1})}{\longrightarrow} M_1'$, and*
5. *there exists p_{w_1} in P such that $p_{v_1} \rightsquigarrow p_{w_1}$ and M_1' is type safe at w_1 with ψ_3.*

Then, either the configuration M_2 diverges or there exists a machine configuration M_2' such that

(a) *$M_2 \Longrightarrow M_2'$,*
(b) *there exists p_{w_2} in P such that $p_{v_2} \overset{*}{\rightsquigarrow} p_{w_2}$ and M_2' is type safe at w_2 with ψ_4, and*
(c) *$\rhd_P M_1' : \Gamma_{w_1}, \Lambda_{w_1}, \psi_3 \approx_\zeta M_2' : \Gamma_{w_2}, \Lambda_{w_2}, \psi_4$ $\mathsf{mConfig}$.*

The main technical difficulty here is the proof of the case when one execution does a `cjmp` instruction that lowers the pc level. In this case, the other execution should, in a number of steps, also reduce its pc level accordingly. This is guaranteed by two facts. First, high-indistinguishable stacks share a sub-stack whose top is the label to the junction point where the pc level is reduced and both executions converge. Second, well-typed programs reach final states only with an empty stack, having visited all the labels indicated by the junction point stack.

4 Related Work

Information flow analysis has been an active research area in the past three decades [18]. Pioneering work by Bell and LaPadula [4], Feiertag et al. [9], Denning and Denning [8,7], Neumann et al. [17], and Biba [5] set the basis of multilevel security by defining a model of information flow where subjects and objects have a security level from a lattice of security levels. Such a lattice is instrumental in representing a security policy where a subject cannot read objects of level higher than its level, and it cannot write objects at levels lower than its own level.

The notion of *non-interference* was first introduced by Goguen and Meseguer [10], and there has been a significant amount of research on type systems for confidentiality for high-level languages including Volpano and Smith [20], and Banerjee and Naumann [2]. Type systems for low-level languages have been an active subject of study for several years now, including TAL [14], STAL [15], DTAL [21], Alias Types [19], and HBAL [1].

In his PhD thesis [16], Necula already suggests information flow analysis as an open research area at the assembly language level. Zdancewic and Myers [22] present a low-level, secure calculus with ordered linear continuations. An earlier version of our type system was inspired by that work. However, we discovered that in a typed assembly language it is enough to have a junction point stack instead of mimicking ordered linear continuations. Moreover, their language has an `if-then-else` constructor that guides the information flow analysis, while SIF has pseudo-instructions (`cpush` L and `cjmp` L) for the same purpose. However, while the `if-then-else` constructor remains part of their language after typechecking, `cpush` and `cjmp` are eliminated.

Barthe et al. [3] define a JVM-like low-level language with a heap and an operand stack. The type system is parameterized by control dependence regions, and it is assumed that there exist functions that obtain such regions. In contrast, SIF allows such regions to be expressed in the language by using code labels and its well-formedness to be verified during type-checking. Crary et al. [6] define a low-level calculus for information flow analysis, however, their calculus has the structuring construct `if-then-else`, unlike SIF that uses typed pseudo-instructions that are assembled to standard machine instructions.

5 Conclusions and Future Work

We defined SIF, a typed assembly language for secure information flow analysis. Besides the standard features, such as heap and register bank, SIF introduces a stack of code labels in order to simulate at the assembly level the block structure of high-level languages. The type system guarantees that well-typed programs assembled on type-safe machine configurations satisfy the non-interference property: for a security level ζ, if two type-safe machine configuration are ζ-indistinguishable, then the resulting machine configurations after execution are also ζ-indistinguishable.

An alternative to our approach is to have a list of the program points where the security level of the pc can be lowered safely. This option delegates the security analysis of where the pc level can be safely lowered to a previous step (that may use, for example, a function to calculate control dependence regions [12]). This delegation introduces a new trusted structure into the type system. Our type system, however, does not need to trust the well-formation of such a list. Moreover, even the signature (Σ) attached to SIF programs is untrusted in our setting, since, as we explained in section 2.3, its information about the security level of the pc is checked in the rules for `cpush` and `cjmp` in order to prevent illegal information flows.

Currently we are implementing the type system proposed in this paper. We already developed a compiling function from a very simple high-level imperative programming language to SIF and the typechecker for SIF programs. We intend to make the software available upon completion of the system.

We are also developing a version of our language that includes a runtime stack, in order to define a stack-based compilation function from a more complex high-level language to SIF.

Acknowledgments. We are grateful to Pablo Garralda, Healfdene Goguen, David Naumann, and Alejandro Russo for enlightening discussions and comments on previous drafts. We want to thank Nicholas Egebo and Haifan Lu, two undergraduate students at Stevens, for their help with the implementation of SIF. This work was partially supported by the *NSF* project *CAREER: A formally verified environment for the production of secure software* – #0093362 and the Stevens Technogenesis Fund.

References

1. David Aspinall and Adriana B. Compagnoni. Heap bounded assembly language. *Journal of Automated Reasoning, Special Issue on Proof-Carrying Code*, 31(3-4):261–302, 2003.
2. A. Banerjee and D. Naumann. Secure information flow and pointer confinement in a java-like language. In *Proceedings of Fifteenth IEEE Computer Security Foundations - CSFW*, pages 253–267, June 2002.
3. G. Barthe, A. Basu, and T. Rezk. Security types preserving compilation. In *Proceedings of VMCAI'04*, volume 2937 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
4. D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report Technical Report MTR 2547 v2, MITRE, November 1973.
5. K. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977.
6. Karl Crary, Aleksey Kliger, and Frank Pfenning. A monadic analysis of information flow security with mutable state. Technical Report CMU-CS-03-164, Carnegie Mellon University, September 2003.
7. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, July 1977.
8. Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, May 1976.
9. R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *6th ACM Symp. Operating System Principles*, pages 57–65, November 1977.
10. J. A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20. IEEE Press, 1982.
11. Daniel Hedin and David Sands. Timing aware information flow security for a javacard-like bytecode. In *Proceedings of BYTECODE, ETAPS'05, to appear*, 2005.
12. Xavier Leroy. Java bytecode verification: an overview. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV'01*, volume 2102, pages 265–285. Springer-Verlag, 2001.
13. Ricardo Medel, Adriana Compagnoni, and Eduardo Bonelli. A typed assembly language for secure information flow analysis. `http://www.cs.stevens.edu/~rmedel/hbal/publications/sifTechReport.ps`, 2005.
14. G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to Typed Assembly Language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, May 1999.
15. Greg Morrisett, Karl Crary, Neal Glew, and David Walker. Stack-based typed assembly language. In *Second International Workshop on Types in Compilation*, pages 95–117, Kyoto, March 1998. Published in Xavier Leroy and Atsushi Ohori, editors, *Lecture Notes in Computer Science*, volume 1473, pages 28-52. Springer-Verlag, 1998.
16. George Necula. *Compiling with Proofs*. PhD thesis, Carnegie Mellon University, September 1998.
17. Peter G. Neumman, Richard J. Feiertag, Karl N. Levitt, and Lawrence Robinson. Software development and proofs of multi-level security. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 421–428. IEEE Computer Society, October 1976.
18. A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.
19. Frederick Smith, David Walker, and Greg Morrisett. Alias types. In Gert Smolka, editor, *Ninth European Symposium on Programming*, volume 1782 of *LNCS*, pages 366–381. Springer-Verlag, April 2000.

20. Dennis M. Volpano and Geoffrey Smith. A type-based approach to program security. In *TAPSOFT*, pages 607–621, 1997.
21. Hongwei Xi and Robert Harper. A dependently typed assembly language. Technical Report OGI-CSE-99-008, Oregon Graduate Institute of Science and Technology, July 1999.
22. S. Zdancewic and A. Myers. Secure information flow via linear continuations. *Higher Order and Symbolic Computation*, 15(2–3), 2002.

Improved Exact Exponential Algorithms for Vertex Bipartization and Other Problems

Venkatesh Raman, Saket Saurabh, and Somnath Sikdar

The Institute of Mathematical Sciences, Chennai, India, 600113
{vraman, saket, somnath}@imsc.res.in

Abstract. We develop efficient exact algorithms for several NP-hard problems including VERTEX BIPARTIZATION, FEEDBACK VERTEX SET, 4-HITTING SET, and MAX CUT in graphs with maximum degree at most 4. Our main results include:

- an $O^*(1.9526^n)$ [1] algorithm for VERTEX BIPARTIZATION problem in undirected graphs;
- an $O^*(1.8384^n)$ algorithm for VERTEX BIPARTIZATION problem in undirected graphs of maximum degree 3;
- an $O^*(1.945^n)$ algorithm for FEEDBACK VERTEX SET and VERTEX BIPARTIZATION problem in undirected graphs of maximum degree 4;
- an $O^*(1.9799^n)$ algorithm for 4-HITTING SET problem;
- an $O^*(1.5541^m)$ algorithm for FEEDBACK ARC SET problem in tournaments.

To the best of our knowledge, these are the best known exact algorithms for these problems. In fact, these are the first exact algorithms for these problems with the base of the exponent < 2. En route to these algorithms, we introduce two general techniques for obtaining exact algorithms. One is through parameterized complexity algorithms, and the other is a 'colored' branch-and-bound technique.

1 Introduction

In recent years there has been a growing interest in designing exact algorithms for NP-hard problems. Fast exponential-time algorithms lead to practical algorithms for at least moderate instance sizes. Furthermore, there is a wide variation in the time complexities of exact algorithms for NP-complete problems. Classical complexity theory cannot explain these differences. The study of exact algorithms may lead to a finer classification, and hopefully a better understanding, of NP-complete problems. For a recent survey on exact algorithms by Woeginger, see [16].

Parameterized complexity is a recently developed approach devised by Downey and Fellows for dealing with hard computational problems arising from industry and applications. The theory of parameterized complexity is based on

[1] The O^* notation suppresses polynomial terms. Thus we write $O^*(T(x))$ for a time complexity of the form $O(T(x) \cdot \text{poly}(|x|))$ where $T(x)$ grows exponentially with $|x|$, the input size. See the survey by Woeginger[16] for a detailed discussion on this.

the observation that many hard problems are associated with a parameter that varies within a small or moderate range. By taking advantage of small parameter values many hard problems can be solved practically. A parameterized problem consists of a tuple (π, k) where π is the problem instance and k is the parameter. A parameterized problem is said to be *fixed parameter tractable* if there exists an algorithm for the problem with time complexity $O(f(k) \cdot |\pi|^{O(1)})$, where f is a function of k alone and $|\pi|$ represents the size of the input instance. For an introduction to parameterized complexity see the book by Downey and Fellows [3]. For recent developments see the survey by Downey and Fellows [4].

The Vertex Bipartization problem is to find, given an undirected graph G on n vertices, the minimum number of vertices whose removal makes the graph bipartite. This problem has numerous applications, for instance, in VLSI design [2], computational biology [13], and register allocation [17]. The Vertex Bipartization problem is known to be NP-Complete even for graphs with maximum degree 3 [2]. This problem has been studied extensively from different algorithmic paradigms. An approximation algorithm with factor $\log n$ is known for the problem [5]. The parameterized version of this problem has been recently shown to be fixed parameter tractable by Reed et al [12]. Their algorithm runs in time $O(3^k \cdot kmn)$, where k is the parameter, n is the number of vertices and m is the number of edges. The (optimization) problem can be solved exactly by exhaustively looking at all possible subsets of vertices in time $O^*(2^n)$. So far there has been no exact algorithm better than this trivial brute-force algorithm.

In this paper we describe a generic technique that allows us to construct efficient exact algorithms for a problem using a parameterized algorithm for it. Let Q be an NP-optimization problem such that for every instance I of Q there is a polynomial time computable universe U of size, say n, such that an optimum solution of I is a subset of U. All the optimization problems considered in this paper satisfy this condition. The parameterized version of an NP-optimization problem Q consists of pairs of the form $\langle \pi, k \rangle$, where π is an instance of Q and k is a positive integer. Here the question is: *does Q have a solution of size k ?* We show that if the parameterized version of Q has a fixed parameter algorithm with time complexity $O^*(c^k)$ then its optimization version has an exact algorithm with time complexity $O^*(d^n)$, where $d < c$, and $n = |U|$. Using this technique we obtain an exact algorithm for the Vertex Bipartization problem that runs in time $O^*(1.9526^n)$, where n is the number of vertices. We devise another technique which is a modified form of the branch-and-bound method and obtain an exact algorithm with time complexity $O^*(1.8384^n)$ for the Vertex Bipartization problem in maximum degree 3 graphs. We extend this technique to obtain exact algorithms for the Feedback Vertex Set and Vertex Bipartization problems in maximum degree 4 graphs. The Feedback Vertex Set problem is to find the minimum number of vertices whose deletion makes the given graph acyclic. The Feedback Vertex Set problem is polynomial time solvable in maximum degree 3 graphs [14], but is NP-complete for maximum degree 4 graphs.

The paper is organized as follows. In Section 2, we present exact algorithms for the Vertex Bipartization problem in graphs with maximum degrees 3 and 4, and the Feedback Vertex Set problem in graphs with maximum degree 4. These algorithms use a 'colored' branch and bound technique. In Section 3, we develop a general technique by which we can convert a parameterized algorithm of time complexity $O^*((4-\epsilon)^k),\ \epsilon > 0$ to an exact algorithm of time complexity $O^*((2-\eta)^n),\ \eta > 0$ where $n = |U|$. In Section 3.1, we give several applications of this result. In particular, we give the best known exact algorithms for

1. Vertex Bipartization problem in general undirected graphs;
2. 4-Hitting Set problem; and
3. Feedback Arc Set problem in tournaments.

Furthermore, we give simple efficient exact algorithms for the Max Cut problem in graphs with average vertex degree 3 and 4 and for the 3-Hitting Set problem; these are not the best known exact algorithms for these problems but are stated in this paper to highlight the applicability of our technique. In Section 4, we conclude with some remarks and open problems. All graphs in this paper are undirected with n vertices and m edges unless stated otherwise.

2 Exact Algorithms for Vertex Bipartization and Feedback Vertex Set in Graphs with Maximum Degree 4

In this section, we give improved exact algorithms for the Vertex Bipartization problem in graphs of maximum degrees 3 and 4, and the Feedback Vertex Set problem in graphs of maximum degree 4.

The main idea behind these algorithms is to use the techniques of *preprocessing* and *branching.* Typical branch-and-bound algorithms (for Independent Set, Vertex Cover) build a solution by either picking a vertex or excluding it from the solution. When they exclude a vertex from the solution they typically delete it and work on the resulting smaller graph. For the problems we work on, when we exclude a vertex from the solution we cannot delete it, because we need to identify and cover cycles passing through it.

