Josep Diaz
Ivan Lanese
Davide Sangiorgi (Eds.)

LNCS 8705

Theoretical Computer Science

8th IFIP TC 1/WG 2.2 International Conference, TCS 2014
Rome, Italy, September 1–3, 2014
Proceedings

ifip

Lecture Notes in Computer Science 8705

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

Josep Diaz Ivan Lanese Davide Sangiorgi (Eds.)

Theoretical Computer Science

8th IFIP TC 1/WG 2.2 International Conference, TCS 2014
Rome, Italy, September 1-3, 2014
Proceedings

Springer

Volume Editors

Josep Diaz
Universitat Politecnica de Catalunya
Departament de Llenguatges i Sistemes Informatics
08034 Barcelona, Spain
E-mail: diaz@lsi.upc.edu

Ivan Lanese
Davide Sangiorgi
Università di Bologna
Dipartimento di Informatica - Scienza e Ingegneria (DISI)
Bologna, Italy
E-mail:{ivan.lanese, davide.sangiorgi}@gmail.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-662-44601-0 e-ISBN 978-3-662-44602-7
DOI 10.1007/978-3-662-44602-7
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014946241

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the 26 contributed papers and the abstracts of the two invited lectures presented at the 8th IFIP Theoretical Computer Science (TCS-2014), held in Rome, during September 1–3, 2014. The two invited speakers were Giuseppe Italiano and Jane Hillston.

Previous TCS conferences took place in Amsterdam 2012, Brisbane 2010, Milan 2008, Santiago 2006, Tolouse 2004, Montreal 2002, and Sendai 2000.

TCS-2014 consisted of two tracks, with separate Program Committees, which dealt with:

- Track A: Algorithms, Complexity, and Models of Computation
- Track B: Logic, Semantics, Specification, and Verification

The Program Committees for track A and track B are listed in the next section.

Track A selected 12 out of 38 submissions and track B selected 14 out of 35 submissions.

IFIP TCS 2014 was co-located with CONCUR 2014. We would like to thank the general chairs of the whole event, Daniele Gorla and Rossella Petreschi, as well as the members of the TCS Steering Committee, Giorgio Ausiello, Jos Baeten, and Jacques Sakarovitch for their help and support.

July 2014

Josep Diaz
Ivan Lanese
Davide Sangiorgi

Organization

Program Committee from Track A Algorithms, Complexity and Models of Computation

Cristian Calude	University of Auckland, New Zealand
Josep Diaz	UPC Barcelona, Spain
Irene Finocchi	University of Rome La Sapienza, Italy
Paola Flocchini	University of Ottawa, Canada
Aristides Gionis	Aalto University, Finland
Mordecai Golin	Hong Kong University of Science and Technology, SAR China
Juhani Karhumäki	University of Turku, Finland
Elvira Mayordomo	Universidad de Zaragoza, Spain
Massimo Melucci	University of Padua, Italy
Friedhelm Meyer-Auf-Der-Heide	University of Paderborn, Germany
Pekka Orponen	Aalto University, Finland
Giuseppe Persiano	Università di Salerno, Italy
Geraud Senizergues	Université Bordeaux, France
Olivier Serre	CNRS, France
Paul Spirakis	University of Liverpool, UK
Osamu Watanabe	Tokyo Institute of Technology, Japan
Peter Widmayer	ETH Zürich, Switzerland

Additional Reviewers from Track A Algorithms, Complexity and Models of Computation

Agnarsson, Geir	Freivalds, Rusins
Benter, Markus	Giotis, Yiannis
Berwanger, Dietmar	Grellois, Charles
Bohmova, Katerina	Hirvensalo, Mika
Brandstädt, Andreas	Hromkovic, Juraj
Calamonieri, Tiziana	Hruz, Tomas
Campeanu, Cezar	Janin, David
Caramia, Massimiliano	Jansen, Bart M.P.
Carton, Olivier	Jeandel, Emmanuel
Courcelle, Bruno	Jurgensen, Helmut
Dinneen, Michael J.	Koutsopoulos, Andreas
Doerr, Carola	Le Gall, Francois
Dregi, Markus	Leroux, Jerome
Fijalkow, Nathanaël	Li, Shouwei

Michail, Othon
Mihalák, Matúš
Minato, Shin-Ichi
Montanari, Sandro
Okamoto, Yoshio
Okhotin, Alexander
Papaioannou, Evi
Pecher, Arnaud
Penna, Paolo
Petreschi, Rossella
Podolskii, Vladimir
Prusa, Daniel
Pröger, Tobias
Raptopoulos, Christoforos
Riechers, Sören
Romashchenko, Andrei
Saarela, Aleksi
Say, A.C. Cem
Segev, Danny
Vocca, Paola
Yamazaki, Koichi

Organization

Program Committee from Track B
Logic, Semantics, Specification and Verification

Jos Baeten	Eindhoven University of Technology, The Netherlands
Lars Birkedal	Aarhus University, Denmark
Andrei Bulatov	Simon Fraser University, Canada
James Cheney	University of Edinburgh, UK
Ugo Dal Lago	University of Bologna/Inria, Italy
Josée Desharnais	Université Laval, Canada
Jörg Endrullis	Vrije Universiteit Amsterdam, The Netherlands
Jérôme Feret	ENS Paris, France
Marcelo Fiore	University of Cambridge, UK
Silvio Ghilardi	Università degli Studi di Milano, Italy
Benjamin Gregoire	Inria, France
Ichiro Hasuo	University of Tokyo, Japan
Marieke Huisman	University of Twente, The Netherlands
Radha Jagadeesan	DePaul University Chicago, USA
Jean Krivine	CNRS, France
Ivan Lanese	University of Bologna/Inria, Italy
Grigore Rosu	University of Illinois at Urbana-Champaign, USA
Davide Sangiorgi	University of Bologna/Inria, Italy
Vladimiro Sassone	University of Southampton, UK
Igor Walukiewicz	Université Bordeaux, France

Additional Reviewers from Track B
Logic, Semantics, Specification and Verification

Abbes, Samy	Ciancia, Vincenzo
Accattoli, Beniamino	Clouston, Ranald
Asada, Kazuyuki	Cristescu, Ioana
Avanzini, Martin	Fournier, Paulin
Beklemishev, Lev	Gauwin, Olivier
Bernardo, Marco	Gebler, Daniel
Bizjak, Aleš	Goldblatt, Robert
Bonsangue, Marcello	Hirai, Yoichi
Bottoni, Paolo	Jansen, Christina

Klein Wolterink, Wouter
Kurz, Alexander
Lluch Lafuente, Alberto
May, Jonathan
Mazza, Damiano
Paulevé, Loïc
Perera, Roly
Pitcher, Corin
Pretnar, Matija
Pérez, Jorge A.
Rehof, Jakob
Riba, Colin
Schubert, Aleksy
Simon, Laurent
Sokolova, Ana
Spicher, Antoine
Strub, Pierre-Yves
Terui, Kazushige
Thiemann, René
Tribastone, Mirco
Urban, Caterina
Van Oostrom, Vincent
Varacca, Daniele
Vignudelli, Valeria
Volpe, Marco

Abstracts

Strong Bridges and Strong Articulation Points of Directed Graphs

Giuseppe F. Italiano

Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università di Roma "Tor Vergata", Roma, Italy

Abstract. Given a directed graph G, an edge is a strong bridge if its removal increases the number of strongly connected components of G. Similarly, a vertex is a strong articulation point if its removal increases the number of strongly connected components of G. Strong articulation points and strong bridges are related to the notion of 2-vertex and 2-edge connectivity of directed graphs, which surprisingly seems to have been overlooked in the past. In this talk, we survey some very recent work in this area, both from the theoretical and the practical viewpoint.

The Benefits of Sometimes Not Being Discrete

Jane Hillston

LFCS, School of Informatics, University of Edinburgh
jane.hillston@ed.ac.uk
http://www.quanticol.eu

Abstract. Discrete representations of systems are usual in theoretical computer science and they have many benefits. Unfortunately they also suffer from the problem of state space explosion, sometimes termed the *curse of dimensionality*. In the area of quantitative modelling, high-level formal modelling formalisms such as stochastic Petri nets and stochastic process algebras have eased the construction of underlying Markovian models. But the combination of improved model construction techniques and the increasing scale and complexity of the systems being developed have exacerbated the problem of state space explosion.

In recent years, research has shown that there are cases in which we can reap the benefits of discrete representation during system description but then gain from more efficient analysis by approximating the discrete system by a continuous one. When the system under consideration can be presented as a *population* model and the populations involved are known to be *large*, then a good approximation of the discrete behaviour can be achieved through a continuous or fluid approximation whereby the discrete Markov chain is replaced by a set of ordinary differential equations. Moreover, this model is scale-free in the sense that the computational effort to solve it remains the same even as the populations involved grow larger.

This paper will motivate this approach, explaining the theoretical foundations and their practical benefits.

Table of Contents

Track A: Algorithms, Complexity and Models of Computation

Track B: Logic, Semantics, Specification and Verification

Zero-Suppressed Binary Decision Diagrams Resilient to Index Faults*

Anna Bernasconi[1] and Valentina Ciriani[2]

[1] Dipartimento di Informatica, Università di Pisa, Italy
anna.bernasconi@unipi.it
[2] Dipartimento di Informatica, Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

Abstract. This paper discusses the error resilience of Zero-Suppressed Binary Decision Diagrams (ZDDs), which are a particular family of Ordered Binary Decision Diagrams used for representing and manipulating combination sets. More precisely, we design a new ZDD canonical form, called index-resilient reduced ZDD, such that a faulty index can be reconstructed in time $O(k)$, where k is the number of nodes with a corrupted index.