To overcome this, we resort to coloring the vertices. All vertices are colored good initially. When we branch on a good vertex, we either include it in our solution and delete it, or exclude it and color it bad. Coloring a vertex bad *decreases the number of good vertices.* As we always branch on good vertices we end up reducing the graph size in both cases. Just doing this gives us an $O^*(2^n)$ algorithm. Our main contribution is in pushing this idea to get an $O^*(c^n)$ algorithm where $c < 2$. In all the algorithms to follow Col is a function from the vertex set of the input graph to the set $\{\mathsf{good}, \mathsf{bad}\}$.

2.1 Vertex Bipartization in Graphs with Maximum Degree 3

The Vertex Bipartization problem is to find a minimum set of vertices whose removal makes the graph bipartite. The algorithm first preprocesses the graph using a preprocessing algorithm P (see Figure 2) and then either does a brute-force enumeration or finds a path xyz of length two consisting only of good vertices and branches on z. The algorithm is depicted in Figure 1.

```
Algorithm VBP-D3($G = (V, E)$, $S$, Col)
Input: An undirected multigraph $G = (V, E)$ with maximum degree 3, whose vertices
have been colored. Here $n$ is the number of vertices in the input graph. Initially the
algorithm is called with $S \leftarrow \emptyset$ and $\mathsf{Col}(v) = \mathsf{good}$ for all vertices $v \in G(V)$.
Output: A minimum vertex bipartization set of $G$.

Step 1 Call $P(G, S, \mathsf{Col})$. If $P$ returns NO then return NO.
Step 2 Apply the first step which is applicable:
    Step 2a Let $n'$ be the number of good vertices in the current graph. If $n' \leq 0.6n$
        or if every path of length 2 has at least one bad vertex then try all possible
        solutions $S \cup T$, where $T$ is a some subset of the remaining good vertices, and
        return the one with minimum size.
    Step 2b Pick a path $xyz$ in $G$ where all of $x, y,$ and $z$ are good. Call the algorithm
        on the following instances and return the smaller solution.
        – $S \leftarrow S \cup \{z\}$ and call VBP-D3($G - \{z\}$, $S$, Col).
        – Set $\mathsf{Col}(z) = \mathsf{bad}$ and call VBP-D3($G$, $S$, Col).
```

Fig. 1. Algorithm VBP-D3

Correctness. Step 2a of the algorithm VBP-D3 simply does a brute-force enumeration. In Step 2b, the algorithm branches on a good vertex z and constructs two solutions: one containing z and one not containing z and returns the one with minimum size. Both these steps do not need any further justification. We only need to justify the steps of the preprocessing algorithm P.

In Step 1 of the preprocessing algorithm, we recursively remove vertices of degree ≤ 1. Such vertices cannot be part of any minimum solution and can be safely removed. In Step 2, we check whether the subgraph $G[B]$ induced by the bad vertices contains an odd cycle. If this is the case, then there cannot be a minimum solution containing good vertices only. Thus P returns NO. The correctness of Step 3 is obvious.

We next consider Step 4 in detail. In this step, we look for a vertex v_i of degree 2 which does not have a self-loop. Such a vertex must be part of some path $uv_1 \ldots v_i \ldots v_k w$, where u and w are of degree 3 and are possibly identical. There are two broad cases to handle:

Case 1: At least one of u or w is colored good. Without loss of generality assume that $\mathsf{Col}(u) = \mathsf{good}$. Every odd cycle that passes through v_i also passes through u and w. Thus if there is a good vertex v_j that is part of some minimum solution S, then $(S \setminus \{v_j\}) \cup \{u\}$ is also a minimum solution. Therefore we can label all the vertices v_j bad, $(1 \leq j \leq k)$. We would also like to maintain the parity of all cycles passing through u (and w). Thus if the path length is even we retain only one of the vertices v_j and add the edges (u, v_j) and (v_j, w); if the path length is odd we delete all the vertices v and add an edge between u and w.

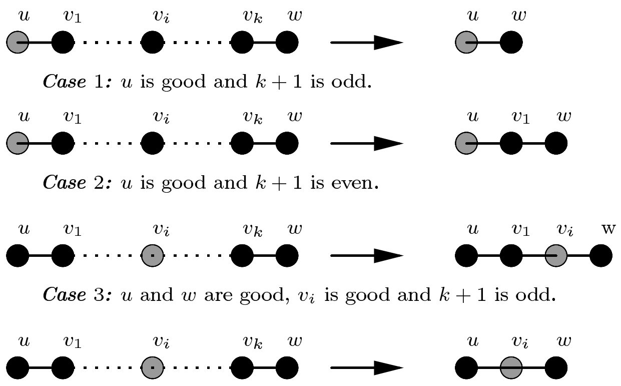
Preprocessing Algorithm $P(G, S, \mathsf{Col})$
Input: A multigraph G whose vertices have been colored good or bad along with a partially constructed solution S.
Output: A (possibly smaller) colored multigraph along with a (possibly larger) solution or NO signifying that there does not exist a solution containing good vertices only.
Let B be the set of bad vertices of G. Perform the following steps as long as possible.

1. If G has a vertex of degree ≤ 1, remove it along with the incident edge.
2. Check whether $G[B]$ is bipartite. If not then return NO.
3. Check whether any connected component of G is a cycle. Remove all connected components that are even cycles; if a connected component C is an odd cycle, include any good vertex of C in S and remove this cycle.
4. If G has a vertex v_i of degree 2 (which is not a self-loop) then it must be that v_i is part of some path of the form $uv_1 \dots v_i \dots v_k w$ where each v_j $1 \leq j \leq k$ is a degree 2 vertex and degree of u and w are ≥ 3. Here u and w could be the same vertex.
   (a) Case 1: At least one of the u and w is colored good. If the path length ($k+1$ above) between u and w is odd then delete all the vertices v_j and add the edge (u, w) (although u and w might already have an edge between them). If the path length between u and w is even then delete vertices $v_2, v_3, \dots, v_k$ and add an edge between v_1 and w and color v_1 bad.
   (b) Case 2: Both u and w are colored bad. Suppose the path length ($k+1$ above) is even and there is at least one v_i colored good, then replace the path $uv_1v_2 \dots v_k w$ by uv_iw. If no vertex v_i is colored good then replace the path $uv_1v_2 \dots v_k w$ by uv_1w. Next suppose the path length is odd. If there is a good vertex v_i then replace the path $uv_1v_2 \dots v_k w$ by uv_iv_jw, where $i \neq j$ and color v_j bad if it is not already so. If there are no good vertices then replace the path $uv_1v_2 \dots v_k w$ by the edge uw (*See Figure 3.*)
5. Include all vertices with self loops in S and remove them from the graph. Also remove degree 2 vertices from all cycles of length 2. Find all triangles Δ_{uvw} with $\mathsf{Col}(u) = \mathsf{good}$, $\mathsf{Col}(v) = \mathsf{bad}$, $\mathsf{Col}(w) = \mathsf{bad}$. Set $S \leftarrow S \cup \{u\}$ and remove u from the graph.

Fig. 2. The preprocessing algorithm for the Vertex Bipartization problem

Case 2: Both u and w are colored bad. If the path $\mathscr{P}_{1k} = v_1 \dots v_k$ does not have any good vertex then we only need to worry about maintaining the parity of the cycles passing through it and hence this case is similar to Case 1. But if $\mathscr{P}_{1k}$ has a good vertex, say v_i, then not only do we need to maintain parity of the cycles passing through it but we also need to retain at least one good vertex from this path. This is because the good vertices on $\mathscr{P}_{1k}$ could be the only good ones on the odd cycles passing through it. We retain exactly one good vertex since any optimal solution contains at most one good vertex from this path. Hence if $\mathscr{P}_{1k}$ is an odd length path then we retain v_i and another vertex v_j and add the edges (u, v_i), (v_i, v_j), and (v_j, w), and we color v_j bad. Else, we retain only v_i and add the edges (u, v_i), (v_i, w).

In Step 5, we add vertices having self-loops in our solution. This is because vertices with self-loops represent an odd cycle in the original graph and there-



Case 1***:*** u is good and $k+1$ is odd.

Case 2***:*** u is good and $k+1$ is even.

Case 3***:*** u and w are good, v_i is good and $k+1$ is odd.

Case 4***:*** u and w are bad, v_i is good and $k+1$ is even.

Fig. 3. Step 4 of the Preprocessing Algorithm P. The black vertices are bad and the shaded ones are good.

fore must be included in any minimum solution. A similar argument holds for triangles containing only one good vertex. A degree 2 vertex which is part of a length 2 cycle can be safely removed from the graph since such a vertex cannot be part of the solution as it is not part of any odd cycle in the current graph.

Time Complexity. Let $G = (V, E)$ be the input graph of maximum degree 3 with n vertices and m edges. Since the graph is of maximum degree 3, $m \leq \frac{3n}{2}$.

First we will show that if we reach Step $2a$ of the algorithm the number of good vertices n' in the current graph is at most $0.6n$. To do this we partition the set of good vertices into following three types.

Type 1: Degree 2 good vertices.
Type 2: Degree 3 good vertices with one good neighbor.
Type 3: Degree 3 good vertices with all bad neighbors.

Step 4 of the preprocessing routine ensures that any degree 2 good vertex u has both its neighbors bad. Also observe that a good vertex of degree 3 can have at most one good neighbor; for if not then there exists a path of length 2 containing only good vertices. Thus any good vertex will be of one of the three types mentioned above. Let n_1, n_2, and n_3 be the number of vertices of Type 1, Type 2, and Type 3 respectively. We obtain an upper bound on the number of good vertices by counting the number of edges between good and bad vertices. Define $n_g = n_1 + n_2 + n_3$. Define a ***good-bad edge*** to be one with one end point labelled good and the other labelled bad. Similarly define a ***good-good edge***. The number of good-bad edges in the current graph is $2n_1 + 2n_2 + 3n_3$ and the number of good-good edges is $n_2/2$. Moreover the graph has maximum degree 3. We therefore have the following inequalities.

$$2n_1 + 2n_2 + 3n_3 \leq \frac{3(n - n_1) + 2n_1}{2} - \frac{n_2}{2}$$

$$2.5(n_1 + n_2 + n_3) \leq \frac{3n}{2} \Rightarrow n_g \leq 0.6n.$$

In Step 2b, we find a path xyz of length 2 consisting of good vertices only. Here we have two situations to deal with. If we include the vertex z in the solution, we remove it from the graph which results in at least one good vertex y with degree at most 2 having a good neighbor x. But the preprocessing algorithm will either delete y or label it bad. In either case, the number of good vertices reduces by at least 2. If we don't pick z in our solution, then we label it bad, and this reduces the number of good vertices by 1. Thus the time complexity of the algorithm is bounded by the recurrence below:

$$\begin{aligned} T(n_g) &\leq T(n_g - 1) + T(n_g - 2) \\ T(0.6n_g) &= 2^{0.6n_g}. \end{aligned}$$

Here $T(n_g)$ is bounded by $(1.62)^{0.4n_g} \cdot 2^{0.6n_g}$ which is $O^*(1.8384^{n_g})$. Moreover on any path in the recursion tree, the algorithm takes polynomial space. Initially $n_g = n$ and therefore we have the following.

Theorem 1. *Let $G = (V, E)$ be an undirected graph with maximum degree 3 with n vertices and m edges. Then* VERTEX BIPARTIZATION *problem on G can be solved exactly using polynomial space and in time $O^*(1.8384^n)$.*

2.2 The FVS and VBP Problems in Graphs with Maximum Degree 4

In this subsection, we extend the ideas described in previous section for the VERTEX BIPARTIZATION problem to graphs with maximum degree 4. We also give an exact algorithm for FEEDBACK VERTEX SET problem on graphs with maximum degree 4. Again both algorithms in this subsection rely on preprocessing and branching. We will use the same preprocessing algorithm for the VERTEX BIPARTIZATION problem. The preprocessing algorithm for FEEDBACK VERTEX SET is almost the same and is described in Figure 4.

The main strategy of the FEEDBACK VERTEX SET algorithm is to find a good vertex with a sufficient number of good neighbors so that on the branch where we include a good vertex v in the solution, we can either delete at least one good neighbor of v or color the neighbor bad without making any further branches. The detailed algorithm is described in Figure 5.

Correctness. The argument for correctness closely follows the one given for the VERTEX BIPARTIZATION problem in the previous section and is omitted.

Time Complexity. We claim that in Step 2a, the number of good vertices is bounded by $2n/3$. If we reach Step 2a then either $n' \leq 2n/3$ or every good vertex of degree three has at most one good neighbor and every good vertex of degree four has at most two good neighbors. In the case when $n' \leq 2n/3$, our claim follows trivially since the number of good vertices is $\leq n'$. As for the second case, we can bound the number of good vertices by counting the number of edges between good and bad vertices. We can have following types of good vertices:

```
Preprocessing Algorithm $P1(G, S, \mathsf{Col})$
Input: A multigraph $G$ whose vertices have been colored good or bad along with a partially constructed solution $S$.
Output: A (possibly smaller) colored multigraph along with a (possibly larger) solution or NO signifying that there does not exist a solution containing good vertices only.
Let $B$ be the set of bad vertices of $G$. Perform the following steps as long as possible.
 1. If $G$ has a vertex of degree $\leq 1$, remove it along with the incident edge.
 2. Check whether $G[B]$ is acyclic. If not then return NO.
 3. Check whether any connected component of $G$ is a cycle. If a connected component
    $C$ is a cycle, include any good vertex of $C$ in $S$ and remove this cycle.
 4. If $G$ has a vertex $v_i$ of degree 2 (which is not a self-loop) then it must be that $v_i$ is
    part of some path of the form $uv_1 \dots v_i \dots v_k w$ where each $v_j$ $1 \leq j \leq k$ is a degree 2
    vertex and $u$ and $w$ are vertices of degree $\geq 3$. Here $u$ and $w$ could be the same vertex.
    (a) Case 1: At least one of the vertices $u$ and $w$ is colored good. Then delete all the
        vertices $v_j$ and add the edge $(u, w)$ (although $u$ and $w$ might already have an edge
        between them).
    (b) Case 2: Both $u$ and $v$ are colored bad. Suppose there is at least one $v_i$ colored
        good, then replace the path $uv_1v_2 \dots v_k w$ by $uv_iw$. If no vertex $v_i$ is colored good
        then replace the path $uv_1v_2 \dots v_k w$ by the edge $uw$.
 5. Include all vertices with self loops in $S$ and remove them from the graph. Find all
    cycles of length 2 with a good vertex and include it in $S$ and remove it from the
    graph. Find all triangles $\Delta_{uvw}$ with $\mathsf{Col}(u) = $ good, $\mathsf{Col}(v) = $ bad, $\mathsf{Col}(w) = $ bad. Set
    $S \leftarrow S \cup \{u\}$ and remove $u$ from the graph.
```

Fig. 4. The preprocessing algorithm for the FEEDBACK VERTEX SET problem

Type 1. Degree 2 good vertex (with both its neighbors bad).
Type 2. Degree 3 good vertex with one good neighbor.
Type 3. Degree 3 good vertex with all its neighbors bad.
Type 4. Degree 4 good vertex with two good neighbors.
Type 5. Degree 4 good vertex with one good neighbor.
Type 6. Degree 4 good vertex with all its neighbors bad.