1 Introduction

Algorithms and data structures resilient to memory faults [11–13] are able to perform the tasks they were designed for, even in the presence of unreliable or corrupted information. The design of resilient algorithms and data structures is a fundamental issue, as fast, large, and cheap memories in computer platforms are characterized by non-negligible error rates [14].

Several error models have been proposed for designing resilient data structures [22]. A fault model in which any error is detectable via an error message when the program tries to reach the faulty object is proposed in [2]. The authors assume that an error denies access to an entire node of the structure. A model with higher granularity, called *faulty-RAM*, is presented in [9, 10, 13]. In faulty-RAM an adversary can corrupt any memory word and it is impossible to determine a priori if a memory area is corrupted or not. Such a scenario is realistic since an error can be induced by an external source, perhaps temporary, which can change any memory location that can not be discovered a priori. Moreover, faulty-RAM model has an extreme granularity: any memory location (from a single bit, the single data, or an entire structure) can be affected by a fault. Another interesting error model is the *single-component model* described in [22], which focuses on single attributes of an item at a time and assumes that each error affects one component of one node of the storage structure, e.g., a pointer, a count, an identifier field.

* This work was supported in part by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project AMANDA Algorithmics for MAssive and Networked DAta.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 1–12, 2014.

The purpose of this paper is to discuss the error resilience of a data structure called *Zero-Suppressed Binary Decision Diagrams* (ZDDs) [17], that are a particular type of Ordered Binary Decision Diagrams (OBDDs). OBDDs are a fundamental family of data structures for Boolean function representation and manipulation [5]. They have been originally studied for circuit design and formal verification. Recently, the area of application of OBDDs has widened including representation and manipulation of combination sets in different research fields as data mining [17–19], bioinformatics [20, 21, 23], data protection [6]. The growing interest in these data structures is evidenced by the fact that in 2009 Knuth dedicated the first fascicle in the volume 4 of "The Art of Computer Programming" to OBDDs [15].

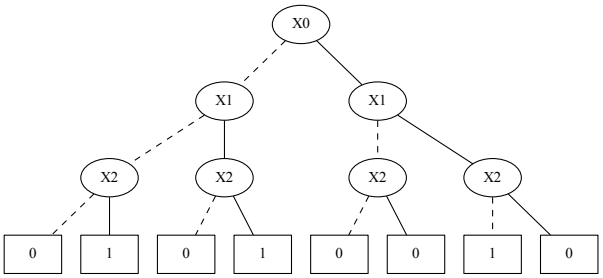
OBDDs are DAG representations of Boolean functions, where each internal node N is labeled by a Boolean variable x_i and has exactly two outgoing edges: 0-edge and 1-edge. Terminal nodes (leaves) are labeled 0 or 1. OBDDs can be constructed by applying some reduction rules to a Binary Decision Tree, and depending on the set of reduction rules, different representations can be derived. For example, Figures 1(b), 1(c), and 1(d) are different decision diagrams, derived by the decision tree in Figure 1(a), representing the same Boolean function. In particular, Reduced OBDDs (ROBDDs) [5] are typically used for the representation of general Boolean functions, while *Zero-Suppressed* BDDs (ZDDs) are used for representing family of subsets of combination sets [15, 17]. Indeed, ZDDs can be used to describe and manipulate solutions to combinatorial problems as, in this framework, they are much more compact than ROBDDs. For instance, the family of subsets $\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_1\}\}$ of the set $\{x_1, x_2, \ldots, x_{10}\}$ needs a ROBDD representation with 10 variables, while the ZDD representation uses only the four variables included in the subsets.

Security aspects of implementation techniques of ROBDDs have been discussed in [7], and an error resilient version has been proposed in [3, 4]. In this paper we study error resilience of ZDDs. We exploit the single-component error model and we assume that errors are reported when the program tries to use the fault component of a node. We consider, as component of a node N in a ZDD, the index i of the variable x_i associated to the node. In particular, we design a ZDD, called *index-resilient reduced ZDD*, such that a faulty index can be reconstruct in time $O(k)$, where k is the number of nodes with a corrupted index. Moreover, the proposed index-resilient ZDD is a canonical form.

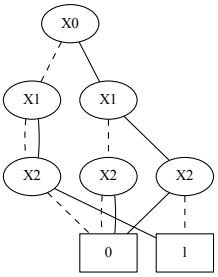
The paper is organized as follows. Preliminaries on OBDDs and ZDDs are described in Section 2. In Section 3 we discuss the error resilience of the standard ZDD structure, and in Section 4 we introduce and study index-resilient ZDDs. Section 5 concludes the paper.

2 OBDDs and Zero-Suppressed BDDs

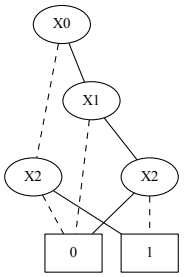
A *Binary Decision Tree* (BDT) on a set of Boolean variables $\{x_0, x_1, \ldots x_{n-1}\}$ is a rooted binary tree, where each non-terminal (internal) node N is labeled by a Boolean variable x_i and has exactly two outgoing edges: 0-edge and 1-edge.



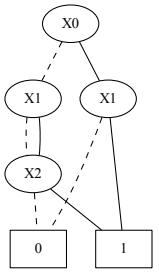
(a) BDT



(b) QR-BDD (m-rule)



(c) ROBDD (m-rule and r-rule)



(d) ZDD (m-rule and z-rule)

Fig. 1. Example of transformations of a BDT using the reduction rules

Terminal nodes (leaves) are labeled 0 or 1 (e.g., see Figure 1(a) where dashed, rep., solid, lines represent 0-edges, resp., 1-edges). Without loss of generality, we can assume that each node containing the variable x_i (with $0 \leq i \leq n-1$) lyes on the i-th level of the tree. Thus, the variable x_0 is the root of the BDT and the leaves are on level n (see for example the BDT in Figure 1(a)).

BDTs are typically used to represent completely specified Boolean functions (i.e., any function $f : \{0,1\}^n \rightarrow \{0,1\}$). The leaves represent the constants 0 and 1 and the root represents the entire Boolean function f. The value of f on the input $x_0, \ldots, x_{n-1}$ is found by following the path indicated in the BDT by the values of $x_0, \ldots, x_{n-1}$ on the edges: the value of $f(x_0, \ldots, x_{n-1})$ is the label of the reached leaf. For example, the BDT in Figure 1(a) represents the Boolean function $f : \{0,1\}^3 \rightarrow \{0,1\}$ such that $f(0,0,0) = 0, f(0,0,1) = 1, \ldots, f(1,1,1) = 0$.

In order to give a more compact description of Boolean functions, a BDT can be compressed in an acyclic graph (called BDD) that represents the same function. In particular, a *Binary Decision Diagram* (BDD) on a set of Boolean

variables $X = \{x_0, x_1, \ldots x_{n-1}\}$ is a rooted, connected direct acyclic graph, where each non-terminal (internal) node N is labeled by a Boolean variable x_i, and has exactly two outgoing edges, 0-edge and 1-edge, pointing to two nodes called 0-child and 1-child of node N, respectively. Terminal nodes (leaves) are labeled 0 or 1. A 0-*parent* (resp., 1-*parent*) of a node N is a node M such that N is a 0-child (resp, 1-child) of M. For instance, the decision diagrams in Figures 1(b), 1(c), and 1(d) are examples of BDDs.

A BDD is *ordered* (OBDD) if there exists a total order $<$ over the set X of variables such that if an internal node is labeled by x_i, and its 0-child and 1-child have labels x_{i_0} and x_{i_1}, respectively, then $x_i < x_{i_0}$ and $x_i < x_{i_1}$. Hereafter we will consider ordered BDDs only.

In order to obtain an OBDD starting from a BDT we can apply several *reduction rules*:

- **m-rule:** (or merge rule) if M and N are two distinct nodes that are roots of isomorphic subgraphs, then N is deleted, and all the incoming edges of N are redirected to M (N and M are called *mergeable*);
- **r-rule:** (or redundant rule) a node N that has both edges pointing to the same node M is deleted and all its incoming edges are redirected to M (N is called *redundant node* or *r-node*);
- **z-rule:** (or zero-suppress rule) a node N that has the 1-edge pointing to the constant leaf 0 is deleted and all its incoming edges are redirected to the subgraph pointed by the 0-edge (N is called *z-node*).

A *zr-node* is a redundant z-node, i.e., is a node with both edges pointing to the constant leaf 0.

There are different *reduced BDD forms* that derive from the use of one or two reduction rules:

- **QR-BDD:** (*Quasi-Reduced BDD*) [16] is the OBDD derived from a BDT repeatedly applying the m-rule until it is no longer applicable (see Figure 1(b));
- **ROBDD:** (*Reduced Ordered BDD*) [1, 5, 8, 15] is the OBDD derived from a BDT repeatedly applying the m-rule and r-rule until they are no longer applicable (see Figure 1(c));
- **ZDD:** (*Zero-suppressed BDD*) [15, 17] is the OBDD derived from a BDT repeatedly applying the m-rule and z-rule until they are no longer applicable (see Figure 1(d)).