It should be clear that any good vertex at this stage falls in one of the types mentioned above. Let n_i represent the number of good vertices of Type i and let n_g be the total number of good vertices. Then $n_g = \sum_{i=1}^{6} n_i$.

We will now count the number of good-bad edges in the graph. The total number of good-bad edges is $2n_1 + 2n_2 + 3n_3 + 2n_4 + 3n_5 + 4n_6$. It is easy to see that the quantity $n_2 + 2n_4 + n_5$ counts every good-good edge *twice*. Thus number of good-good edges is $(n_2 + 2n_4 + n_5)/2$. Hence,

$$2n_1 + 2n_2 + 3n_3 + 2n_4 + 3n_5 + 4n_6 \leq \frac{4(n - n_1 - n_2 - n_3) + 3(n_2 + n_3) + 2n_1}{2} - \frac{n_2}{2} - \frac{2n_4}{2} - \frac{n_5}{2}$$

A simple calculation shows that $3n_g \leq 2n$ from which it follows that $n_g \leq \frac{2n}{3}$. This shows that number of good vertices in Step $2a$ is bounded by $2n/3$.

```
Algorithm FVS-D4($G = (V, E)$, $S$, Col)
Input: A multigraph $G = (V, E)$ with maximum degree 4, whose vertices have been colored.
Here $n$ is the number of vertices in the input graph. Initially the algorithm is called with
$S \leftarrow \emptyset$ and $\mathsf{Col}(v) = \mathsf{good}$ for all vertices $v \in G(V)$.
Output: A minimum feedback vertex set of $G$.

Step 1 Call $P1(G, S, \mathsf{Col})$. If $P1$ returns NO then return NO.
Step 2 Apply the first step which is applicable:
    Step 2a Let $n'$ be the size of the current graph. If $n' \leq 2n/3$ or if every good vertex
        $v$ of degree 3 has at most one good neighbor and if every good vertex $v$ of degree 4
        has at most two good neighbors, then use brute-force and try all possible solutions
        $S \cup T$, where $T$ is some subset of the good vertices of the current graph, and return
        the one with minimum size.
    Step 2b Find a vertex $u$ of degree 3 with at least two good neighbors, say $v$ and $w$.
        Call the algorithm on following instances and return the smaller solution.
          – Set $S \leftarrow S \cup \{v\}$ and call FVS-D4($G - \{v\}$, $S$, Col).
          – Set $\mathsf{Col}(v) = \mathsf{bad}$ and call FVS-D4($G$, $S$, Col).
    Step 2c Find a vertex $u$ of degree 4 with at least 3 good neighbors, say $v$, $w$ and
        $z$. Here we consider three cases and branch accordingly: 1. $v$ is not part of the
        solution, 2. both $v$ and $w$ are part of the solution, and 3. $v$ is part of the solution
        but $w$ isn't and return the smallest solution.
          – Set $\mathsf{Col}(v) = \mathsf{bad}$ and call FVS-D4($G$, $S$, Col).
          – Set $S \leftarrow S \cup \{v, w\}$ and call FVS-D4($G - \{v, w\}$, $S$, Col).
          – Set $S \leftarrow S \cup \{v\}$ and $\mathsf{Col}(w) = \mathsf{bad}$ and call FVS-D4($G - \{v\}$, $S$, Col).
```

Fig. 5. Algorithm FVS-D4

In Step 2*b*, we have a good vertex u of degree 3 that has at least two good neighbors v and w. When we include v in the solution, the degree of u becomes 2 and since it has a good neighbor w it is either removed or labelled bad by the preprocessing step. Thus we end up eliminating at least two good vertices from the graph. If we do not include v in the solution, we label it bad and end up decreasing the number of good vertices by one. Then we have

$$T(n_g) \leq T(n_g - 1) + T(n_g - 2).$$

In Step 2*c*, we have a vertex u of degree 4 with at least three good neighbors v, w and z. We branch on three cases:

1. v is not in the solution,
2. v and w are in the solution, and
3. v is in the solution but w isn't.

In the first case, v is labelled bad and the number of good vertices reduces by at least 1. When both v, w are part of the solution then, on removing them, u has degree 2 and since it has a good neighbor z it is either removed or labelled bad by the preprocessing step. Thus we eliminate at least 3 good vertices in this case. In the last case, v is removed from the graph and w is labelled bad, which reduces the number of good vertices by at least 2. Thus we have the following recurrence on the number of good vertices.

$$T(n_g) \leq T(n_g - 1) + T(n_g - 2) + T(n_g - 3).$$

Combining the above, we get the following recurrence for the problem in the worst case, modulo the polynomial time used at every node to find the vertex of required type.

$$\begin{aligned} T(n_g) &\leq T(n_g - 1) + T(n_g - 2) + T(n_g - 3) \\ T(2n_g/3) &= 2^{2n_g/3} \end{aligned}$$

$T(n_g)$ is bounded by $(1.8393)^{n_g/3} \cdot 2^{2n_g/3}$ which is $O^*(1.945^{n_g})$. As initial value of n_g is n we get the following theorem.

Theorem 2. *Let $G = (V, E)$ be an undirected graph with maximum degree 4 with n vertices and m edges. Then the* Feedback Vertex Set *problem on G can be solved exactly using polynomial space and in time $O^*(1.945^n)$.*

We can modify our algorithm for the Feedback Vertex Set presented above for the Vertex Bipartization problem in graphs of maximum degree 4. The only modification we need to do is to call the preprocessing algorithm for Vertex Bipartization problem in Step 1. This gives us following theorem.

Theorem 3. *Let $G = (V, E)$ be an undirected graph with maximum degree 4 with n vertices and m edges. Then the* Vertex Bipartization *problem on G can be solved exactly using polynomial space and in time $O^*(1.945^n)$.*

3 Using FPT Algorithms to Design Exact Algorithms

In the last section, we gave efficient algorithms for Vertex Bipartization and Feedback Vertex Set, but they critically used the fact that the maximum degree of the graphs is 3 or 4. Here we give a general technique of designing exact algorithms using parameterized algorithms as a subroutine and apply it to several problems. Let Q be an NP-optimization problem such that for every instance I of Q there is a polynomial time computable universe U of size, say n, such that an optimum solution of I is a subset of U and suppose that its parameterized version (Q, k) (given an instance of Q, does it have a solution of size k?) is fixed parameter tractable having an algorithm $\mathscr{A}$ with time complexity $O^*(c^k)$. This algorithm $\mathscr{A}$ immediately gives us an exact algorithm for Q with time complexity $O^*(c^n)$, where n is an upper bound on the optimum solution size. What is interesting is that the FPT algorithm can actually give us an exact algorithm for Q with time complexity $O^*(d^n)$, where $d < c$. Moreover, if $c < 4$ then we will show that $d < 2$.

This fact has an interesting consequence. There are many optimization problems such as Max Independent Set, Min Vertex Cover, Min Feedback Vertex Set which have trivial brute-force enumeration algorithms of time complexity 2^n. If the parameterized versions of any of these problems is solvable in time $O^*(c^k)$, where $c < 4$ then we immediately obtain exact algorithms for these

```
Algorithm Exact($Q$,$\mathscr{A}$,$c$)
($Q$ is a minimization problem and $\mathscr{A}$ is the FPT algorithm that solves its parameterized
version in time $O^*(c^k)$, where $c$ is a constant and $k$ is the parameter. Here $n$ is the size
of the universe $U$.)
Compute the largest $\lambda$ such that $c^{\lfloor n\lambda \rfloor} \leq \binom{n}{n-\lfloor \lambda n \rfloor}$.

 for $i = 1$ to $\lfloor \lambda n \rfloor$
     use the FPT algorithm $\mathscr{A}$ for $Q$ to check whether there is solution of size $i$; if yes
     output $i$ and halt.
 for $i = \lfloor \lambda n \rfloor + 1$ to $n$
     try all subsets of size $i$ of $U$ to check whether there exists a solution of size $i$; if yes,
     then output $i$ and halt.
```

Fig. 6. Algorithm Exact()

problems which are better than the trivial brute-force algorithms. We will show that this technique simplifies exact algorithms for many optimization problems and for some (e.g. VERTEX BIPARTIZATION) gives the best known exact algorithm.

Our algorithm makes clever use of the FPT algorithm $\mathscr{A}$ and brute-force enumeration. Consider a problem such as VERTEX BIPARTIZATION. Had we used brute-force throughout, the time complexity would have been $O^*(\sum_{i=0}^{i=n} \binom{n}{i})$ $= O^*(2^n)$. It is well known that the function $\binom{n}{i}$ increases with increasing i, attains a maximum at $i = n/2$, and then decreases. Brute-force pushes the time complexity to $O^*(2^n)$ because it is costlier to search exhaustively when i is near $n/2$, since $\binom{n}{n/2} \approx 2^n$. Therefore, if we adopt the strategy of using brute-force only for those values of i which are far removed from $n/2$ and using the FPT algorithm $\mathscr{A}$ for the remaining i values (that is, those near $n/2$), then we might end up with an exponential time complexity better than that of $\mathscr{A}$. And indeed we do. Our algorithm is given in Figure 6. For simplicity the algorithm considers minimization problems only. For maximization problems we can modify the algorithm to output the largest i for which there exists a solution.

Suppose the FPT algorithm $\mathscr{A}$ for Q takes $O^*(c^k)$ time, where c is some constant. Then from the description of Algorithm Exact, it is easy to observe that its time complexity is upper bounded by following:

$$O^*\left(\max\left\{c^{\lambda n}, \binom{n}{n-\lfloor \lambda n \rfloor}\right\}\right).$$

The trivial brute-force algorithm for Q (enumeration of subsets of U) has time complexity $O^*(2^n)$. We show that if we want Algorithm Exact to beat this trivial time bound then we must have $c < 4$. We need a lemma.

Lemma 1. *Let $\frac{1}{2} < \lambda < 1$. Then $\binom{n}{n-\lambda n}$ is bounded by d^n, where d is some constant < 2.*

Proof. We know that

$$\binom{n}{n-\lambda n} = \binom{n}{\lambda n} \leq \frac{n^n}{(\lambda n)^{\lambda n}((1-\lambda)n)^{(1-\lambda)n}} = \left(\left(\frac{1}{\lambda}\right)^{\lambda}\left(\frac{1}{1-\lambda}\right)^{1-\lambda}\right)^n$$

One can easily verify using calculus that the function

$$h(\lambda) = \left(\tfrac{1}{\lambda}\right)^{\lambda} \left(\tfrac{1}{1-\lambda}\right)^{1-\lambda} \quad (0 < \lambda < 1)$$

attains a maximum of 2 at $\lambda = 1/2$. At other points in the interval $(\frac{1}{2}, 1)$ it has a value less than 2. This proves the claim. □

Equating c^{λ} and $h(\lambda) = d$ we get $c = \{h(\lambda)\}^{1/\lambda} < 2^{1/\lambda}$. Since $1/2 < \lambda < 1$ we see that $c < 4$. Also note that $d = c^{\lambda} < c$. We thus have the following theorem.

Theorem 4. *Let Q be an NP-optimization problem such that for every instance I of Q there is a polynomial time computable universe U of size, say n, such that an optimum solution of I is a subset of U. Suppose that the parameterized version of Q is FPT with time complexity $O^*(c^k)$ then there is an exact algorithm for Q with time complexity $O^*(d^n)$, where $d = c^{\lambda}$ and λ (< 1) is the largest value such that $c^{\lfloor n\lambda \rfloor} \leq \binom{n}{n-\lfloor \lambda n \rfloor}$. In particular $c < 4$ implies $d = c^{\lambda} < 2$.*

3.1 Applications

In this section, we apply Theorem 4 to obtain exact algorithms with nontrivial worst-case time bounds for several problems.

Vertex Bipartization Problem : The Vertex Bipartization problem in general undirected graphs can be solved exactly in $O^*(2^{|V|})$ time. Reed, Kaleigh, and Vetta [12] have recently given an FPT algorithm for the parameterized version of this problem with running time $O(3^k kmn)$. If we use their FPT algorithm directly to solve the optimum version of the problem we will take time $O^*(3^n)$ which is worse than that taken by the trivial exponential time algorithm. However, if we use the algorithm of Theorem 4 then with $c = 3$, we obtain $\lambda = 0.6091$ and get a running time of $O^*(1.9526^n)$. We therefore have the following theorem.

Theorem 5. *Let $G = (V, E)$ be an undirected graph with n vertices then* Vertex Bipartization *problem can be solved in time $O^*(1.9526^n)$.*

3- and 4-Hitting Set Problems: The Hitting Set (HS) problem is defined as follows:

Instance	A finite family of sets $S_1, S_2, \ldots, S_m$ comprised of elements from a universal set U.
Goal	Find a minimum sized subset $T \subseteq U$ such that $S_i \cap T \neq \emptyset$ for all i.

The 3- and 4-HS problems are special cases of the Hitting Set problem. In the 3-HS problem $|S_i|$ $(1 \leq i \leq m)$ is bounded by 3 and in the 4-HS problem by 4. The parameterized versions of these problems have been shown to be fixed parameter tractable by Neidermeier et al [8]. The main results in [8] can be summarized in the following theorem.

Theorem 6. *[8] The parameterized version of the 3-HS and the 4-HS problem can be solved in time $O^*(2.27^k)$ and $O^*(3.3^k)$ respectively.*

Using the parameterized algorithm of Theorem 6, we get $\lambda = 0.72$ with $c = 2.27$ for the 3-HS problem and $\lambda = 0.5721$ with $c = 3.3$ for the 4-HS problem. This gives us the following theorem.

Theorem 7. *The* 3- *and* 4-Hitting Set *problems can be solved exactly in time* $O^*(1.80933)^n$ *and* $O^*(1.9799^n)$, *where* $n = |U|$.

Recently Wahlström [15] proposed an exact algorithm for the 3-HS problem with time complexity $O^*(1.6316^n)$. The algorithm for the 3-HS problem in [15] does not directly generalize to the 4-HS problem. To the best of our knowledge our algorithm is the first exact algorithm for the 4-HS problem with the base of the exponent less than 2.

Feedback Set Problems in Tournaments: The Feedback Arc (Vertex) Set problem in directed graphs is defined as follows:

Instance	A directed graph $G = (V, E)$.
Goal	Find a minimum sized subset $F \subseteq E$ ($F \subseteq V$) such that $G' = (V, E - F)$ ($G' = (V - F, E')$) is acyclic.

Raman and Saurabh [10] give an $O^*(2.27^k)$ algorithm for the Feedback Vertex Set problem. For the Feedback Vertex Set we get $\lambda = 0.72$ with $c = 2.27$. Then using Theorem 4 we obtain the following theorem.

Theorem 8. *Let* $G = (V, E)$ *be a tournament with* n *vertices and* m *arcs. Then the feedback vertex set can be found in time* $O^*(1.80933^n)$.

For Feedback Arc Set, c is 2.415 by [10]. We get $\lambda = 0.696$ with $c = 2.415$. Observe that in any directed graph, the size of the minimum feedback arc set is at most $m/2$. This fact ensures that algorithm in Theorem 4 will never use brute-force as $\frac{m}{2} \leq 0.696m$. Hence the time complexity of the algorithm is bounded by $O^*((2.415)^{m/2})$ and therefore we get the following theorem.

Theorem 9. *Let* $G = (V, E)$ *be a tournament with* n *vertices and* m *arcs. Then the minimum feedback arc set can be found in time* $O^*(1.5541^m)$.

Max Cut in Graphs of Average Degree 3 or 4: An instance of the Edge Bipartization problem is an undirected graph $G = (V, E)$ and the question is to find a minimum set of edges that needs to be deleted from G to make it bipartite. The relationship between this problem and the Max Cut problem is straightforward. The maximum cut in a graph is:

$$E - \{\text{the minimum set of edges to make the graph bipartite}\}.$$

Thus solving the Edge Bipartization problem is equivalent to solving the Max Cut problem. The parameterized algorithm for Edge Bipartization presented in [7] has time complexity $O^*(2^k)$.

Poljak et al in [9] give a lower bound of $\frac{m}{2} + \frac{1}{2}\left\lceil\frac{n-c}{2}\right\rceil$ for maximum cut, where c is number of connected components and m and n are, respectively, the number of edges and vertices in the graph. If G is a connected graph on n vertices and m edges with average degree 3, then $m = 1.5n$. The minimum number of edges to be removed from G to make it bipartite is then

$$m - |\text{max cut}| \leq m - \left(\frac{m}{2} + \frac{n-1}{4}\right) = \frac{3n}{2} - \left(\frac{3n}{4} + \frac{n-1}{4}\right) = \frac{n}{2} + \frac{1}{4}.$$

We use the parameterized algorithm in [7] to solve the EDGE BIPARTIZATION problem. Since $\lambda = 0.773$ for $c = 2$, algorithm in Theorem 4 will never use brute-force when the input graph has average degree 3, since in this case the solution size is bounded by $n/2$ as shown above. Because of the upper bound on the number of edges needed to be removed, we achieve a time complexity of $O^*(2^{n/2}) = O^*(1.414^n)$. Had G been of average degree 4, then the upper bound on the number of edges to be deleted would be $3n/4 + 1/4$. This upper bound is also less than $0.773m$ and hence algorithm in theorem 4 will never use brute-force. So the time complexity of solving the EDGE BIPARTIZATION problem would be $O^*(2^{3n/4}) = O^*(1.6818^n)$. We thus have the following theorem.

Theorem 10. *The Max Cut problem can be solved exactly in time* $O^*(1.4141^n)$ *in graphs with average degree* 3 *and in time* $O^*(1.6818^n)$ *in graphs with average degree* 4. *Both of these algorithms take polynomial space.*

Neidermeir et al [6] achieve the same time bound for graphs with maximum degree 3 and a better time bound of $O^*(1.5871^n)$ for graphs with maximum degree 4.

4 Conclusion

We have obtained improved exact algorithms for several problems including VERTEX BIPARTIZATION in general undirected graphs, 4-HITTING SET, FEEDBACK VERTEX SET in graphs with maximum degree at most 4, and FEEDBACK ARC SET in tournaments. We introduced two general techniques to obtain efficient exact algorithms. One of these is a modified version of the general branch-and-bound technique and the other one is based on parameterized complexity algorithms. Further reduction in the base of the exponent of all these algorithms remains open.

It would be interesting to investigate the practical performance of these algorithms. Another major open problem is to devise an exact algorithm with time complexity less than $O^*(c^n)$, $c < 2$ for the FEEDBACK VERTEX SET problem in general undirected graphs.

After submitting this paper, we learnt about an exact algorithm on VERTEX BIPARTIZATION by Byskov [1], with time complexity $O^*(1.8631^n)$. An improvement to this result along with several other results are presented in the full version of the paper [11].

References

1. J. M. BYSKOV. *On the Number of Maximal Bipartite Subgraphs of a Graph.* Journal of Graph Theory 48 (2): 127-135, 2005.
2. H. CHOI, K. NAKAJIMA AND C. S. RIM. *Graph Bipartization and Via Minimization.* SIAM Journal of Discrete Mathematics 2 (1): 38-47, 1989.

3. R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
4. R. Downey and M. Fellows. *Parameterized Complexity for the Skeptic.* In the Proc. of 18th CCC : 147-169, 2003.
5. N. Garg, V. Vazirani and M. Yannakakis. *Approximate Max-Flow Min-(Multi) Cut Theorems and Their Applications.* SIAM Journal on Computing 25 (2): 235-251, 1996.
6. J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith *Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT.* Discrete Applied Mathematics 130 (2): 139-155, 2003.
7. J. Guo, J. Gramm, F. Hüffner, R. Neidermeier and S. Wernicke. *Improved Fixed-Parameter Algorithms for Two Feedback Set Problems.* To appear in the proceedings of 9th Workshop on Algorithms and Data Structures (WADS). Springer Verlag, Lecture Notes in Computer Science, 2005.
8. R. Niedermeier and P. Rossmanith. *An efficient fixed parameter algorithm for 3-Hitting Set.* Journal of Discrete Algorithms 1 (1): 89-102, 2003.
9. S. Poljak and D. Turzik. *A Polynomial Algorithm for Constructing a Large Bipartite Subgraph, with an Application to a Satisfiability Problem.* Canad. J. Math 34 (3): 519-524, 1982.
10. V. Raman, and S. Saurabh. *Improved Parameterized Algorithms for Feedback Set Problems in Weighted Tournaments.* In the Proc. of 1st International Workshop on Exact and Parameterized Algorithms (IWPEC): 260-270, 2004.
11. V. Raman, S. Saurabh, and S. Sikdar. *Exact Algorithms for Odd Cycle Transversal and Other Problems.* Technical Report, IMSC-2005-06-16, The Institute of Mathematical Sciences, 2005.
12. B. Reed, K. Smith and A. Vetta. *Finding Odd Cycle Transversals*, Operations Research Letters 32: 299-301, 2004.
13. R. Rizzi, V. Bafna, S. Istrail and G. Lancia. *Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem.* In the Proc of WABI: 29-43, 2002.
14. S. Ueno, Y. Kajitani and S. Gotoh. *On the Nonseparating Independent Set Problem and Feedback Set Problem for Graphs with no Vertex Degree Exceeding Three.* Discrete Mathematics 72: 355-360, 1988.
15. M. Wahlström. *Exact algorithms for finding minimum transversals in rank-3 hypergraphs.* Journal of Algorithms 51(2): 107 - 121, 2004.
16. G. Woeginger. *Exact algorithms for NP-hard problems: A survey.* In *Combinatorial Optimization—Eureka! You shrink!* Springer LNCS 2570: 185-207, 2003.
17. X. Zhunag and S. Pande. *Resolving Register Bank Conflicts for a Network Processor.* In Proc. of 12th PACT: 260-278, 2003.

A Typed Semantics of Higher-Order Store and Subtyping

Jan Schwinghammer

Informatics, University of Sussex, Brighton, UK
j.schwinghammer@sussex.ac.uk

Abstract. We consider a call-by-value language, with higher-order functions, records, references to values of arbitrary type, and subtyping. We adapt an intrinsic denotational model for a similar language based on a possible-world semantics, recently given by Levy [14], and relate it to an untyped model by a logical relation. Following the methodology of Reynolds [22], this relation is used to establish coherence of the typed semantics, with a coercion interpretation of subtyping. We obtain a typed denotational semantics of (imperative) object-based languages.

1 Introduction

Languages such as Standard ML and Scheme allow to store values of arbitrary types, including function types. Essentially the same effect is pervasive in object-based languages (see [1,21]), where objects are created on-the-fly and arbitrary method code needs to be kept in the store. This feature is often referred to as *higher-order store* or *general references*, and complicates the semantics (and logics) of such languages considerably: Besides introducing recursion to the language [13], higher order store in fact requires the semantic domain to be defined by a *mixed-variant* recursive equation. So far, only few models of (typed) languages with general references appeared in the literature [4,5,14], and most of the work done on semantics of storage does not readily apply to languages with higher-order store.

In a recent paper, Paul Levy proposed a typed semantics for a language with higher-order functions and higher-order store [14]. This is a possible worlds model, explicating the dynamic allocation of new (typed) storage locations in the course of a computation. We recall this model below, and extend it to accommodate subtyping by using coercion maps. In the terminology of Reynolds [22], we obtain an *intrinsic* semantics: Meaning is given to derivations of typing judgements, rather than to terms, with the consequence that

– ill-typed phrases are meaningless,
– terms satisfying several judgements will be assigned several meanings, and
– coherence between the meaning of several derivations of the *same* judgement must be established.

Due to the addition of subtyping to Levy's model, derivations are indeed no longer unique and we must prove coherence. A standard approach for such proofs is to transform derivations into a normal form while preserving their semantics. This can be quite involved, even for purely functional languages (see, e.g., [7]).

In contrast to intrinsic semantics, an *extrinsic* semantics gives meaning to *all* terms. Types (and typing judgements) are interpreted as, e.g., predicates or partial equivalence relations over an untyped model. Usually, the interpretation of subtyping is straightforward in such models. In [22], Reynolds uses a logical relation between intrinsic and extrinsic cpo models of a lambda calculus with subtyping (but no state) to prove coherence. The proof essentially relies on the fact that (the denotations of) all derivations of a judgement $\Gamma \rhd e : A$ are related to the denotation $[\![e]\!]$ of e in the untyped model underlying the extrinsic semantics, via the *basic lemma* of logical relations. A family of retractions between intrinsic and extrinsic semantics is then used to obtain the meaning of $[\![\Gamma \rhd e : A]\!]$ in terms of Γ, $[\![e]\!]$ and A alone, i.e., independent of any particular derivation of the judgement.

We apply the same ideas to obtain a coherence proof for the language considered here. Two modifications have to be made: Firstly, because of the indexing by worlds we use a *Kripke logical relation* [16] to relate intrinsic and extrinsic semantics — this is straightforward. Secondly, due to the mixed-variant recursion forced by the higher-order store we can no longer use induction over the type structure to establish properties of the relations. In fact even the existence of the logical relation requires a non-trivial proof — we use the framework of Pitts [18] to deal with this complication.

While the combination of higher-order storage and subtyping is interesting in its own right, we see the current work as a step toward our longer-term goal of investigating logics for languages involving higher-order store. In particular, we are interested in semantics and reasoning principles for object-oriented programs, and it should be noted that a number of object encodings used a target language similar to the one considered here [2,11]. Some evidence that the model of this paper can indeed serve as basis for such logics is provided, by giving a semantics to the object calculus [1]. This is done using a typed variant of Kamin and Reddy's "closure model" [11]. To the best of our knowledge this is the first (intrinsically typed) domain-theoretic model of the imperative object calculus.

In summary, our technical contributions here are (1) we present a model of a language that includes general references and subtyping, (2) we successfully apply the ideas of Reynolds [22] to prove coherence, and (3) we provide the first (typed) model of the imperative object calculus, based on cpos.

Structure of the Paper. In the next section, language and type system are introduced. Then, typed and untyped models are presented (Sects. 3 and 4). The logical relation is defined next, and retractions between types of the intrinsic semantics and the untyped value space are used to prove coherence in Sect. 6. In Sect. 7 both a derived per semantics and an interpretation of objects in the model are discussed.

Complete proofs, further examples and discussions can be found in the technical report [23].

2 Language

We consider a single base type of booleans, bool, records $\{\mathsf{m}_i : A_i\}_{i \in I}$ with labels $\mathsf{m} \in \mathbb{L}$, and function types $A \Rightarrow B$. We set $\mathbf{1} \overset{def}{=} \{\}$ to be the empty record type. Finally, we have a type $\mathsf{ref}\ A$ of mutable references to values of type A. Term forms include constructs for creating, dereferencing and updating of storage locations. The syntax of types and terms is given by the grammar:

$$\begin{aligned}
A, B \in \mathit{Type} &::= \mathsf{bool} \mid \{\mathsf{m}_i : A_i\}_{i \in I} \mid A \Rightarrow B \mid \mathsf{ref}\ A \\
v \in \mathit{Val} &::= x \mid \mathsf{true} \mid \mathsf{false} \mid \{\mathsf{m}_i = x_i\}_{i \in I} \mid \lambda x.e \\
e \in \mathit{Exp} &::= v \mid \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 \mid \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid x.\mathsf{m} \mid x(y) \\
&\quad\ \mid \mathsf{new}_A\ x \mid \mathsf{deref}\ x \mid x{:=}y
\end{aligned}$$

The subtyping relation $A \preceq B$ is the least reflexive and transitive relation closed under the rules

$$\frac{A_i \preceq A'_i \ \ \forall i \in I' \quad I' \subseteq I}{\{\mathsf{m}_i : A_i\}_{i \in I} \preceq \{\mathsf{m}_i : A'_i\}_{i \in I}} \qquad \frac{A' \preceq A \quad B \preceq B'}{A{\Rightarrow}B \preceq A'{\Rightarrow}B'}$$

Note that there is no rule for reference types as these need to be invariant, i.e., $\mathsf{ref}\ A \preceq \mathsf{ref}\ B$ only if $A \equiv B$. A type inference system is given in Table 1, where contexts Γ are finite sets of variable-type pairs, with each variable occurring at most once. As usual, in writing $\Gamma, x{:}A$ we assume x does not occur in Γ. A subsumption rule is used for subtyping of terms.