QR-BDDs, ROBDDs and ZDDs are *canonical forms*. In particular, given a function f and a variable ordering $<$, there is exactly one QR-BDD, one OBDD, and one ZDD with variable ordering $<$ that represent f. Thus, once we have fixed the variable ordering, we can compute the QR-BDD, the ROBDD and the ZDD starting from a BDT repeatedly applying the corresponding reduction rules in any order. Moreover, it is possible to first build a QR-BDD (applying the m-rule) and then transform it in a ROBDD (resp., ZDD) using the r-rule (resp., z-rule) on it. In fact, starting from a QR-BDD, the r-rule and the z-rule cannot create new mergeable nodes (as shown in [4] for the r-rule, and in

Section 4 for the z-rule). The interpretation of a QR-BDD as a Boolean function is equivalent to the interpretation of a BDT since the m-rule simply merges isomorphic subgraphs resulting in an OBDD that has all the paths from the root to the leaves containing *all* the variables in X. For ROBDDs and ZDDs we have to give a correct interpretation of possibly missing nodes (in a path), which have been deleted using the r-rule or the z-rule. In particular, a missing variable in a path of a ROBDD means that the variable can have any value (0 or 1). For example, in Figure 1(c), the path "x_0 0-edge x_2 1-edge 1", where x_1 is missing, represents two possible input values (i.e., 001, 011) on which the function takes the value 1. On the other hand, the interpretation of a missing variable x_i in a path of a ZDD means that if $x_i = 1$ the function outputs 0, otherwise (i.e., if $x_i = 0$) the function outputs the value obtained following the path. For example, in Figure 1(d), the path "x_0 1-edge x_1 1-edge 1", where x_2 is missing, means that $f(1, 1, 1) = 0$ and $f(1, 1, 0) = 1$.

3 Index Reconstruction Cost

In this section we discuss error resilient indexes in ZDDs, we analyze the cost of the reconstruction of a corrupted index, and study the impact of the ZDD reduction rules on this cost. This study gives us the knowledge to describe in Section 4 a new index resilient version of ZDDs.

Monitoring the work on error resilient OBDDs [3, 4], we give some definitions useful to describe error resilient ZDDs. Without loss of generality, let us assume that the chosen variable ordering is $x_0 < x_1 < \ldots < x_{n-1}$, so that the index of a variable in a node is the *level* of the node in the corresponding ZDD. In order to facilitate the index reconstruction of a faulty node N we define the range of indexes that contains the original index of the node. Let N be an internal node in a ZDD Z, the *node range* $I_N = [i_P + 1, i_C - 1]$ is the range containing all the possible levels for N in Z, where i_P is the maximum index of N's parents in Z, and i_C is the minimum index of its children, where the leaves have "index" n, and if N is the root, i.e., N has no parent, $i_P = -1$.

Obviously, by definition of ZDD, the index of node N belongs to its range I_N. Thus, in presence of an error in the index i of N we have a lower and an upper bound for the reconstruction of i given by $i_P + 1$ and $i_C - 1$, respectively. In particular, if $i_P + 1 = i_C - 1$, then i is $i_C - 1$.

Let us now examine which characteristics make a ZDD more suitable to the reconstruction of a corrupted index. To this aim, we introduce a metric to measure the cost of the reconstruction of a corrupted index of a ZDD node in the worst case. The *index reconstruction cost* $C(N)$ of the faulty index i in the node N is given by the number of indexes that are candidate to be the correct one in N.

If we consider the case of one fault only in node N, we have that $C(N)$ is at most $|I_N|$. In particular, $C(N) = |I_N|$ whenever there is no additional knowledge on the structure of the ZDD. In the rest of this section, we therefore assume that $C(N) = |I_N|$. Instead, in Section 4 we will study ZDDs with a particular structure implying that $C(N) \leq |I_N|$.

In the best case, for each node N of a ZDD we have that $C(N)$ is 1, meaning that any index can be reconstructed in constant time (considering one single error). This condition is obviously satisfied by BDTs. In fact, in a BDT, all paths from the root to the terminal nodes contain exactly n nodes, where n is the number of input variables. Thus, for each node N, $C(N) = |I_N| = 1$. It is interesting to notice that the optimal cost $C(N) = 1$ can also be reached by reduced ZDDs.

Recalling that a reduced ZDD can be constructed from a BDT by iteratively applying the reduction rules (m-rule and z-rule) and noticing that a BDT has optimal cost, we can study how the node range can increase using the two reduction rules.

We first consider the merge rule, i.e., the rule that is also used for the reduction of OBDDs. In the OBDD context, Theorem 1 in [3] shows that this rule does not increase the index reconstruction cost of the nodes. In fact, the merge of isomorphic subgraphs does not change the node range of the involved nodes. In other words, each node N in a QR-BDDs is such that $C(N) = |I_N| = 1$.

On the other hand, the second reduction rule (z-rule), that distinguishes ZDDs from OBDDs, can increase the index reconstruction costs. For example, consider the ZDD in Figure 1(d), that can be obtained applying the z-rule to the QR-BDD in Figure 1(b). While each node of the QR-BDD has cost 1, node x_1 on the right of the ZDD has cost 2, since its range is increased by the z-rule. We finally note that not always the z-rule increases the node cost.

4 Index-Resilient Reduced ZDDs

The analysis of the previous section shows that, while the merge rule never increases the overall index reconstruction cost, the application of the z-rule could increase it. In this section, we describe a new reduced ZDD model where we maintain some z-nodes in the diagram, in order to guarantee a constant index reconstruction cost for each node. In particular we will define a ZDD, called *index-resilient reduced ZDD*, satisfying the following properties:

1. the index reconstruction cost of each node N is $C(N) = 1$;
2. in presence of k nodes with a corrupted index in an index-resilient ZDD, the cost needed to reconstruct a faulty index is $O(k)$;
3. starting from a QR-BDD, the construction of the corresponding index-resilient reduced ZDD is linear in time;
4. the index-resilient reduced ZDD is canonical.

Due to space limitations, formal proofs are omitted and will be discussed in the extended version of this paper.

We note that, since the z-rule can increase the index reconstruction cost, we could decide not to apply this rule during the reduction of a ZDD. In this way, we only use the m-rule and obtain a QR-BDD that has a cost $C(N) = 1$ for each node N. Recall that an important property of QR-BDD is that each node at level i has all parents at level $i-1$ and all children on level $i+1$. QR-BDDs are still a compact representation and could represent a convenient and canonical trade-off between memory saving, reduction time and error reconstruction time.

However, the use of the z-rule does not always increase the index reconstruction cost. In other words, it is still possible to delete some z-nodes in a QR-BDD guaranteeing that, in the final OBDD, the index reconstruction cost of each node N is still $C(N) = 1$. Most importantly, as we will show in this section, it is possible to apply the z-rule to some z-nodes, and derive a *canonical* OBDD, *more compact* than a quasi-reduced one, and with a cost $C(N) = 1$ for each node N. We will call these OBDDs *index-resilient ZDDs*.

Definition 1 (Index-Resilient ZDD). *An* Index-Resilient ZDD *is an OBDD obtained from a QR-BDD applying several times, possibly never, the z-rule guaranteeing that each internal node N on level i has at least one child on level $i+1$, for any level of the OBDD.*

In particular, a QR-BDD is an index-resilient ZDD where each node on level i has *all* parents on level $i-1$ and *all* children on level $i+1$.

Observe that the index reconstruction cost for any node N in an index-resilient ZDD is $C(N) = 1$, since the variable index of a node N is directly given by $i = \min\{i_0, i_1\} - 1$, where i_0 and i_1 are the indexes of the 0- and 1-child of N. Note that, for any internal node N in a ZDD, the number of children of N is 2, but the number of parents of N can be $O(m)$, where m is the total number of nodes in the ZDD, and, in the worst case, $m \in \Theta(2^n/n)$ [16]. Note also that for the reconstruction of the index of N we do not need to know the indexes of its parents (whose number can be exponential in the number of variables), but only the indexes of its children. In fact, as shown below, we can define a structure where any node containing a variable x_i must have at least one child containing the variable x_{i+1}.

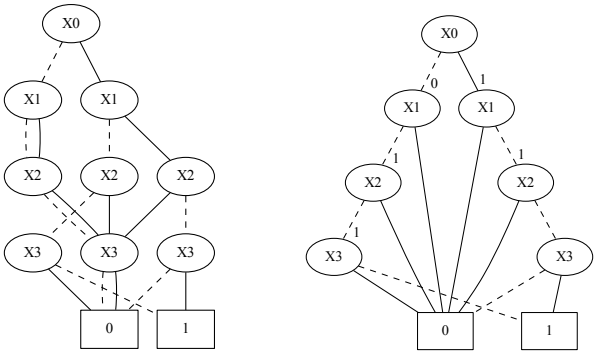
To compute a compact index-resilient ZDD, we start from a QR-BDD deleting some z-nodes while preserving the index-resilient property. For this purpose we first observe that in a QR-BDD there are at most one zr-node and one non-redundant z-node. These nodes are on level $n-1$. See, for example, the QR-BDD depicted in Figure 1(b).