Table 1. Typing

$$\frac{\Gamma \rhd e : A \quad A \preceq B}{\Gamma \rhd e : B} \qquad \frac{x{:}A \in \Gamma}{\Gamma \rhd x : A}$$

$$\frac{\Gamma \rhd e_1 : B \quad \Gamma, x{:}B \rhd e_2 : A}{\Gamma \rhd \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 : A} \qquad \frac{}{\Gamma \rhd \mathsf{true} : \mathsf{bool}}$$

$$\frac{\Gamma \rhd x : \mathsf{bool} \quad \Gamma \rhd e_1 : A \quad \Gamma \rhd e_2 : A}{\Gamma \rhd \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A} \qquad \frac{}{\Gamma \rhd \mathsf{false} : \mathsf{bool}}$$

$$\frac{\Gamma \rhd x_i : A_i \ \ \forall i \in I}{\Gamma \rhd \{\mathsf{m}_i = x_i\}_{i \in I} : \{\mathsf{m}_i : A_i\}_{i \in I}} \qquad \frac{\Gamma \rhd x : \{\mathsf{m}_i : A_i\}_{i \in I}}{\Gamma \rhd x.\mathsf{m}_j : A_j}\ (j \in I)$$

$$\frac{\Gamma, x{:}A \rhd e : B}{\Gamma \rhd \lambda x.e : A \Rightarrow B} \qquad \frac{\Gamma \rhd x : A \Rightarrow B \quad \Gamma \rhd y : A}{\Gamma \rhd x(y) : B}$$

$$\frac{\Gamma \rhd x : A}{\Gamma \rhd \mathsf{new}_A\ x : \mathsf{ref}\ A} \qquad \frac{\Gamma \rhd x : \mathsf{ref}\ A}{\Gamma \rhd \mathsf{deref}\ x : A}$$

$$\frac{\Gamma \rhd x : \mathsf{ref}\ A \quad \Gamma \rhd y : A}{\Gamma \rhd x{:=}y : \mathbf{1}}$$

3 Intrinsic Semantics

In this section we recall the possible worlds model of [14]. Its extension with records is straightforward, and we interpret the subsumption rule using coercion maps.

Worlds. For each $A \in \mathit{Type}$ let Loc_A be mutually disjoint, countably infinite sets of *locations.* We let l range over $\mathsf{Loc} \overset{def}{=} \bigcup_{A \in \mathit{Type}} \mathsf{Loc}_A$, and may use the notation l_A to emphasize that $l \in \mathsf{Loc}_A$. A *world* w is a finite set of locations $l_A \in \mathsf{Loc}$. A world w' extends w, written $w' \geq w$, if $w' \supseteq w$. We write $\mathcal{W} = (\mathcal{W}, \leq)$ for the poset of worlds.

Semantic Domain. Let **pCpo** be the category of cpos (not necessarily containing a least element) and partial continuous functions. For a partial continuous function f we write $f(a) \downarrow$ if the application is defined, and $f(a) \uparrow$ otherwise. Let **Cpo** be the subcategory of **pCpo** where the morphisms are *total* continuous functions.

Informally, a world describes the shape of the store, i.e., the number of locations of each type allocated in the store. In the semantics we want a cpo S_w of w-stores for each $w \in \mathcal{W}$, and a cpo $[\![A]\!]_w$ of values of type A. In fact, we require that each $[\![A]\!]$ denotes a co-variant functor from $\mathcal{W}$ to **Cpo**, formalising the intuition that values can always be used with larger stores. We write the image of $w \leq w'$ under $[\![A]\!]$ as $[\![A]\!]_w^w$.

The cpo of w-stores is defined as $S_w = \prod_{l_A \in w} [\![A]\!]_w$. For worlds $w \in \mathcal{W}$, $[\![\mathsf{bool}]\!]_w = \mathsf{BVal}$ denotes the set $\{\mathit{true}, \mathit{false}\}$ of truth values considered as discrete cpo, and similarly, $[\![\mathsf{ref}\ A]\!]_w = \{l_A \mid l_A \in w\}$ is the discretely ordered cpo of A-locations allocated in w-stores. Further, $[\![\{\mathsf{m}_i : A_i\}_{i \in I}]\!]_w = \{\!|\mathsf{m}_i : [\![A_i]\!]_w |\!\}_{i \in I}$ is the cpo of records $\{\!|\mathsf{m}_i = a_i|\!\}_{i \in I}$ with component m_i in $[\![A_i]\!]_w$, ordered pointwise.

On morphisms $w \leq w'$, $[\![\mathsf{bool}]\!]_w^w = \mathsf{id}_{\mathsf{BVal}}$ is the identity map, and $[\![\mathsf{ref}\ A]\!]_w^w$ is the inclusion $[\![\mathsf{ref}\ A]\!]_w \subseteq [\![\mathsf{ref}\ A]\!]_w$. Records act pointwise on the components, $[\![\{\mathsf{m}_i : A_i\}]\!]_w^w = \lambda r. \{\!|\mathsf{m}_i = [\![A_i]\!]_w^w\ (r.\mathsf{m}_i)|\!\}$. The type of functions $A \Rightarrow B$ is the most interesting since it involves the store S,

$$[\![A \Rightarrow B]\!]_w \overset{def}{=} \prod_{w\ \geq w}(S_w\ \times [\![A]\!]_w\ \rightharpoonup \textstyle\sum_{w\ \geq w}(S_w\ \times [\![B]\!]_w\ )) \tag{1}$$

This says that a function $f \in [\![A \Rightarrow B]\!]_w$ may be applied in any future (larger) store w' to a w'-store s and value $v \in [\![A]\!]_w$. The computation $f_w\ (s, v)$ may allocate new storage, and upon termination it yields a store and value in a yet larger world $w'' \geq w'$. For a morphism $w \leq w'$, $[\![A \Rightarrow B]\!]_w^w\ (f) = \lambda_{w\ \geq w}\, f_w$ is the restriction to worlds $w'' \geq w'$.

Equation (1) clearly shows the effect of allowing higher-order store: Since functions $A \Rightarrow B$ can also be stored, S and $[\![A \Rightarrow B]\!]$ are mutually recursive. Due to the use of S in both positive and negative positions in (1) a mixed-variant domain equation for S must be solved. To this end, in [14] a *bilimit-compact* category $\mathcal{C}$ is considered, i.e.,

- $\mathcal{C}$ is **Cpo**-enriched and each hom-cpo $\mathcal{C}(A, B)$ has a least element $\perp_{A,B}$ s.t. $\perp \circ f = \perp = g \circ \perp$;
- $\mathcal{C}$ has an initial object; and
- in the category $\mathcal{C}^E$ of embedding-projection pairs of $\mathcal{C}$, every ω-chain $\Delta = D_0 \to D_1 \to \dots$ has an O-colimit [24], i.e., a cocone $\langle e_i, p_i \rangle_{i \in \mathbb{N}} : \Delta \to D$ in $\mathcal{C}^E$ s.t. $\bigsqcup_i e_i \circ p_i = \mathsf{id}_D$ in $\mathcal{C}(D, D)$.

It follows that every locally continuous functor $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ has a minimal invariant, i.e., an object D in $\mathcal{C}$ s.t. $F(D, D) = D$ (omitting isomorphisms) and id_D is the least fixed point of the continuous endofunction $\delta : \mathcal{C}(D, D) \to \mathcal{C}(D, D)$ given by $\delta(e) \overset{def}{=} F(e, e)$ [18].

Following [14] the semantics of types can now be obtained as minimal invariant of the locally continuous functor $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ (derived from the domain equations for types by separating positive and negative occurrences of the store) over the bilimit-compact category

$$\mathcal{C} \; = \; \textstyle\prod_{w \in \mathcal{W}} \mathbf{pCpo} \times \prod_{A \in \mathit{Type}} [\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}] \tag{2}$$

Here, $[\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}]$ denotes the category where objects are functors $A, B : \mathcal{W} \to \mathbf{Cpo}$ and morphisms are partial natural transformations $\mu : A \dot{\rightharpoonup} B$, i.e., for $A, B : \mathcal{W} \to \mathbf{Cpo}$ the diagram

$$\begin{tikzcd} A_w \arrow[r, rightharpoonup, "\mu_w"] \arrow[d, "A^w_w"'] & B_w \arrow[d, "B^w_w"] \\ A_w \arrow[r, rightharpoonup, "\mu_w"'] & B_w \end{tikzcd} \tag{3}$$

commutes. The first component of the product in (2) is used to define $S_w \overset{def}{=} D_{Sw}$ from the minimal invariant $D = \langle \{D_{Sw}\}_w, \{D_A\}_A \rangle$, and the second component yields $[\![A]\!] \overset{def}{=} D_A$. In fact, D gives isomorphisms $F(D, D)_A = D_A$ in the category $[\mathcal{W}, \mathbf{Cpo}]$ of functors $\mathcal{W} \to \mathbf{Cpo}$ and *total* natural transformations.

Semantics. Each subtyping derivation $A \preceq B$ determines a *coercion*, which is in fact a (total) natural transformation from $[\![A]\!]$ to $[\![B]\!]$, defined in Table 2. We follow the notation of [22] and write $\mathcal{P}(J)$ to distinguish a *derivation* of judgement J from the judgement itself.

Writing $[\![\Gamma]\!]_w$ for the set of maps from variables to $\bigcup_A [\![A]\!]_w$ s.t. $\rho(x) \in [\![A]\!]_w$ for all $x{:}A \in \Gamma$, we define the semantics of (derivations of) typing judgments

$$[\![\Gamma \rhd e : A]\!]_w : [\![\Gamma]\!]_w \to S_w \rightharpoonup \textstyle\sum_{w \ \geq w} (S_w \ \times [\![A]\!]_w \ ) \ .$$

As observed in Levy's paper, each *value* $\Gamma \rhd v : A$ determines a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$ in $[\mathcal{W}, \mathbf{Cpo}]$. Here this is a consequence of the fact that (i) values do not affect the store and (ii) coercion maps determine (total) natural transformations. We make use of this fact in the statement of the semantics. For example, in the case of records we do not have to fix an order for the evaluation of the components.

Table 2. Coercion maps

$$\left[\!\!\left[\frac{}{A \preceq A}\right]\!\!\right]_w = \mathsf{id}_{[\![A]\!]_w}$$

$$\left[\!\!\left[\frac{\mathcal{P}(A \preceq A')\;\; \mathcal{P}(A' \preceq B)}{A \preceq B}\right]\!\!\right]_w = [\![\mathcal{P}(A' \preceq B)]\!]_w \circ [\![\mathcal{P}(A \preceq A')]\!]_w$$

$$\left[\!\!\left[\frac{I' \subseteq I \quad \mathcal{P}(A_i \preceq A_i')\; \forall i \in I'}{\{\mathsf{m}_i : A_i\}_{i \in I} \preceq \{\mathsf{m}_i : A_i'\}_{i \in I}}\right]\!\!\right]_w = \lambda r.\{\!|\mathsf{m}_i = [\![\mathcal{P}(A_i \preceq A_i')]\!]_w\,(r.\mathsf{m}_i)|\!\}_{i \in I}$$

$$\left[\!\!\left[\frac{\mathcal{P}(A' \preceq A) \quad \mathcal{P}(B \preceq B')}{A \Rightarrow B \preceq A' \Rightarrow B'}\right]\!\!\right]_w$$

$$= \lambda f \lambda_{w\ \geq w}\, \lambda \langle s, x\rangle. \begin{cases} \langle w'', \langle s', [\![\mathcal{P}(B \preceq B')]\!]_w\;\; x'\rangle\rangle \\ \qquad\qquad \text{if } f_w\ \langle s, [\![\mathcal{P}(A' \preceq A)]\!]_w\;\; (x)\rangle = \langle w'', \langle s', x'\rangle\rangle \downarrow \\ \text{undefined otherwise} \end{cases}$$

The semantics of subtyping judgements is used for the subsumption rule,

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \rhd e : A)\;\; \mathcal{P}(A \preceq B)}{\Gamma \rhd e : B}\right]\!\!\right]_w \rho s = \begin{cases} \langle w', \langle s', [\![\mathcal{P}(A \preceq B)]\!]_w\;\; a\rangle\rangle \\ \qquad\qquad \text{if } [\![\mathcal{P}(\Gamma \rhd e : A)]\!]_w\, \rho s = \langle w', \langle s', a\rangle\rangle \downarrow \\ \text{undefined otherwise} \end{cases}$$

As explained above, the semantics of functions is parameterised over extensions of the current world w,

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma, x : A \rhd e : B)}{\Gamma \rhd \lambda x.e : A \Rightarrow B}\right]\!\!\right]_w \rho s$$

$$= \langle w, \langle s, \lambda w' \geq w \lambda \langle s', a\rangle.\, [\![\mathcal{P}(\Gamma, x{:}A \rhd e : B)]\!]_w\;\; ([\![\Gamma]\!]_w^w\;\; \rho)[x := a]\, s'\rangle\rangle$$

Function application is

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \rhd x : A \Rightarrow B) \quad \mathcal{P}(\Gamma \rhd y : A)}{\Gamma \rhd x(y) : B}\right]\!\!\right]_w \rho s = f_w(s, a)$$

where $\langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : A \Rightarrow B)]\!]_w\, \rho s$ and $\langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd y : A)]\!]_w\, \rho s$. The remaining cases are similarly straightforward (see [14,23]).

4 An Untyped Semantics

We give an untyped semantics of the language in **pCpo**. Let Val satisfy

$$\mathsf{Val} \;=\; \mathsf{BVal} + \mathsf{Loc} + \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) + (\mathsf{St} \times \mathsf{Val} \rightharpoonup \mathsf{St} \times \mathsf{Val}) \tag{4}$$

where $\mathsf{St} \overset{def}{=} \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Val})$ denotes the cpo of records with labels from Loc, ordered by $r_1 \sqsubseteq r_2$ iff $\mathsf{dom}(r_1) = \mathsf{dom}(r_2)$ and $r_1.\mathsf{m} \sqsubseteq r_2.\mathsf{m}$ for all $\mathsf{m} \in \mathsf{dom}(r_1)$. The interpretation of terms, $[\![e]\!] : \mathsf{Env} \to \mathsf{St} \rightharpoonup \mathsf{St} \times \mathsf{Val}$, is essentially straightforward, typical cases are those of abstraction and application:

$$[\![\lambda x.e]\!]\, \eta\sigma = \langle \sigma, \lambda\langle \sigma', v\rangle.\; [\![e]\!]\, \eta[x := v]\, \sigma'\rangle$$

$$[\![x(y)]\!]\, \eta\sigma = \begin{cases} \eta(x)\langle \sigma, \eta(y)\rangle \text{ if } \eta(x) \in [\mathsf{St} \times \mathsf{Val} \rightharpoonup \mathsf{St} \times \mathsf{Val}] \text{ and } \eta(y)\downarrow \\ \text{undefined} \quad\; \text{otherwise} \end{cases}$$

Table 3. Kripke logical relation

$\langle x,y\rangle \in R_w^{\mathsf{bool}}$	$\overset{def}{\iff}$	$y \in \mathsf{BVal} \;\wedge\; x = y$
$\langle r,s\rangle \in R_w^{\{\mathsf{m}_i:A_i\}}$	$\overset{def}{\iff}$	$s \in \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) \;\wedge\; \forall i.\ (s.\mathsf{m}_i\downarrow \;\wedge\; \langle r.\mathsf{m}_i, s.\mathsf{m}_i\rangle \in R_w^{A_i})$
$\langle f,g\rangle \in R_w^{A\Rightarrow B}$	$\overset{def}{\iff}$	$g \in [\mathsf{St}\times\mathsf{Val} \rightharpoonup \mathsf{St}\times\mathsf{Val}] \;\wedge$
		$\forall w' \geq w\ \forall\langle s,\sigma\rangle \in R_w^{St}\ \forall\langle x,y\rangle \in R_w^A$
		$\quad (f_w\ (s,x)\uparrow \;\wedge\; g(\sigma,y)\uparrow)$
		$\quad \vee\ \exists w'' \geq w'\ \exists s' \in S_w\ \ \exists x' \in [\![B]\!]_w\ \ \exists\sigma' \in \mathsf{St}\ \exists y' \in \mathsf{Val}.$
		$\qquad (f_w\ (s,x) = \langle w'', \langle s', x'\rangle\rangle \;\wedge\; g(\sigma,y) = \langle\sigma',y'\rangle$
		$\qquad \wedge\ \langle s',\sigma'\rangle \in R_w^{St} \;\wedge\; \langle x',y'\rangle \in R_w^B\ )$
$\langle x,y\rangle \in R_w^{\mathsf{ref}\ A}$	$\overset{def}{\iff}$	$y \in w \cap \mathsf{Loc}_A \;\wedge\; x = y$
with the auxiliary relation $R_w^{St} \subseteq S_w \times \mathsf{St}$,		
$\langle s,\sigma\rangle \in R_w^{St}$	$\overset{def}{\iff}$	$\mathsf{dom}(s) = w = \mathsf{dom}(\sigma) \;\wedge\; \forall l_A \in w.\ \langle s.l_A, \sigma.l_A\rangle \in R_w^A$

Compared to the intrinsic semantics of the previous section, there are now many more possibilities of undefinedness if things "go wrong", e.g., if x in $x(y)$ does not denote a function value.

The semantics of new_A may be slightly surprising as there is still some type information in the choice of locations:

$$[\![\mathsf{new}_A\ x]\!]\,\eta\sigma = \langle\sigma + \{\![l_A = \eta(x)]\!\}, l_A\rangle \qquad \text{where } l_A \in \mathsf{Loc}_A \setminus \mathsf{dom}(\sigma)$$

if $\eta(x)\downarrow$, and undefined otherwise. Informally, the worlds of the intrinsic semantics are encoded in the domain of untyped stores. Although σ with $\mathsf{dom}(\sigma) = w$ need not necessarily correspond to a (typed) w-store in any sense, this will be the case for stores being derived from well-typed terms. This is one of the results of Sect. 5 below. See also the discussion in Sect. 7.1.

5 A Kripke Logical Relation

While in [22] a logical relation between typed and untyped models was used to establish coherence, here this must be slightly generalised to a Kripke logical relation. Kripke logical relations are not only indexed by types but also by possible worlds, subject to a monotonicity condition (Lemma 2 below).

In Table 3 such a family of *Type*- and $\mathcal{W}$-indexed relations $R_w^A \subseteq [\![A]\!]_w \times \mathsf{Val}$ is defined. The existence of this family R has to be established: There are both positive and negative occurrences of R_w^{St} in the case of function types $A\Rightarrow B$. Thus R cannot be defined by induction on the type structure, nor does it give rise to a monotone operation (on the complete lattice of admissible predicates).

5.1 Existence of R_w^A

To establish the existence of such a relation one uses Pitts' technique for the bilimit-compact product category $\mathcal{C} \times \mathbf{pCpo}$. Let $G : \mathbf{pCpo}^{op} \times \mathbf{pCpo} \to \mathbf{pCpo}$

be the locally continuous functor for which (4) is the minimal invariant, so that $\langle D, \mathsf{Val}\rangle$ is the minimal invariant of $F \times G$. A relational structure $\mathcal{R}$ on the category $\mathcal{C} \times \mathbf{pCpo}$, in the sense of [18], is given by the following data.

– For each object $\langle X, Y\rangle$ of $\mathcal{C} \times \mathbf{pCpo}$, let $\mathcal{R}(X, Y)$ consist of the type- and world-indexed families R of admissible relations, where $R^A_w \subseteq X_{Aw} \times Y$ and $R^{St}_w \subseteq X_{Sw} \times \mathsf{Rec}_{\mathsf{Loc}}(Y)$.
– For morphisms $f = \langle f_1, f_2\rangle : \langle X, Y\rangle \to \langle X', Y'\rangle$, and relations $R \in \mathcal{R}(X, Y)$ and $S \in \mathcal{R}(X', Y')$, we define $f : R \subset S$ iff, for all $w \in \mathcal{W}$, $A \in \mathit{Type}$, for all $x \in X_{Aw}$, $y \in Y$, $s \in X_{Sw}$ and $\sigma \in \mathsf{Rec}_{\mathsf{Loc}}(Y)$,

$$\langle x, y\rangle \in R^A_w \implies \begin{cases} f_{1\,Aw}(x)\uparrow \wedge f_2(y)\uparrow \quad \text{or} \\ f_{1\,Aw}(x)\downarrow \wedge f_2(y)\downarrow \wedge \langle f_{1\,Aw}(x), f_2(y)\rangle \in S^A_w \end{cases}$$

$$\langle s, \sigma\rangle \in R^{St}_w \implies \begin{cases} f_{1\,Sw}(x)\uparrow \wedge \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\uparrow \quad \text{or} \\ f_{1\,Sw}(x)\downarrow \wedge \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\downarrow \wedge \langle f_{1\,Sw}(x), \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\rangle \in S^{St}_w \end{cases}$$

We define a functional $\Phi(R^-, R^+)$ on $\mathcal{R}$ corresponding to the equations for the Kripke logical relation R above (by separating positive and negative occurrences of R in the right-hand sides) such that for $S \in \mathcal{R}(X, Y)$ and $S' \in \mathcal{R}(X', Y')$ we have $\Phi(S, S') \in \mathcal{R}((F \times G)(\langle X, Y\rangle\langle X', Y'\rangle))$. The map Φ is an admissible action of the functor $F \times G$ on $\mathcal{R}$, in the following sense:

Lemma 1. *For all $e = \langle e_1, e_2\rangle, f = \langle f_1, f_2\rangle$ and R, R', S, S', if $e : R' \subset R$ and $f : S \subset S'$ then $(F \times G)(e, f) : \Phi(R, S) \subset \Phi(R', S')$.*

According to [18], Lemma 1 guarantees that Φ has a unique fixed point $\mathit{fix}(\Phi)$ in $\mathcal{R}(D, \mathsf{Val})$, and we obtain the Kripke logical relation $R = \mathit{fix}(\Phi)$ satisfying $R = \Phi(R, R)$ as required.

Theorem 1 (Existence, [18]). *The functional Φ has a unique fixed point.*

5.2 The Basic Lemma

By induction on A and the derivation of $A \preceq B$, resp., the following monotonicity properties are established:

Lemma 2 (Kripke Monotonicity). *Suppose $\langle a, u\rangle \in R^A_w$ and $w' \geq w$. Then $\langle [\![A]\!]^w_w\ (a), u\rangle \in R^A_w$.*

Lemma 3 (Subtype Monotonicity). *Let $w \in \mathcal{W}$, $A \preceq B$ and $\langle a, u\rangle \in R^A_w$. Then $\langle [\![A \preceq B]\!]_w\,(a), u\rangle \in R^B_w$.*

Lemmas 2 and 3 show a key property of the relation R, which is at the heart of the coherence proof: For $\langle a, u\rangle \in R^A_w$ we can apply coercions to a and enlarge the world w while remaining in relation with $u \in \mathsf{Val}$.

We extend R to contexts Γ in the natural way. It is not hard to prove the fundamental property of logical relations which says that the (typed and untyped) denotations of well-typed terms compute related results.

Table 4. Bracketing maps

$\phi_w^{\mathsf{bool}}(b)$	$=$	b
$\psi_w^{\mathsf{bool}}(v)$	$=$	$\begin{cases} v & \text{if } v \in \mathsf{BVal} \\ \text{undefined} & \text{otherwise} \end{cases}$
$\phi_w^{\{\mathsf{m}_i:A_i\}}(r)$	$=$	$\{\!\vert \mathsf{m}_i = \phi_w^{A_i}(r.\mathsf{m}_i) \vert\!\}$
$\psi_w^{\{\mathsf{m}_i:A_i\}}(v)$	$=$	$\begin{cases} \{\!\vert \mathsf{m}_i = \psi_w^{A_i}(v.\mathsf{m}_i) \vert\!\} & \text{if } v \in \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) \text{ and } \psi_w^{A_i}(v.\mathsf{m}_i)\downarrow \text{ for all } i \\ \text{undefined} & \text{otherwise} \end{cases}$
$\phi_w^{A \Rightarrow B}(f)$	$=$	$\lambda\langle\sigma, v\rangle. \begin{cases} \langle \phi_w^{St}\ (s), \phi_w^{B}\ (b)\rangle & \text{if } \mathsf{dom}(\sigma) = w' \in \mathcal{W}, \psi_w^{St}\ (\sigma)\downarrow, \psi_w^{A}\ (v)\downarrow \\ & \text{and } f_w\ (\psi_w^{St}\ (\sigma), \psi_w^{A}\ (v)) = \langle w'', \langle s, b\rangle\rangle \\ \text{undefined} & \text{otherwise} \end{cases}$
$\psi_w^{A \Rightarrow B}(g)$	$=$	$\lambda_{w\ \geq w}\, \lambda\langle s, a\rangle. \begin{cases} \langle w'', \langle \psi_w^{St}\ (\sigma), \psi_w^{B}\ (v)\rangle\rangle & \text{if } g(\phi_w^{St}\ (s), \phi_w^{A}\ (a)) = \langle \sigma, v\rangle\downarrow \\ & \mathsf{dom}(\sigma) = w'' \in \mathcal{W}, \\ & \psi_w^{St}\ (\sigma)\downarrow \text{ and } \psi_w^{B}\ (v)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$
$\phi_w^{\mathsf{ref}\ A}(l)$	$=$	l
$\psi_w^{\mathsf{ref}\ A}(v)$	$=$	$\begin{cases} v & \text{if } v \in \mathsf{Loc}_A \\ \text{undefined} & \text{otherwise} \end{cases}$
$\phi_w^{St}(s)$	$=$	$\{\!\vert l_A = \phi_w^{A}(s.l_A) \vert\!\}_{l_A \in w}$
$\psi_w^{St}(\sigma)$	$=$	$\begin{cases} \{\!\vert l_A = \psi_w^{A}(\sigma.l_A) \vert\!\}_{l_A \in w} & \text{if } \psi_w^{A}(\sigma.l_A)\downarrow \text{ for all } l_A \in w \\ \text{undefined} & \text{otherwise} \end{cases}$

Lemma 4 (Basic Lemma). *Suppose $\Gamma \triangleright e : A$, $w \in \mathcal{W}$, $\langle \rho, \eta\rangle \in R_w^{\Gamma}$ and $\langle s, \sigma\rangle \in R_w^{St}$. Then*

- *either $[\![\Gamma \triangleright e : A]\!]_w\, \rho s\uparrow$ and $[\![e]\!]\, \eta\sigma\uparrow$, or*
- *there are $w' \geq w, s', x', \sigma', y'$ s.t.$[\![\Gamma \triangleright e : A]\!]_w\, \rho s = \langle w', \langle s', a\rangle\rangle$ and $[\![e]\!]\, \eta\sigma = \langle \sigma', u\rangle$ s.t. $\langle s', \sigma'\rangle \in R_w^{St}$ and $\langle a, u\rangle \in R_w^{A}$.*

Proof. By induction on the derivation of $\Gamma \triangleright e : A$, using Lemmas 2 and 3. $\square$

5.3 Bracketing

Next, in Table 4, we define families of "bracketing" maps ϕ_w, ψ_w,

$$[\![A]\!]_w \underset{\psi_w^A}{\overset{\phi_w^A}{\rightleftarrows}} \mathsf{Val} \qquad \text{and} \qquad S_w \underset{\psi_w^{St}}{\overset{\phi_w^{St}}{\rightleftarrows}} \mathsf{St}$$

such that $\psi_w^A \circ \phi_w^A = \mathsf{id}_{[\![A]\!]_w}$, i.e., each $[\![A]\!]_w$ is a retract of the untyped model.

As in [22], the retraction property follows from a more general result which justifies the term "bracketing", $\phi_w^A \subseteq R_w^A \subseteq (\psi_w^A)^{op}$, relating the (graphs of the) bracketing maps and the Kripke logical relation of the previous section.

Theorem 2 (Bracketing). *For all $w \in \mathcal{W}$ and $A \in$ Type,*

- *$\forall x \in [\![A]\!]_w\,.\ \langle x, \phi_w^A(x)\rangle \in R_w^A$;*

– $\forall s \in S_w.\ \langle s, \phi^{St}_w(s)\rangle \in R^{St}_w$;
– $\forall \langle x, y\rangle \in R^A_w.\ x = \psi^A_w(y)$; *and*
– $\forall \langle s, \sigma\rangle \in R^{St}_w.\ s = \psi^{St}_w(\sigma)$.

Proof (Sketch). Compared to Reynolds work, the proof of the theorem is more involved, again due to the (mixed-variant) type recursion caused by higher-order store. By a simultaneous induction on n we first prove the properties

– $\forall x \in [\![A]\!]_w.\ \pi^{Aw}_n(x)\downarrow \implies \langle \pi^{Aw}_n(x), \phi^A_w(\pi^{Aw}_n(x))\rangle \in R^A_w$
– $\forall \langle x, y\rangle \in R^A_w.\ \pi^{Aw}_n(x)\downarrow \implies \pi^{Aw}_n(x) = \pi^{Aw}_n(\psi^A_w(y))$

for all $n \in \mathbb{N}$, using the projection maps that come with the minimal invariant solution D of the endofunctor F on $\mathcal{C}$: For $\delta(e) = F(e, e)$ we set $\pi^{Aw}_n \overset{def}{=} \delta^n(\bot)_{Aw}$, and similarly $\pi^{Sw}_n \overset{def}{=} \delta^n(\bot)_{Sw}$. By definition of the minimal invariant solution, $\bigsqcup_n \pi^{Aw}_n = (\bigsqcup_n \delta^n(\bot))_{Aw} = (\mathsf{lfp}(\delta))_{Aw} = \mathsf{id}_{Aw}$ follows. Also, $\bigsqcup_n \pi^{Sw}_n = \mathsf{id}_{Sw}$.