We first consider the zr-node N_{zr}, if existing, in the QR-BDD. We can note that the removal of N_{zr} can generate new z-nodes, i.e., the 1-parents of N_{zr} (see Proposition 1). In particular, if N_{zr} has an r-node parent M, the removal of N_{zr} transforms M in a zr-node but the ZDD is not index-resilient since M is at level $n-2$ and has both children (the 0 constant) at level n. If we remove the zr-node M, we can generate again new z-nodes and one possible zr-node that has reconstruction cost greater than 1. We can, therefore, consider the entire chain of r-nodes that ends with a zr-node defined as follows:

Definition 2 (zr-chain). *A* zr-chain *in an index-resilient ZDD is a chain $C = N_1, N_2, \ldots, N_k$ (with $k \geq 1$) of nodes such that:*

1. *N_1 has no redundant parents,*
2. *N_i, with $i \in [1, \ldots, k-1]$, is an r-node and its unique child is N_{i+1},*
3. *N_k is a zr-node.*

The node N_1 is called head *of the chain, and the leaf* 0 *is the* child *of the chain.*



(a) A zr-chain (b) Two z-chains

Fig. 2. Examples of zr-chain and z-chains. The value $P_C(N)$ is depicted over each node N in the z-chains. The z-chain on the left is removable.

When $k = 1$ the chain corresponds to the zr-node N_1. As already observed, the deletion of a zr-chain generates new z-nodes in the obtained index-resilient ZDD:

Proposition 1. *Let C be a zr-chain in an index-resilient ZDD Z. If C is removed from Z, then any node in Z that is a 1-parent of a node in C becomes a z-node.*

For example, consider the QR-BDD in Figure 2(a) that contains a zr-chain of three nodes. The removal of the zr-chain will produce two new z-nodes (the nodes with indexes x_2 that are not part of the chain).

If we remove the entire zr-chain, the resulting OBDD is still an index-resilient ZDD, as proved in the following proposition. Moreover, in a QR-BDD the zr-chain is unique and, once deleted, the resulting index-resilient ZDD does not contain a new zr-chain.

Proposition 2. *Let B be a QR-BDD containing a zr-chain C. We have that:*

1. *C is the unique zr-chain in B,*
2. *the OBDD B' derived by deleting C from B is an index-resilient ZDD,*
3. *B' does not contain any zr-chain,*
4. *B' does not contain any mergeable node.*

We can observe that, after the deletion of a zr-chain in a QR-BDD, each node N at level i in the resulting index-resilient ZDD has both children at level $i + 1$, or one child at level n (the 0 leaf) and a child at level $i + 1$. Moreover, at level $n - 1$ there exists at most one single z-node (i.e., the node that have the terminal 0 as 1-child and the terminal 1 as 0-child). The index-resilient ZDD obtained after the deletion of the zr-chain, can still contain z-nodes that can be

removable. In order to efficiently test whether we can delete a z-node N, we first need the following parameter that counts the number of parents of N whose index reconstruction cost is affected by the deletion of N.

Definition 3 (P_C). *Let N be a z-node in an index-resilient ZDD resulting by the deletion of the zr-chain from a QR-BDD. $P_C(N)$ is the number of parents P of N satisfying* at least one *of the following properties:*

1. *Both children of P are z-nodes (possibly, the same z-node if P is redundant) and N is the* 1*-child of P;*
2. *P has another child $N' \neq N$ on a level strictly greater than $i+1$, where i is the level of P (i.e., N' is the terminal node 0).*

Note that if N is the root, then $P_C(N) = 0$. We can observe that Definition 3 derives from the fact that the cost $C(P) = 1$, of a node P at level i, is not increased by the deletion of its z-node child N if P has the other child $N' \neq N$, on level $i+1$ and N' cannot be removed. The child N' is not removed in two possible cases: 1) N' is not a z-node; 2) N' is a z-node (like N) but is the 1-child of P. This second criterion is an arbitrary choice due to the necessity of deleting one of the two z-nodes that are children of P while maintaining the index reconstruction cost and the canonicity of the representation. More precisely, when a node P has two children that are z-nodes, one of them can be removed without increasing the cost of P. In this paper we always remove the 0-child of P in order to guarantee that the resulting index-resilient ZDD is canonical (see Theorem 2). The choice of removing the 1-children is similar.

For example, see the index-resilient ZDD in Figure 2(b). Each z-node N in the figure has a value that corresponds to $P_C(N)$. We note that $P_C(N)$ can be efficiently computed with a simple visit of a index-resilient ZDD obtained after the deletion of the zr-chain from the QR-BDD.

When the QR-OBDD is constructed, the zr-chain (if existing) deleted, and P_C is computed, we can define chains of z-nodes (*z-chains*) and we can characterize the z-chains that can be removed, maintaining equal to 1 the index reconstruction cost of each remaining node. We therefore introduce the concept of *removable z-chain.*

Definition 4 (Removable z-chain). *A* removable z- chain *in an index-resilient ZDD, which does not contain zr-chains, is a chain $C = N_1, N_2, \ldots, N_k$ (with $k \geq 1$) of z-nodes such that:*

1. *N_i, with $i \in [2, \ldots, k]$, is the 0-child of N_{i-1},*
2. *$P_C(N_1) = 0$,*
3. *$\forall i \in [2, \ldots, k]$, $P_C(N_i) = 1$,*
4. *if M is a z-node then $P_C(M) > 1$, where M is the the 0-child of N_k.*

The node N_1 is called head *of the chain, and M is called* child *of the chain.*

The second requirement states that the head of the chain N_1 can be removed without affecting the reconstruction cost of its parents, as detailed in Proposition 4. Note that this requirement implies that all parents of N_1 are not z-nodes

or r-nodes. The third requirement states that any other node N_i $(1 < i \leq k)$ of the chain affects only the reconstruction cost of its parent in the chain. The last requirement guarantees that the removable z-chain is maximal. When the chain is composed by a single z-node node N, we have that N is removable when $P_C(N) = 0$. Consider, for example, see the index-resilient ZDD in Figure 2(b). While, the z-chain on the left is removable, the z-chain on the left is not removable since the first node N_1 has $P_C(N_1) = 1$.

In an index-resilient ZDD there are no "crossing" z-chains, i.e., a node cannot be part of two different z-chains. This is a direct consequence of the following property.

Proposition 3. *In an index-resilient ZDDs there are no nodes with two or more z-nodes as parents.*

The following proposition shows that the deletion of all removable z-chains in an index-resilient ZDD Z does not change the overall index reconstruction cost, i.e., after the removal of the chains, each internal node on level i still has at least a child on level $i + 1$, for any level i in Z.

Proposition 4. *Let $C = N_1, N_2, \ldots, N_k$, $k \geq 1$, be a removable z-chain in an index-resilient ZDD, which does not contain a zr-chain. The OBDD resulting from the deletion of C is still an index-resilient ZDD.*

Observe that once removable z-chains have been deleted, we are left with an index-resilient ZDD that can still contain some z-nodes: those that do not form a removable chains.

We can now propose a new OBDD reduction algorithm that, starting from a QR-BDD, deletes first the zr-chain and then all the removable z-chains. The following Theorem 1 shows that the deletion of removable z-chains does not construct new removable z-chains. Therefore, after the removal of the zr-chain, we can detect (and than delete) all the removable chains at the same time.

The reduction algorithm is based on a constant number of visits starting from a quasi-reduced OBDD. The first visit is a breadth first search used to detect and remove the zr-chain, if exists. Another visit is used to compute the parameter P_C for each z-node; then with a breadth first visit, all removable z-chains are identified and their nodes are removed with a final visit of the OBDD, executed by a simple recursive depth first visit that deletes from the OBDD all nodes identified as removable.

The correctness of the new reduction algorithm is proved in the following theorem.

Theorem 1. *Let B be a quasi-reduced OBDD. The reduction algorithm computes an index-resilient ZDD Z equivalent to B that contains neither removable z-chains nor a zr-chain.*

The cost of the algorithm is linear in the size of the quasi-reduced OBDD in input, as it consists in a constant number of visits of the data structure.

We now formally introduce the concept of *Index-Resilient Reduced ZDD*.

Definition 5 (Index-Resilient Reduced ZDD). *An index-resilient ZDD is* reduced *if it contains neither a zr-chain nor removable z-chains.*

Theorem 2. *Let Z be an index-resilient reduced ZDD obtained with the reduction algorithm. Then*

1. *for each node N in Z, $C(N) = 1$;*
2. *Z does not contain mergeable nodes;*
3. *Z is canonical, i.e., given a function f and a variable ordering $<$, Z is the only index-resilient reduced ZDD with variable ordering $<$ that represents f.*

The index reconstruction cost remains limited even in presence of more than one error on the indexes, as stated and proved in the following theorem, that shows a result similar to the one obtained for OBDDs in [4].

Theorem 3. *The reconstruction cost of a node N on level i in an index-resilient reduced ZDD Z affected by k errors on the indexes is $O(\min(k, 2^{n-i}))$.*

Finally observe that, even if in our analysis we have implicitly assumed that a ZDD is constructed correctly, and that memory faults occur when the data structure is in use, this assumption can be completely removed for index-resilient reduced ZDDs. Indeed, their construction starts from a binary decision tree that is transformed into a QR-BDD, and in both models each node has all children on the level immediately below. Moreover, during the execution of the reduction algorithm, we always guarantee that each node has at least one child on the level below, thus a faulty index can be immediately detected and restored.

5 Conclusion

This paper has proposed a new ZDD canonical form that is resilient to errors in indexes. This form can be derived in linear time starting from a quasi-reduced OBDD. Future work on this subject includes the analysis of error in edges. Indeed, this problem is part of the more general problem of designing an error resilient DAG structure. Furthermore, it could be interesting to design error resilient algorithms for standard operations on ZDDs, like union, intersection, and set difference.