Now for the first part of the theorem let $x \in [\![A]\!]_w$. Thus, $x = \bigsqcup_n \pi^{Aw}_n(x)$ entails $\pi^{Aw}_n(x)\downarrow$ for n sufficiently large. By the above, $\langle \pi^{Aw}_n(x), \phi^A_w(\pi^{Aw}_n(x))\rangle \in R^A_w$ for all sufficiently large $n \in \mathbb{N}$. This is a countable chain in $[\![A]\!]_w \times \mathsf{Val}$, and admissibility of R^A_w and continuity of ϕ^A_w prove the result. The other parts are similar. □

6 Coherence of the Intrinsic Semantics

We have now all the parts assembled in order to prove coherence (which proceeds exactly as in [22]): Suppose $\mathcal{P}_1(\Gamma \rhd e : A)$ and $\mathcal{P}_2(\Gamma \rhd e : A)$ are derivations of the judgement $\Gamma \rhd e : A$. We show that their semantics agree. Let $w \in \mathcal{W}$, $\rho \in [\![\Gamma]\!]_w$ and $s \in S_w$. By Theorem 2 parts (1) and (2), $\langle \rho, \phi^\Gamma_w(\rho)\rangle \in R^\Gamma_w$ and $\langle s, \phi^{St}_w(s)\rangle \in R^{St}_w$. Hence, by two applications of the Basic Lemma, either

$$[\![\mathcal{P}_1(\Gamma \rhd e : A)]\!]_w\, \rho s\uparrow \ \wedge\ [\![e]\!]\,(\phi^\Gamma_w(\rho))(\phi^{St}_w(s))\uparrow \ \wedge\ [\![\mathcal{P}_2(\Gamma \rhd e : A)]\!]_w\, \rho s\uparrow$$

or else there exist $w_i \geq w$, $s_i \in S_{w_i}$, $v_i \in [\![A]\!]_{w_i}$ and $\sigma \in \mathsf{St}$, $v \in \mathsf{Val}$ such that

$$\begin{aligned}&[\![\mathcal{P}_1(\Gamma \rhd e : A)]\!]_w\, \rho s = \langle w_1, \langle s_1, v_1\rangle\rangle \ \wedge\ [\![e]\!]\,(\phi^\Gamma_w(\rho))(\phi^{St}_w(s)) = \langle \sigma, v\rangle \\ &\wedge\ [\![\mathcal{P}_2(\Gamma \rhd e : A)]\!]_w\, \rho s = \langle w_2, \langle s_2, v_2\rangle\rangle\end{aligned}$$

where $\langle s_i, \sigma\rangle \in R^{St}_{w_i}$ and $\langle v_i, v\rangle \in R^A_{w_i}$, for $i = 1, 2$. The definition of the relation $R^{St}_{w_i}$ entails $w_1 = \mathsf{dom}(\sigma) = w_2$, and by Theorem 2 parts (3) and (4), $s_1 = \psi^{St}_{w_1}(\sigma) = \psi^{St}_{w_2}(\sigma) = s_2$ and $v_1 = \psi^A_{w_1}(v) = \psi^A_{w_2}(v) = v_2$. Thus we have shown

Theorem 3 (Coherence). *All derivations of a judgement $\Gamma \rhd e : A$ have the same meaning in the intrinsic semantics.*

Note that this result does not hold if the type annotation A in new_A was removed. In particular, there would then be two different derivations of the judgement

$$x{:}\{\mathsf{m} : \mathsf{bool}\} \rhd \mathsf{new}\ x; \mathsf{true} : \mathsf{bool} \qquad (5)$$

one without use of subsumption, and one where x is coerced to type $\mathbf{1}$ before allocation. The denotations of these two derivations are *different* (clearly not even the resulting extended worlds are equal). It could be argued that, at least in this particular case, this is a defect of the underlying model: The use of a global store does not reflect the fact that the cell allocated in (5) above remains *local* and cannot be accessed by any enclosing program. However, in the general case we do not know if the lack of locality is the only reason preventing coherence for terms without type annotations.

7 Discussion

We consider some aspects in more detail. Firstly, the technical development so far can be used to obtain an (extrinsic) semantics over the untyped model, based on partial equivalence relations. Secondly, we show that our simple notion of subtyping is useful in obtaining a pleasingly straightforward semantics of the object calculus [1]. Finally, we demonstrate how to prove (non-trivial) properties of programs using higher-order store in the model: We consider an object-oriented, "circular" implementation of the factorial function.

7.1 Extrinsic PER Semantics

Apart from proving coherence, Reynolds used (his analogue of) Theorem 2 to develop an *extrinsic* semantics of types in the language [22]. Besides Theorem 2 this only depends on the Basic Lemma, and we can do exactly the same here. More precisely, the binary relation $||A||_w$, defined as $(R^A_w)^{op} \circ R^A_w$, is a partial equivalence relation (per) on $\mathsf{Val} \times \mathsf{Val}$. We observe that a direct proof of transitivity is non-trivial, but it follows easily with part (3) of Theorem 2.

This definition induces a per $||w|| \subseteq \mathsf{St} \times \mathsf{St}$ for every $w \in \mathcal{W}$ by $\langle \sigma, \sigma' \rangle \in ||w||$ iff $\mathsf{dom}(\sigma) = w = \mathsf{dom}(\sigma')$ and $\langle \sigma.l_A, \sigma'.l_A \rangle \in ||A||_w$ for all $l_A \in w$. The Basic Lemma then shows that the semantics is well-defined on $||-||$-equivalence classes, in the sense that if $\Gamma \triangleright e : A$ then for all $w \in \mathcal{W}$, for all $\langle \eta, \eta' \rangle \in ||\Gamma||_w$ and all $\langle \sigma, \sigma' \rangle \in ||w||$,

$$[\![e]\!]\,\eta\sigma\!\downarrow \;\vee\; [\![e]\!]\,\eta'\sigma'\!\downarrow \implies \begin{cases} [\![e]\!]\,\eta\sigma = \langle \sigma_1, u \rangle \;\wedge\; [\![e]\!]\,\eta'\sigma' = \langle \sigma_1', u' \rangle \;\wedge \\ \exists w' \geq w.\ \langle \sigma_1, \sigma_1' \rangle \in ||w'|| \;\wedge\; \langle u, u' \rangle \in ||A||_w \end{cases} \tag{6}$$

The resulting per model satisfies some of the expected typed equations: For instance, $\{\!|\mathsf{m} = \mathit{true}, \mathsf{m}' = \mathit{true}|\!\}$ and $\{\!|\mathsf{m} = \mathit{true}, \mathsf{m}' = \mathit{false}|\!\}$ are equal at $\{\mathsf{m} : \mathsf{bool}\}$. Unfortunately, no non-trivial equations involving store are valid in this model; in particular, locality and information hiding are not captured. This is no surprise since we work with a global store, and the failure of various desirable equations has already been observed for the underlying typed model [14].

However, locality is a fundamental assumption underlying many reasoning principles about programs, such as object and class invariants in object-oriented programming. The work of Reddy and Yang [19], and Benton and Leperchey [6], shows how more useful equivalences can be built in into typed models of languages with storable references. It would be interesting to investigate if these ideas carry over to full higher-order store.

We remark that, unusually, the per semantics sketched above does not seem to work over a "completely untyped" partial combinatory algebra: The construction relies on the partition of the location set $\mathsf{Loc} = \bigcup_A \mathsf{Loc}_A$. In particular, the definition of the pers depends on this rather arbitrary partition. The amount of type information retained by using typed locations allows to express the invariance required for references in the presence of subtyping. We have been unable to find a more "semantic" condition.

Further, it is interesting to observe the role the typed "witness" of $\langle x_1, x_2\rangle \in ||A||_w$ play, i.e., the unique element $a \in [\![A]\!]_w$ with $\langle a, x_i\rangle \in R^A_w$: Crucially, a determines the world $w' \geq w$ over which the result store and value are to be interpreted in the case of application.

Previously we have given a denotational semantics for a logic of objects [3], where an untyped cpo model was used [20]. This logic has a built-in notion of invariance which makes it very similar to a type system, and the semantic structure of function types used in [20] closely resembles (6). In fact, in [20] an ad-hoc construction was necessary to "determinise" the existential quantification over world extensions of (6) in order to preserve admissibility of predicates (corresponding to types and specifications of the logic). Regarding the setting of the present paper, the tracking of the computation on $\mathcal{W}$ is hard-wired into the witnesses coming from the typed model.

7.2 A Semantics of Objects

Next, we sketch how to give a semantics to Abadi and Cardelli's imperative object calculus with first-order types [1], where we distinguish between fields and methods (with parameters). Fields are mutable, but methods cannot be updated. The type of objects with fields f_i of type A_i and methods m_j of type C_j (with self parameter y_j) and parameter z_j of type B_j, is written $[\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j}$. The introduction rule is

$$\frac{\begin{array}{c}A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j}\\ \Gamma \vdash x_i : A_i \;\forall i \quad \Gamma, y_j{:}A, z_j{:}B_j \vdash b_j : C_j \;\forall j\end{array}}{\Gamma \vdash [\mathsf{f}_i = x_i, \mathsf{m}_j = \varsigma(y_j)\lambda z_j.\, b_j]_{i,j} : A} \tag{7}$$

Subtyping on objects is by width, and for methods also by depth:

$$\frac{B_j{\Rightarrow}C_j \preceq B'_j{\Rightarrow}C'_j \;\forall j \in J' \quad I' \subseteq I \quad J' \subseteq J}{[\mathsf{f}_i : A_i, \mathsf{m}_j : B_j \Rightarrow C_j]_{i\in I, j\in J} \preceq [\mathsf{f}_i : A_i, \mathsf{m}_j : B'_j \Rightarrow C'_j]_{i\in I\;', j\in J}} \tag{8}$$

The following is essentially a (syntactic) presentation of the fixed-point (or *closure*) model of objects [11], albeit in a typed setting: Objects of type $A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j}$ are simply interpreted as *records* of the corresponding record type $A^* \equiv \{\mathsf{f}_i{:}\mathsf{ref}\ A^*_i, \mathsf{m}_j{:}B^*_j{\Rightarrow}C^*_j\}_{i,j}$. Note that the self parameter does not play any part in this type (in contrast to functional interpretations of objects, cf. [8]), and soundness of (8) follows directly from the rules of Sect. 2.

A new object $[\mathsf{f}_i{=}x_i, \mathsf{m}_j{=}\varsigma(y_j)\lambda z_j.\, b_j]_{i,j}$ of type A is created by allocating a state record s and defining the methods by mutual recursion (using obvious syntax sugar),

$$\mathsf{let}\ s = \{\mathsf{f}_i = \mathsf{new}_{A_i}(x_i)\}_{i\in I}\ \mathsf{in}\ \mathit{Meth}_A(s)(\{\mathsf{m}_j = \lambda y_j \lambda z_j.\, b_j\}_{j\in J})$$

where $Meth_A : \{\mathsf{f}_i{:}\mathsf{ref}\ A_i\}_{i\in I} \Rightarrow \{\mathsf{m}_j{:}A^*{\Rightarrow}B_j{\Rightarrow}C_j\}_{j\in J} \Rightarrow A^*$ is given by

$$Meth_A \equiv \mu f(s).\lambda m.\ \{\mathsf{f}_i = s.\mathsf{f}_i, \mathsf{m}_j = \lambda z_j.\,(m.\mathsf{m}_j(f(s)(m)))(z_j)\}_{i\in I, j\in J}$$

Here $\mu f(x).e$ is a recursively defined function f; note that this is meaningful: Using the fixed-point of the map $h \mapsto [\![\lambda x.e]\!]_w\, \rho[f := h]$ in **Cpo** recursive functions can be interpreted in the model. The Basic Lemma holds also for the language extended with recursive functions [23]. Soundness of (7) follows immediately from this interpretation of objects and object types.

7.3 Reasoning About Higher-Order Store and Objects

In the following program let $A \equiv [\mathsf{fac} : \mathsf{int} \Rightarrow \mathsf{int}]$, and $B \equiv [\mathsf{f} : A, \mathsf{fac} : \mathsf{int} \Rightarrow \mathsf{int}]$ (so $B \preceq A$). The program computes the factorial, making the recursive calls through the store.

```
let $a : A = [\mathsf{fac} = \varsigma(x)\lambda n.\, n]$
let $b : B = [\mathsf{f} = a, \mathsf{fac} = \varsigma(x)\lambda n.$ if $n < 1$ then $1$ else $n \times (x.\mathsf{f}.\mathsf{fac}(n-1))]$
in $b.\mathsf{f} := b$; $b.\mathsf{fac}(x)$
```

While we certainly do not claim that this is a particularly realistic example, it does show how higher-order store complicates reasoning. We illustrate a pattern for dealing with the self-application arising from higher-order store, following the general ideas of [21]: To prove that the call in the last line indeed computes the factorial of x, consider the family of predicates $P = (P_w)_w$ where w ranges over worlds $\geq \{l{:}A\}$ and $P_w \subseteq [\![\mathsf{int} \Rightarrow \mathsf{int}]\!]_w$,

$$\begin{aligned} h \in P_w \quad \overset{def}{\Longleftrightarrow} \quad & \forall w' \geq w\, \forall s \in S_w\ \forall n \in [\![\mathsf{int}]\!]_w\ .\ (s.l.\mathsf{fac} \in P_w \ \wedge\ n \geq 0\ \wedge\ h_w\ (s,n)\!\downarrow) \\ & \Longrightarrow\ \exists w'' \geq w'\ \exists s' \in S_w\ .\ h_w\ (s,n) = \langle w'', \langle s', n!\rangle\rangle \end{aligned}$$

Note that P_w corresponds to a partial correctness assertion, i.e., *if* the result is defined, then it is indeed $n!$. This example has also been considered in the context of total correctness, in recent work of Honda *et al.* [9] (where, rather different to here, the proof relies on well-founded induction using a termination order).

Existence of P is established along the lines of Theorem 1. Then, assuming that l is the location allocated for field f, a simple fixed-point induction shows

$$[\![x{:}\mathsf{int}, a : A \rhd [\mathsf{f} = a, \mathsf{fac} = \varsigma(x)\lambda n.\ \ldots\ ] : B]\!]_w\, \rho s = \langle w', \langle s', o\rangle\rangle$$

such that w' is $w \cup \{l{:}A\}$, and $o.\mathsf{fac} \in P_w$. Now let $\hat{s} = s'[l := [\![B \preceq A]\!]_w\ (o)]$. Thus, $\hat{s}.l.\mathsf{fac} = o.\mathsf{fac} \in P_w$; and if $\rho(x) \geq 0$ we conclude

$$\begin{aligned} & [\![x{:}\mathsf{int}, a{:}A, b{:}[\mathsf{f}{:}A,\ \mathsf{fac}{:}\mathsf{int}{\Rightarrow}\mathsf{int}] \rhd b.\mathsf{f} := b; b.\mathsf{fac}(x) : \mathsf{int}]\!]_w\ \ \rho[b := o]\hat{s} \\ & = \hat{s}.l.\mathsf{fac}_w\ (\hat{s}, \rho(x)) \\ & = \langle w'', \langle s'', \rho(x)!\rangle\rangle \end{aligned}$$

for some $w'' \in \mathcal{W}$ and $s'' \in S_w$.