References

1. Akers, S.: Binary Decision Diagrams. IEEE Transactions on Computers 27(6) (1978)
2. Aumann, Y., Bender, M.A.: Fault Tolerant Data Structures. In: 37th Annual Symposium on Foundations of Computer Science, FOCS (1996)
3. Bernasconi, A., Ciriani, V., Lago, L.: Error Resilient OBDDs. In: IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 246–249 (2013)
4. Bernasconi, A., Ciriani, V., Lago, L.: On the Error Resilience of Ordered Binary Decision Diagrams. Technical Report arXiv:1404.3919 (2014)

5. Bryant, R.: Graph Based Algorithm for Boolean Function Manipulation. IEEE Transactions on Computers (1986)
6. Ciriani, V., di Vimercati, S.D.C., Foresti, S., Livraga, G., Samarati, P.: An OBDD approach to enforce confidentiality and visibility constraints in data publishing. Journal of Computer Security 20(5), 463–508 (2012)
7. Drechsler, R.: Verifying integrity of decision diagrams. In: Computer Safety, Reliability and Security (1998)
8. Ebendt, R., Fey, G., Drechsler, R.: Advanced BDD Optimization. Springer (2005)
9. Finocchi, I., Grandoni, F., Italiano, G.: Designing reliable algorithms in unreliable memories. Computer Science Review 1(2), 77–87 (2007)
10. Finocchi, I., Italiano, G.: Sorting and Searching in Faulty Memories. Algorithmica (2008)
11. Finocchi, I., Grandoni, F., Italiano, G.F.: Optimal resilient sorting and searching in the presence of memory faults. Theor. Comput. Sci. 410(44), 4457–4470 (2009)
12. Finocchi, I., Grandoni, F., Italiano, G.F.: Resilient dictionaries. ACM Transactions on Algorithms 6(1) (2009)
13. Italiano, G.: Resilient Algorithms and Data Structures. In: Algorithms and Complexity (2010)
14. Jacob, B., Ng, S., Wang, D.: Cache, DRAM, Disk. Morgan Kaufmann (2008)
15. Knuth, D.: The Art of Computer Programming Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams. Addison-Wesley Professional (2009)
16. Liaw, H.T., Lin, C.S.: On the OBDD-representation of general Boolean functions. IEEE Transactions on Computers (1992)
17. Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In: ACM/IEEE 30th Design Automation Conference (DAC), pp. 272–277 (1993)
18. Minato, S.: Data Mining Using Binary Decision Diagrams. In: Progress in Representation of Discrete Functions, ch. 5, pp. 97–109. Morgan & Claypoo (2010)
19. Minato, S.: Techniques of bdd/zdd: Brief history and recent activity. IEICE Transactions 96-D(7), 1419–1429 (2013)
20. Minato, S., Kimihito, I.: Symmetric item set mining method using zero-suppressed bdds and application to biological data. Information and Media Technologies 2(1), 300–308 (2007)
21. Requeno, J.I., Colom, J.M.: Compact representation of biological sequences using set decision diagrams. In: Rocha, M.P., Luscombe, N., Fdez-Riverola, F., Rodríguez, J.M.C. (eds.) 6th International Conference on PACBB. AISC, vol. 154, pp. 231–240. Springer, Heidelberg (2012)
22. Taylor, D.: Error models for robust storage structures. In: 20th International Symposium on Fault-Tolerant Computing (1990)
23. Yoon, S., Nardini, C., Benini, L., De Micheli, G.: Discovering coherent biclusters from gene expression data using zero-suppressed binary decision diagrams. IEEE/ACM Trans. Comput. Biol. Bioinformatics 2(4), 339–354 (2005)

On Partial Vertex Cover and Budgeted Maximum Coverage Problems in Bipartite Graphs

Bugra Caskurlu[1,⋆,⋆⋆], Vahan Mkrtchyan[2,⋆], Ojas Parekh[3,⋆⋆⋆], and K. Subramani[2,⋆,†]

[1] Computer Engineering Department, TOBB University of Economics and Technology, Ankara, 06560, Turkey
caskurlu@gmail.com
[2] LDCSEE, West Virginia University, Morgantown, WV, USA
vahan.mkrtchyan@mail.wvu.edu, ksmani@csee.wvu.edu
[3] Sandia National Laboratories, Albuquerque, NM, USA
odparek@sandia.gov

Abstract. Graphs are often used to model risk management in various systems. Particularly, Caskurlu et al. in [6] have considered a system which essentially represents a tripartite graph. The goal in this model is to reduce the risk in the system below a predefined risk threshold level. It can be shown that the main goal in this risk management system can be formulated as a Partial Vertex Cover problem on bipartite graphs. It is well-known that the vertex cover problem is in **P** on bipartite graphs; however, the computational complexity of the partial vertex cover problem on bipartite graphs is open. In this paper, we show that the partial vertex cover problem is **NP-hard** on bipartite graphs. Then, we show that the budgeted maximum coverage problem (a problem related to partial vertex cover problem) admits an $\frac{8}{9}$-approximation algorithm in the class of bipartite graphs, which matches the integrality gap of a natural LP relaxation.

Keywords: Bipartite graph, vertex cover problem, partial vertex cover problem, budgeted maximum coverage problem.

1 Introduction

Covering problems arise often in practice. A mobile phone service provider should ensure that its base stations cover the signals transmitted from the phones of its customers. A chain market such as Walmart should ensure that it has a store close to its customers. The applications of the covering problems are not limited to corporations to sell a service to customers. The Air Force on a no-fly zone mission or border patrol officers trying to secure borders are to solve some form of a covering problem.

⋆ This research has been supported in part by the Air Force of Scientific Research through Award FA9550-12-1-0199.

⋆⋆ This work is done when the author was at West Virginia University. The author was supported by the National Science Foundation through Awards CNS-0849735 and CCF-0827397.

⋆⋆⋆ Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

† The author is supported by the National Science Foundation through Award CCF-1305054.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 13–26, 2014.

In many real life situations the corporations or the government is constrained in the resources it can allocate for the covering mission. The constraints may be hard constraints such as a government agency to operate within its approved budget, or profit dictated soft constraints such as a mobile phone service provider may decide not to cover a rural area since the revenues will not match the covering costs. Therefore, the goal in many real life situation can be cast as covering the domain as much as possible for a given fixed amount of resources to be allocated.

There is merit in studying partial covering problems both due to their wide applicability in a large range of applications and their theoretical importance as being natural generalizations of classical covering problems. In this paper, we study the partial vertex cover (PVC) problem and the related budgeted maximum coverage problem on bipartite graphs. Though it is well-known that the vertex cover problem is polynomial-time solvable on bipartite graphs, the computational complexity of partial vertex cover when restricted to bipartite graphs (PVCB) remained open. In parallel to this, let us note that approximation algorithms for this problem are known, with the best approximation algorithm in the literature offering an approximation ratio of $(\frac{4}{3} + \epsilon)$ [19].

The contributions of this paper are as follows:

(i) The partial vertex cover and budgeted maximum coverage problems are **NP-hard** on bipartite graphs.
(ii) The (weighted) budgeted maximum coverage problem on bipartite graphs admits an $\frac{8}{9}$-approximation algorithm, matching the integrality gap of the natural linear programming relaxation.

The rest of this paper is organized as follows: Section 2 presents a formal definition of the main problems that are considered in the paper. We present a concrete application to motivate the PVCB problem in Section 3. The related work is presented in Section 4. The computational complexity of the PVCB problem is established in Section 5. In Section 6, we present an $\frac{8}{9}$-approximation algorithm for the budgeted maximum coverage problem on bipartite graphs; note that this problem is computationally identical to the PVCB problem. We conclude and point out several research directions in Section 7.

2 Statement of Problems

If $G = \langle V, E\rangle$ is a graph, and u is a vertex, e is an edge of G, respectively, then we will say that u covers e in G, if u is incident to e, that is, u is one of the end-vertices of e. For a set $V' \subseteq V$, let $E_{V'}$ denote the set of edges of G that are covered by vertices of V'.

In the classical vertex cover (VC) problem, we are given an undirected graph $G = \langle V, E\rangle$, where V is the vertex set with $|V| = n$, E is the edge set with $|E| = m$. The goal is to find a minimum cardinality subset $V' \subset V$, such that $E_{V'} = E$.

In this paper we will consider the partial vertex cover (PVC) problem restricted to bipartite graphs.

Definition 1. *Given an integer t, and an undirected bipartite graph $G = \langle V, E\rangle$. The PVCB problem is defined as finding a minimum cardinality subset $S \subset V$ such that $|E_S| \geq t$.*

It is trivial to observe that the PVC problem is a generalization of the VC problem. We will also consider the budgeted maximum coverage problem, and the following its extension.

Definition 2. *Given an integer k, and undirected bipartite graph $G = \langle V, E, w\rangle$, and $w : E \to Z$ denoting a weighting function from the set of edges to the set of positive integers. The EBMCB problem is defined as finding a subset $S \subset V$ with $|S| \leq k$ maximizing $\sum_{e:e\in E_S} w(e)$.*

Definition 3. *Given an integer k, and undirected bipartite graph $G = \langle V, E\rangle$ with V denoting the set of vertices, E denoting the set of edges. The BMCB problem is defined as finding a subset $S \subset V$ with $|S| \leq k$ maximizing the number of covered edges.*

The PVC and the BMC problems are equivalent from the perspective of exact solution. This follows from the observation that one can simply swap the budget constraint with objective and reverse the optimization direction. On the other hand, let us observe that the two problems are likely to be different from the perspective of approximate solution. The BMC problem for sets (not necessarily for graphs) admits a $(1 - \frac{1}{e})$-approximation algorithm [15], where $e = 2.718....$, and it is known that there exists a constant $\gamma > 0$, such that Set Cover problem admits no $\gamma \cdot \log n$ approximation [8], under the assumption that **NP** has no quasi-polynomial time algorithms.