8 Related Work

Possible worlds models of programming languages were first considered in the work of Reynolds and Oles on the semantics of local stack-allocated variables [17]. The current work is closer in spirit to the various possible worlds models for languages with dynamic allocation of *heap* storage [14,19,25,6].

Apart from Levy's work [14,15] which we built upon here, we are aware of only few other semantic models of higher-order store in the literature. The models [4,12] use games semantics and are not location-based, i.e., the store is modelled only indirectly via possible program behaviours. They do not appear to give rise to reasoning principles such as those necessary to establish the existence of the logical relation, or the predicate used in Sect. 7.3. Ahmed, Appel and Virga [5] construct a model with a rather operational flavour: The semantics of types is obtained by approximating absence of type errors in a reduction semantics; soundness of this construction follows from an encoding into type theory. Again we do not see how strong reasoning principles can be obtained. Jeffrey and Rathke [10] provide a model of the object calculus in terms of interaction traces, very much in the spirit of games semantics. Apart from Jeffrey and Rathke's semantics, none of these models deals with subtyping.

The proof principles applied in Sect. 7.3 are direct adaptations of those presented in [21] in the context of an untyped model of the object calculus.

9 Conclusions and Future Work

We have extended a model of general references with subtyping, to obtain a semantics of imperative objects. While the individual facts are much more intricate to prove than for the functional language considered in [22], the overall structure of the coherence proof is almost identical to *loc.cit.* It could be interesting to work out the general conditions needed for the construction.

In a different direction, we can extend the language with a more expressive type system: Recursive types and polymorphism feature prominently in the work on semantics of *functional* objects (see [8]). In [15] it is suggested that the construction of the intrinsic model also works for a variant of recursive types. We haven't considered the combination with subtyping yet, but do not expect any difficulties. In fact, also the extension with ML-like (prenex) polymorphism is straightforward – essentially because there is no interaction with the store.

Finally, we plan to develop (Hoare-style) logics, with pre- and post-conditions, for languages involving higher-order store. As a starting point, we would like to adapt the program logic of [3] to the language considered here.

Acknowledgement. I wish to thank Bernhard Reus for many helpful discussions. Paul Levy pointed out a flaw in an earlier version. Financial support was provided by EPSRC grant GR/R65190/01, "Programming Logics for Denotations of Recursive Objects"

References

1. M. Abadi and L. Cardelli. *A Theory of Objects.* Springer, 1996.
2. M. Abadi, L. Cardelli, and R. Viswanathan. An interpretation of objects and object types. In *Proc. POPL'96*, pages 396–409. 1996.
3. M. Abadi and K. R. M. Leino. A logic of object-oriented programs. In *Verification: Theory and Practice. Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, LNCS, pages 11–41. Springer, 2004.
4. S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc. LICS'98*, pages 334–344. 1998.
5. A. J. Ahmed, A. W. Appel, and R. Virga. A stratified semantics of general references embeddable in higher-order logic. In *Proc. LICS'02*, pages 75–86. 2002.
6. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. TLCA'05*, volume 3461 of LNCS, pages 86–101. 2005.
7. V. Breazu-Tannen, T. Coquand, G. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.
8. K. B. Bruce, L. Cardelli, and B. C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, Nov. 1999.
9. K. Honda, M. Berger, and N. Yoshida. An observationally complete program logic for imperative higher-order functions. To appear in *Proc. LICS'05*, 2005.
10. A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. In *Proc. LICS'02*, pages 101–112. 2002.
11. S. N. Kamin and U. S. Reddy. Two semantic models of object-oriented languages. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, pages 464–495. MIT Press, 1994.
12. J. Laird. A categorical semantics of higher-order store. In *Proc. CTCS'02*, volume 69 of *ENTCS*, pages 1–18. 2003.
13. P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4):308–320, Jan. 1964.
14. P. B. Levy. Possible world semantics for general storage in call-by-value. In *Proc. CSL'02*, volume 2471 of *LNCS*. 2002.
15. P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*, volume 2 of *Semantic Structures in Computation.* Kluwer, 2004.
16. J. C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51(1–2):99–124, 1991.
17. F. J. Oles. *A Category-theoretic approach to the semantics of programming languages.* PhD thesis, Syracuse University, 1982.
18. A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
19. U. S. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1–3):129–160, March 2004.
20. B. Reus and J. Schwinghammer. Denotational semantics for Abadi and Leino's logic of objects. In *Proc. ESOP'05*, volume 3444 of *LNCS*, pages 264–279. 2005.
21. B. Reus and T. Streicher. Semantics and logic of object calculi. *Theoretical Computer Science*, 316:191–213, 2004.
22. J. C. Reynolds. What do types mean? — From intrinsic to extrinsic semantics. In *Essays on Programming Methodology.* Springer, 2002.

23. J. Schwinghammer. A typed semantics for languages with higher-order store and subtyping. Technical Report 2005:05, Informatics, University of Sussex, 2005.
24. M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, Nov. 1982.
25. I. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998.

Two Variables Are Not Enough

Rick Statman

Carnegie Mellon University, Dept. of Mathematical Sciences, Pittsburgh, PA 15213
statman@cs.cmu.edu

Abstract. Let n be the smallest integer such that every closed lambda term beta converts to one with at most n bound variables. We show that $n = 3$.

We shall work in the untyped lambda calculus. For the most part we shall adopt the notation and terminology of [1].

It is well known that every closed lambda term beta converts to one with only three distinct bound variables. This can be seen by converting each closed lambda term into an applicative combination of S's and K's. It is equally clear that not every closed term can be beta converted into one with only one bound variable; since, such terms are closed under beta reduction (without the aid of alpha conversion). Here we settle the case of two variables in the negative.

As a consequence we conclude the following.

It is easy to see the mini-max principle holds (modulo insertions of K); the minimum number of bound variables needed to write a closed term equals the maximum number of free variables in a subterm.Thus any closed term beta convertible to S must contain a subterm with three distinct free variables.The same holds true for B, since B, C^*, K, and W^* form a basis ([1] pg 210).

We shall define a combinatory logic suitable for discussing terms with at most two bound variables. To fix ideas we shall fix these variables to be x, y. The combinatory logic consists of atoms (constants, combinators)

$$A$$

together with reduction rules

$$\rightarrowtail .$$

We call the logic HOT (combinators Hereditarily of Order Two). The atoms and rules are defined by a simultaneous recursion as follows.

If X is an applicative combination of x's and y's then the HOT atom A is introduced with the reduction rule

$$Axy \rightarrowtail X$$

and the rank of A is 1.

If X is an applicative combination of x's, y's, and HOT atoms of ranks $< n$ then the HOT atom A is introduced with the reduction rule

$$Axy \rightarrowtail X$$

and the rank of A is n.

For convenience below we shall assume that HOT atoms of rank n also have rank $n+1$ (as if the above recursive definitions were not given simultaneously).

Each HOT combination X of x's, y's, and atoms corresponds to a lambda term X^* by

$x^* = x$

$y^* = y.$

If A has reduction rule $Axy \rightarrowtail X$ then $A^* = \lambda xy.X^*$

$(XY)* = (X * Y*).$

It will be convenient to use the notation $\twoheadrightarrow$ for beta reduction and $\twoheadleftarrow\!\!\twoheadrightarrow$ for beta conversion of lambda terms.

Suppose that X is a lambda term using only the variables x, y (possibly both free and bound). Then there exists a HOT combination H such that $H^* \twoheadrightarrow X$. Define the lambda term $X+$ by

$x+ = x$
$y+ = y$
$(XY)+ = (X + Y+)$

$$\begin{array}{rll} (\lambda x.X)+ = & (\lambda yx.X+)y & \text{if } y \text{ is free in } X \\ & (\lambda yx.X+)K & \text{if } y \text{ is not free in } X \\ (\lambda y.X)+ = & (\lambda xy.X+)x & \text{if } x \text{ is free in } X \\ & (\lambda xy.X+)K & \text{if } x \text{ is not free in } X \end{array}$$

Then $X+ \twoheadrightarrow X$ and $X+$ alpha converts (by the use of a third variable) to $X\hat{}$ defined by

$x\hat{} = x$
$y\hat{} = y$
$(XY)\hat{} = (X\hat{}Y\hat{})$

$$\begin{array}{rll} (\lambda x.X)\hat{} = & (\lambda xy.[x/y, y/x]X\hat{})y & \text{if } x \text{ is free in } X \\ & (\lambda xy.[x/y, y/x]X\hat{})K & \text{if } y \text{ is not free in } X \\ (\lambda y.X)\hat{} = & (\lambda xy.X\hat{})x & \text{if } x \text{ is free in } X \\ & (\lambda xy.X\hat{})K & \text{if } x \text{ is not free in } X. \end{array}$$

From $X\hat{}$ we can define H as $\#X\hat{}$ where
$\#x = x$
$\#y = y$
$\#(XY) = (\#X\#Y)$
$\#(\lambda xy.X)) = A$ where A has the reduction rule $Axy \rightarrowtail \#X$.

HOT combinators together with $\rightarrowtail$ form a regular left normal combinatory reduction system ([2]) and therefore satisfy the Church-Rosser theorem and standardization theorem. $>$ l$\twoheadrightarrow$ stands for leftmost- outermost reduction ([1] pg. 323). $>$ head$\twoheadrightarrow$ stands for head reduction ([1] pg 169).

A closed HOT combination M is said to be "head secure" if there is a combination Y containing y such that $Mxy \rightarrowtail\!\!\!\twoheadrightarrow xY$. The combination X is said to "unroll" to Y if there are HOT combinators (atoms) $A_1, ..., A_n$ (here we allow $n = 0$) such that $X >$ l$\twoheadrightarrow$ $A_1(...(A_nY)......)$. Such Y are clearly closed under head reduction.

Lemma: If M is a head secure combination of HOT combinators of rank $< n+1$ and $Mxy >$ head$\twoheadrightarrow$ X then X is a combination of y's, HOT combinators of rank $< n$ and Y's such that Mx unrolls to Y.

Proof: By induction on the length of a head reduction to X. For the basis case Mx unrolls to Mx. Now suppose that Mxy $>$ head$\twoheadrightarrow$ X1 $>$ head$\rightarrow$ $X2$ and the conclusion is true of $X1$. We distinguish two cases.

Case 1: $X1 = YY1 \ldots Ym$ where Mx unrolls to Y. Since Mxy is head secure Y is not an atom by Church-Rosser. If Y begins with a head redex then the conclusion holds for $X2$. Otherwise $Y = AZ$ and by definition Mx unrolls to Z. Moreover if A' appears on the r.h.s of the reduction rule for A then A' has rank $< n$. Thus in this case $X2$ satisfies the lemma.

Case 2: $X1 = AY1...Ym$ where rank $A < n$. As above $X2$ satisfies the lemma.

Corollary: If M is head secure then Mx has a normal form $A1(...(Anx)...)$.

Proof: $Mxy >$ head$\twoheadrightarrow$ xY where y belongs to Y. Since Mx does not unroll to xY it must unroll to x. Define Hn by

$H0xy \rightarrowtail x(yy)$

$H(n+1)xy \rightarrowtail x(y(Hn))$.

Proposition: There is no combination M of HOT combinators of rank $< n+1$ such that $M^* \twoheadleftarrow\!\!\!\twoheadrightarrow Hn^*$.

Proof: By induction on n, the basis case when $n = 0$ is trivial by definition. Suppose that $n > 0$ and that $M^* \twoheadleftarrow\!\!\!\twoheadrightarrow Hn^*$. Then $Mxy \rightarrowtail\!\!\!\twoheadrightarrow x(yH(n-1)^*)$ and by the standardization theorem there exists a reduction

$$\begin{aligned} &M^*xy > \text{head reduction}\twoheadrightarrow xX \\ &X > \text{head reduction}\twoheadrightarrow xY \\ &Y > \text{standard}\twoheadrightarrow H(n-1)^*. \end{aligned}$$

Thus we have

$$\begin{aligned} Mxy &> \text{head reduction} \twoheadrightarrow x\#X \\ \#X &> \text{head reduction} \twoheadrightarrow y\#Y. \end{aligned}$$

In particular, M is head secure so Mx has a normal form

$$A1(\ldots(Akx)\ldots).$$

We have then by Church-Rosser and standardization

$$\begin{aligned} (@)\ A1(\ldots(Akx)\ldots)y &> \text{head} \twoheadrightarrow xZ \\ Z &> \text{head} \twoheadrightarrow yW. \end{aligned}$$

where $W^* \twoheadrightarrow H(n-1)*$. Now W is a combination of HOT combinators of rank $< n, y$'s and terms $Ai(\ldots(Akx)...)$ for $i = 1, ..., k+1$. Put $N = [OMEGA/x, OMEGA/y]W$. We distinguish two cases.

Case 1: $n = 1$. Then $Nxy \rightarrowtail\!\!\twoheadrightarrow x(yy)$.
Case 2: $n > 1$. Then $Nxy \rightarrowtail\!\!\twoheadrightarrow xX1 \rightarrowtail\!\!\twoheadrightarrow x(yY1))$ where $Y1^* \rightarrowtail\!\!\twoheadrightarrow H(n-2)^*$. In either case N is head secure so Nx has a normal form

$$A'1(...(A'mx)...).$$

Now in case 1, each HOT combinator in N appears in function position iteratively applied to OMEGA. Thus we have that $m = 0$ and the reduction $Nxy \rightarrowtail\!\!\twoheadrightarrow x(yy)$ is impossible. In the second case, by similar reasoning, rank $(A'j) < n$ for $j = 1, ..., m$. By Church-Rosser and standardization there exists a reduction

$$\begin{aligned} A'1(\ldots(A'mx)\ldots)\ y &> \text{head} \twoheadrightarrow xX2 \\ X2 &> \text{head} \twoheadrightarrow yY2 \end{aligned}$$

where $Y2 * H(n-2)*$. This reproduces (@) at lower rank.

Corollary: There is no HOT combination M such that $M^* \twoheadleftarrow\!\!\twoheadrightarrow S$ (or B).

References

[1] Barendregt, The Lambda Calculus, North Holland 1984.
[2] Klop, Combinatory Reduction Systems Math. Centrum Amsterdam 1980.
[3] Statman, Combinators Hereditarily of Order Two CMU Math. Dept. Technical Report 88-33, August 1988.

Author Index