3 Motivation

The number of attacks faced by an ordinary device that has an access to Internet remains quite high, which means that it is unreasonable to expect a manual response to an attack. A good example of a system for which an automated response is needed are devices that have an operating system. One of the approaches to handle the situation, is to dynamically control the device's vulnerability to threats and limit undesirable consequences as much as possible.

In order to protect a system, one needs to minimize the risk it faces. It is reasonable to assume that the risk depends on three factors, which are threats and their probabilities of occurrence, the existing weaknesses of the system, and the consequences that the system experiences after a successful attack.

While threats are not under the control of the device, the other two can be handled by it. Here it is implicitly assumed that the device can be protected by decreasing its functionality. This implies that our main goal is becoming to minimize the tension between functionality and security. In great contrast with the approach of statically configuring a system, one can dynamically monitor the risk level, and allow users to have maximum functionality where the level of risk is assumed to be on a predefined level.

In [6], the risk of a computational system is modeled as a flow between the first and last partitions in a tripartite graph, where the vertices of the three partitions represent threats to the system, vulnerabilities of the system, and the assets of the system as shown in Figure 1.

In the risk management model given in [6], the goal is to reduce the risk in the system (flow between the first and last partitions) below a predefined risk threshold level by

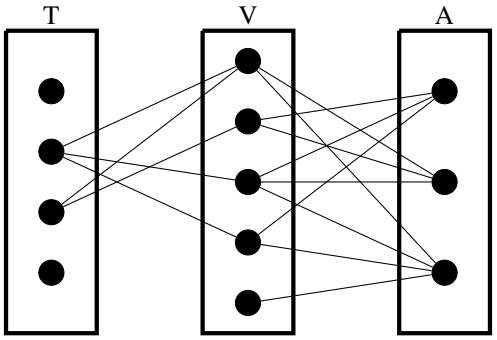


Fig. 1. Risk in a computational system can be modeled in terms of its constituent components. The threats, weaknesses (corresponding to specific vulnerabilities), and assets form three disjoint sets, named as T, V, and A respectively. An edge between vertices represents a contribution to the system risk. The system's risk is the total flow between the first and third sets.

either restricting the permissions of the users, or encapsulating the system assets. These two strategies correspond to deleting minimum number of vertices from the second and the third partitions of the tripartite graph so that the flow between the first and the third partitions are reduced below the predefined threshold level. The equivalence of this risk management system and the partial vertex cover problem on bipartite graphs is established in [6].

From the point of view of risk management the important instances of the problem correspond to the cases when the graph modeling the system has a bounded degree, or has a relatively predictable structure (such as being a tree). This paper focuses on the problem when the graphs are arbitrary bipartite graphs.

4 Related Work

The vertex cover (VC) problem is one of the classical **NP-complete** problems listed by Karp [13]. There are several polynomial-time approximation algorithms for the VC problem within a factor of 2, and the best-known approximation algorithm for the VC problem has an approximation factor of $\left(2 - \theta\left(\frac{1}{\sqrt{\log n}}\right)\right)$ [12]. The VC problem is known to be **APX-complete** [21]. Moreover, it cannot be approximated to within a factor of 1.3606 unless **P = NP** [7], and not within any constant factor smaller than 2, unless the *unique games conjecture* is false [14].

Since the PVC problem subsumes the VC problem for $t = m$, all the hardness results given above for the VC problem directly apply to the PVC problem. The PVC problem and the partial variants of similar graph problems have been extensively studied for more than a decade [4], [18], [16], [17], [3]. Particularly, there are an $O(n \cdot \log n + m)$-time 2-approximation algorithm [20], that is based on the method of primal-dual, a combinatorial 2-approximation algorithm [3], and some $(2-o(1))$-approximation algorithms [2], [5], [9], [11]. Let us also note that in [19] for each $\epsilon > 0$ an approximation algorithm with

performance ratio $(\frac{4}{3}+\epsilon)$ for WPVCB is presented. Moreover, a $\frac{3}{4}$-approximation algorithm for the weighted budgeted maximum coverage problem is general graphs given in [1].

Although the VC problem and the PVC problem have almost matching approximation ratios and inapproximability results, the PVC problem is in some sense more difficult than the VC problem. For instance, the PVC problem is **W[1]-complete** while the VC problem is fixed parameter tractable [10].

5 Computational Complexity of the PVCB Problem

In this section we prove Theorem 1, which shows that the PVCB problem is **NP-hard**. This also establishes the hardness of the weighted versions of the PVCB and the BMCB problems.

Theorem 1. *The PVCB problem is* **NP-hard**.

Proof. We will prove Theorem 1 by giving a Karp reduction from the CLIQUE problem. Recall that in the CLIQUE problem, we are given an undirected graph $G' = \langle V', E' \rangle$, and an integer k, and the goal is to find whether there exists a complete subgraph of G' with k vertices. Assume we are given an arbitrary undirected graph $G' = \langle V', E' \rangle$, where n' and m' denote $|V'|$ and $|E'|$ respectively, and an integer k. We construct a corresponding bipartite graph $G = \langle V_1 \cup V_2, E \rangle$ as explained below.

For every vertex $v'_i \in V'$, G has a corresponding vertex $v_i \in V_1$. For every edge $e' \in E'$, there is a corresponding *edge block* in G. The term *edge block* refers to two vertices and an edge in between. So, for each edge $e' \in E'$, G has two corresponding vertices $e_1 \in V_2$ and $e_2 \in V_1$ and the edge (e_1, e_2). In order to capture the incidence matrix of G', for each edge $e' = (v'_i, v'_j)$ of G', G has 2 additional edges (v_i, e_1) and (v_j, e_1). Let n and m denote the number of vertices and edges of G, respectively. Notice that the bipartite graph G has $n' + 2 \cdot m'$ vertices and $3 \cdot m$ edges. More precisely, we have $n = n' + 2 \cdot m'$, and $m = 3 \cdot m'$. We use the term *left vertex* of an edge block for the vertex of the edge block that belongs to V_2. The other vertex of the edge block that belongs to V_1, is referred to as the *right vertex* of the edge block throughout the paper.

In Figure 2, we are given a simple undirected graph G' on the left that consists of 2 vertices v'_1 and v'_2 and an edge $e' = (v'_1, v'_2)$ in between. The figure has the corresponding bipartite graph G on the right. The vertices v_1 and v_2 of G correspond to 2 vertices of G'. The 2 vertices e_1 and e_2 of G and the edge e in between is the corresponding edge block of the edge e' of G'. The 2 edges (v_1, e_1) and (v_2, e_1) of G capture the incidence matrix of G'.

We will prove Theorem 1 by showing that solving the CLIQUE problem on G' reduces to solving the PVCB problem on G with $t = m - \frac{k \cdot (k-1)}{2}$. In the rest of the paper, without loss of generality, we will assume that $m' > \frac{k \cdot (k-1)}{2}$, and $k \geq 5$. Notice that these assumptions do not violate the soundness of the proof since the CLIQUE problem is still **NP-hard** after these assumptions on the input. We precisely will show the following: there is a complete subgraph of k vertices on G' if and only if there is a subset of $k + m' - \frac{k \cdot (k-1)}{2}$ vertices V'' of G such that the number of edges that are covered by V'' is at least $m - \frac{k \cdot (k-1)}{2}$.

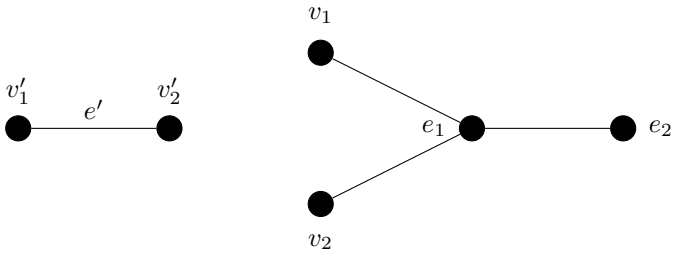


Fig. 2. Construction of the corresponding bipartite graph $\mathbf{G}$ for a given undirected graph $\mathbf{G}'$

Without loss of generality we can assume that V'' does not contain the right vertex of any of the edge blocks of G, since the right vertex of an edge block is incident to only one edge and that edge can be covered by the left vertex of the edge block as well. Therefore, without loss of generality, we can assume that all the vertices of V'' are the vertices that correspond to the vertices of G', or the left vertices of the edge blocks. Since the number of edges that are to be covered by V'' is at least $m - \frac{k\cdot(k-1)}{2}$, the number of edges that are not covered by V'' is at most $\frac{k\cdot(k-1)}{2}$. Therefore, V'' has to contain the left vertices of at least $m' - \frac{k\cdot(k-1)}{2}$ edge blocks. We will complete the proof of Theorem 1 by proving Lemma 1, which maps the yes instances of the CLIQUE problem to the yes instances of the PVCB problem, and Lemma 2, which maps the no instances of the CLIQUE problem to the no instances of the PVCB problem.

Lemma 1. *If there exists a complete subgraph of k vertices on G', then there exists a subset V'' of vertices of G such that $|V''| = k + m' - \frac{k\cdot(k-1)}{2}$, and V'' covers at least $m - \frac{k\cdot(k-1)}{2}$ edges of G.*

Proof. Assume that G' has a complete subgraph of k vertices and let V'' be composed of the following $k + m' - \frac{k\cdot(k-1)}{2}$ vertices of G. For every vertex of the complete subgraph of G', let the corresponding vertex of G be in V''. Notice that there are exactly k such vertices. The complete subgraph of this k vertices has $\frac{k\cdot(k-1)}{2}$ edges in G'. Therefore, there are $m' - \frac{k\cdot(k-1)}{2}$ edges of G' that are *not* in the complete subgraph of k vertices in G'. For each of these $m' - \frac{k\cdot(k-1)}{2}$ edges of G', let the left vertex of the corresponding edge block in G be contained in V''. Notice that there are exactly $m' - \frac{k\cdot(k-1)}{2}$ such vertices in V''. So, $|V''| = k + m' - \frac{k\cdot(k-1)}{2}$ as desired.

All we need to prove is that V'' covers at least $m - \frac{k\cdot(k-1)}{2}$ edges of G. Let us first consider the edges of G that capture the incidence relation of the edges of G'. Recall that for every edge $e' = (v_i', v_j')$ of G', there are 2 edges in G to capture the incidence relation of e', namely (v_i, e_1) and (v_j, e_2). So, in total there are $2 \cdot m'$ such edges in G. The $k \cdot (k-1)$ edges of G that capture the incidence relation of the $\frac{k\cdot(k-1)}{2}$ edges of the complete subgraph of G' are covered by the k vertices of V'' that correspond to the k

vertices of G'. The remaining $2 \cdot m' - k \cdot (k-1)$ edges of G, that capture the incidence relation of the $m' - \frac{k \cdot (k-1)}{2}$ edges of G' that are not part of the complete subgraph, are covered by the left vertices of the $m' - \frac{k \cdot (k-1)}{2}$ edge blocks in V''. Therefore, all $2 \cdot m'$ edges of G that capture the incidence relation of the m' edges of G' are covered by V''.

Recall that there are m' additional edges in G. These edges are the edges of the m' edge blocks. The left vertices of the $m' - \frac{k \cdot (k-1)}{2}$ edge blocks that are contained in V'' cover $m' - \frac{k \cdot (k-1)}{2}$ of those edges. Therefore, there are only $\frac{k \cdot (k-1)}{2}$ edges of G that are not covered by V''. So, V'' covers $m - \frac{k \cdot (k-1)}{2}$ edges as stated by Lemma 1. □

Lemma 2. *If G' does not have a complete subgraph of k vertices, then no subset V'' of vertices of G such that $|V''| = k + m' - \frac{k \cdot (k-1)}{2}$ covers at least $m - \frac{k \cdot (k-1)}{2}$ edges of G.*

Proof. Assume G' does not have a complete subgraph of k vertices. For the purpose of contradiction, assume that there is a subset V'' of vertices of G such that $|V''| = k + m' - \frac{k \cdot (k-1)}{2}$, and V'' covers at least $m - \frac{k \cdot (k-1)}{2}$ edges of G.

Since V'' covers at least $m - \frac{k \cdot (k-1)}{2}$ edges of G and $m' > \frac{k \cdot (k-1)}{2}$, V'' covers at least $m' - \frac{k \cdot (k-1)}{2}$ edges of the edge blocks. Therefore, V'' contains the left vertices of at least $m' - \frac{k \cdot (k-1)}{2}$ edge blocks. Since $|V''| = k + m' - \frac{k \cdot (k-1)}{2}$, V'' contains at most k vertices of G that correspond to the vertices of G'.

First consider the case where V'' contains exactly k vertices of G that correspond to the vertices of G', and exactly $m' - \frac{k \cdot (k-1)}{2}$ left vertices of edge blocks. Since there are only $m' - \frac{k \cdot (k-1)}{2}$ left vertices of edge blocks, V'' does not cover $\frac{k \cdot (k-1)}{2}$ edges of the edge blocks. Since V'' covers at least $m - \frac{k \cdot (k-1)}{2}$ in total, V'' covers all the edges of G that capture the incidence relation of all the edges of G'. Since G' does not have a complete subgraph of k vertices, the k vertices of G that correspond to the vertices of G' cover both of the edges that capture the incidence relation of $\frac{k \cdot (k-1)}{2} - \alpha$ edges of G' for some $1 \leq \alpha < \frac{k \cdot (k-1)}{2}$. Since V'' covers all $2 \cdot m'$ edges of G that capture the incidence relation of the edges of G', V'' contains the left vertices of all the edge blocks of G that correspond to the $m' - \frac{k \cdot (k-1)}{2} + \alpha$ edges of G'. This is a contradiction since we assumed that V'' contains exactly $m - \frac{k \cdot (k-1)}{2}$ left vertices of edge blocks.

Therefore, V'' contains exactly $k - l$ vertices of G that correspond to some $k - l$ vertices of G', and exactly $m' - \frac{k \cdot (k-1)}{2} + l$ left vertices of edge blocks for some $0 < l < k$. Recall that the incidence relation of each edge of G' is captured by 2 edges in G. Notice that the subgraph formed by this $k - l$ vertices of G' contains at most $\frac{(k-l) \cdot (k-l-1)}{2}$ edges of G'. Therefore, the corresponding $k - l$ vertices of V'' cover both of the incidence edges of at most $\frac{(k-l) \cdot (k-l-1)}{2}$ edges of G'. In other words, at least one incidence edge of $m' - \frac{(k-l) \cdot (k-l-1)}{2}$ edges of G' is not covered the $k - l$ vertices of G that correspond to some $k - l$ vertices of V''. Since we already have $\frac{k \cdot (k-1)}{2} - l$ edges of edge blocks left uncovered, the left vertices of the edge blocks in V'' has to cover an incidence edge for at least $m' - \frac{(k-l) \cdot (k-l-1)}{2} - l$ edges of G'. This is not possible since the left vertex of each edge block in G covers the corresponding incidence edges of exactly one edge of G', and $m' - \frac{(k-l) \cdot (k-l-1)}{2} - l > m' - \frac{k \cdot (k-1)}{2} + l$ for $k \geq 5$. □

6 An $\frac{8}{9}$-Approximation Algorithm for the EBMCB Problem

The edge-weighted budgeted maximum coverage problem in general graphs is known to be NP-hard, and the result from the previous section shows that this the case for bipartite graphs as well. Thus a natural question is the approximability of the problem. Ageev and Sviridenko [1] give a $\frac{3}{4}$-approximation for general graphs via the pipage rounding technique. We present an $\frac{8}{9}$-approximation for bipartite graphs and show that this matches the integrality gap of the LP-relaxation. We note that our techniques can also be extended to general graphs to give an alternate LP-based $\frac{3}{4}$-approximation.

We employ the following linear programming relaxation for the problem on a bipartite graph $G = (V = V_L \cup V_R, E)$, where V_L and V_R are the independent sets. For a set $S \subseteq V$, we let $S_L = S \cap V_L$ and $S_R = S \cap V_R$.

$$\max \sum_{e \in E} w(e) \cdot z_e$$

$$x_u + x_v \geq z_{uv}, \text{ for all } uv \in E \tag{1}$$

$$\sum_{u \in V} x_u \leq k \tag{2}$$

$$0 \leq x_u \leq 1, \text{ for all } u \in V \tag{3}$$

$$z_e \leq 1, \text{ for all } e \in E \ , \tag{4}$$

where we assume that $w(e) \geq 0$ for all $e \in E$. This assumption allows us to omit the usual non negativity constraint on each z_e. This omission will slightly simplify our analysis and presentation.

Our results exploit the structure of extreme points of the above polyhedron. Consider a feasible solution (x, z) for the above linear program. We say an edge uv is *slack* (with respect to (x, z)) if,

(i) $x_u + x_v > z_{uv}$, or
(ii) $z_{uv} < 1$.

If an edge is not slack, we call it *tight*. The following lemma shows that extreme points are well-structured. For an extreme point $(\hat{x}, \hat{z})$, let $\hat{V} = \{u \in V \mid 0 < \hat{x}_u < 1\}$ be the set of vertices with fractional $\hat{x}$ value, and let $\hat{G} = (\hat{V}, \hat{E})$ be the graph induced by $\hat{V}$.

Lemma 3. *The tight edges induce a single connected component in $\hat{G}$, spanning all its vertices.*

Proof. Suppose, for the moment, that there is a vertex set $J \subseteq \hat{V}$ such that all edges $e \in \hat{E}$ with exactly one endpoint in J are slack. Note that every edge $uv \in E$ with exactly one endpoint in $\hat{V}$ is slack. To see this, suppose $\hat{x}_u \in \{0, 1\}$ and $\hat{x}_v \in (0, 1)$; if $\hat{x}_u = 0$, condition (ii) is satisfied, and otherwise condition (i) is satisfied since $z_{uv} \leq 1$. Thus we actually have that any edge in E with exactly one endpoint in J is slack.

Now we will show that we may perturb the $\hat{x}$ values for vertices in J such that all constraints, except possibly (2), remain satisfied. We will add a value $\varepsilon \neq 0$ to $\hat{x}_u$ for each vertex $u \in J_L$, and we will subtract ε from each $\hat{x}$ value for vertices in J_R. This

modified solution continues to satisfy (1) for each edge with both endpoints in J. We observed above that edges in E with exactly one endpoint in J are slack.

Suppose we have such a slack edge uv with $u \in J$ and $v \notin J$. Let $\delta \in \{\varepsilon, -\varepsilon\}$ represent the perturbation added to $\hat{x}_u$. If uv satisfies (i), then (1) is satisfied for uv by selecting ε of small enough magnitude so that $\delta \leq \hat{x}_u + \hat{x}_v - \hat{z}_{uv}$, where the right-hand side is positive as desired. If uv satisfies (ii), then we add δ to z_{uv} as well. This ensures that (1) is satisfied for uv, and we may select ε of small enough magnitude to satisfy (4); this does not affect the feasibility of any other constraint, even those for other edges.

In addition to the above constraints on the selection of ε, we may select ε of small enough magnitude to satisfy (3), since vertices in J have fractional $\hat{x}$ value. Let the components of the vector $\boldsymbol{\varepsilon} \in \mathcal{R}^{V \times E}$ correspond to the perturbation described above. We have indeed shown that $(\hat{x}, \hat{z}) + \boldsymbol{\varepsilon}$ is feasible for all constraints except possibly (2); however, we have also shown that we may select ε so that $(\hat{x}, \hat{z}) - \boldsymbol{\varepsilon}$ is feasible for these constraints as well. These two solutions average to $(\hat{x}, \hat{z})$, and since $\varepsilon \neq 0$ they are distinct. In fact if we have $|J_L| = |J_R|$, then observe that both of these solutions satisfy (2) as well and are feasible, contradicting the assumption that $(\hat{x}, \hat{z})$ is an extreme point.

To finish the proof, suppose we have two disjoint vertex sets $S, T \subseteq \hat{V}$ akin to J from above; specifically, any edge in E with exactly one endpoint in some S or T is slack. We may independently select $\varepsilon_S \neq 0$ and $\varepsilon_T \neq 0$ for S and T, respectively, so that the only potentially violated constraint is (2). From above, we may assume $|S_L| \neq |S_R|$ and $|T_L| \neq |T_R|$. Thus, in order to satisfy (2), we need only choose ε_S and ε_T so that:

$$\varepsilon_S \cdot (|S_L| - |S_R|) + \varepsilon_T \cdot (|T_L| - |T_R|) = 0 \ ,$$

which can be done so that ε_S and ε_T have arbitrarily small magnitude as required. Thus $(\hat{x}, \hat{z})$ is a convex combination of the feasible solutions $(\hat{x}, \hat{z}) \pm (\boldsymbol{\varepsilon_S} + \boldsymbol{\varepsilon_T})$ — a contradiction.

If the tight edges induced more than one connected component in $\hat{G}$ (possibly one with just an isolated vertex), then two of these components could play the role of S and T above, which cannot exist. □

We will use this lemma to show the structure of fractional entries in an extreme point solution $(\hat{x}, \hat{z})$. As above let $\hat{V}$ correspond to the vertices with fractional $\hat{x}$ value. Let $\hat{V}_0$ and $\hat{V}_1$ correspond to vertices with $\hat{x}$ value 0 and 1, respectively.

Theorem 2. *The fractional entries of $\hat{x}$ take one of two values, α or $1 - \alpha$, for some $\alpha \in (0, 1)$. Moreover,*

$$\min\{|\hat{V}_L|, |\hat{V}_R|\} + |\hat{V}_1| \leq k \ .$$

Proof. By the above lemma, there is a tree consisting of tight edges spanning the vertices in $\hat{V}$. By the definition of a tight edge uv, we have that $\hat{x}_u + \hat{x}_v = \hat{z}_{uv}$ and $\hat{z}_{uv} = 1$. Select a root r for this tree and suppose $\hat{x}_r = \alpha$. Now the first claim of the theorem follows, since it is apparent that vertices with odd distance to r must have $\hat{x}$ value $1 - \alpha$ and those with even distance to r have value α.

Suppose that the vertices in $\hat{V}_L$ have value α. Then the budget constraint, (2) becomes:

$$\alpha \cdot |\hat{V}_L| + (1 - \alpha) \cdot |\hat{V}_R| + |\hat{V}_1| \leq k \ ,$$

from which the second claim follows. □

One consequence of Theorem 2 is that $\hat{V}_1$ together with the smaller of $\hat{V}_L$ and $\hat{V}_R$ is a feasible solution. Now, let us compose two feasible solutions that will help us in designing the approximation algorithm. Let A be the set of edges between $\hat{V}_L$ and $\hat{V}_R$. Let B_L be the set of edges with one endpoint in $\hat{V}_L$ and one endpoint in $\hat{V}_0$, and let B_R be defined analogously. Note that we will always select the vertices in $\hat{V}_1$ and ignore the edges with at least one endpoint in $\hat{V}_1$ below. For the sake of exposition, suppose $|\hat{V}_L| \leq |\hat{V}_R|$ and that each vertex in $\hat{V}_L$ has $\hat{x}$ value α (hence those in $\hat{V}_R$ have value $1-\alpha$).

1. Select $\hat{V}_L$ and augment this by greedily selecting an additional $k - |\hat{V}_L|$ vertices from $\hat{V}_R$ with largest coverage. This solution has coverage at least $w(A)+w(B_L)+\frac{k-|\hat{V}_L|}{|\hat{V}_R|} \cdot w(B_R)$.
2. Greedily select k vertices from $\hat{V}_R$ with largest coverage. This solution has coverage at least $\frac{k}{|\hat{V}_R|} \cdot (w(A) + w(B_R))$.

The above are derived by applying the following lemma. For a vertex $v \in V$ let $w(v) = \sum_{e \in \delta(v)} w(e)$, where $\delta(v)$ denotes the set of edges of G, that are incident to v. As with edge sets, for a set $S \subseteq V$, let $w(S) = \sum_{u \in S} w(u)$.

Lemma 4. *Let $S \subseteq V$ be an independent set of vertices. The t vertices in S with largest coverage have a collective coverage of at least $t \cdot \frac{w(S)}{|S|}$.*

Proof. Let $T \subseteq S$ be the t vertices in S with largest coverage. Suppose, for the sake of contradiction, that $w(T) < t \cdot \frac{w(S)}{|S|}$. Then some vertex in T has coverage less than $\frac{w(S)}{|S|}$. Therefore by choice of T, every vertex in $S - T$ has coverage less than $\frac{w(S)}{|S|}$. This implies that $w(S) = w(T) + w(S-T) < t \cdot \frac{w(S)}{|S|} + (|S| - t) \cdot \frac{w(S)}{|S|} = w(S)$. □

Applying the lemma with $S = \hat{V}_R$ and $t = k - |\hat{V}_L|$ and $t = k$ yield the bounds from (1) and (2), respectively. As in the proof of Theorem 2, the budget constraint (2) from the LP (ignoring $\hat{V}_1$) yields $k \geq \alpha \cdot |\hat{V}_L| + (1-\alpha) \cdot |\hat{V}_R|$. Substituting this bound for k into the above coverage bounds yield the following lower bounds on the respective coverage obtained by our two solutions:

1. $W_1 = w(A) + w(B_L) + (1-\alpha) \cdot \beta \cdot w(B_R)$, and
2. $W_2 = (1 - \alpha \cdot \beta) \cdot (w(A) + w(B_R))$,

where $\beta = 1 - \frac{|\hat{V}_L|}{|\hat{V}_R|}$. Note that we may assume that $\beta \in (0,1)$. If $\beta = 0$ then $|\hat{V}_L| = |\hat{V}_R| \leq k$, and the set with better coverage is optimal. If $\beta = 1$ then $\hat{V}_L = \emptyset$, hence $B_L = \emptyset$. Since our extreme point solution $\hat{x}$ is not integral, we have $\alpha \in (0,1)$, hence the edge set $A = \emptyset$ as well. The coverage of Solution (2) in this case is $(1-\alpha) \cdot (w(A) + w(B_R)) = w(A) + \alpha w(B_L) + (1-\alpha) \cdot w(B_R) \geq OPT$.

We have shown that we can achieve a coverage of $\max\{W_1, W_2\}$. All we need is to show that the left-side of the inequality given below, an upper bound on the performance

of our algorithm, is at most $\frac{9}{8}$. In order to prove this, consider a nonlinear program that seeks to maximize the ratio,

$$\frac{w + \alpha \cdot w_L + (1-\alpha) \cdot w_R}{\max\{W_1, W_2\}} \geq \frac{OPT}{\max\{W_1, W_2\}} \ .$$

The program has variables w, w_L, and w_R which correspond to values for $w(A)$, $w(B_L)$, and $w(B_R)$ respectively. Similarly there is a variable m corresponding to $\max\{W_1, W_2\}$.

$$\begin{aligned} \rho = \max \frac{w + \alpha \cdot w_L + (1-\alpha) \cdot w_R}{m} \\ w + w_L + (1-\alpha) \cdot \beta \cdot w_R \leq m \\ (1 - \alpha \cdot \beta) \cdot (w + w_R) \leq m \\ w, w_L, w_R \geq 0 \\ \alpha, \beta \in (0,1) \ . \end{aligned}$$

The program finds a worst-case setting of the parameters for our algorithm, and the optimal value ρ is an upper bound on the worst-case performance of our algorithm. We simplify the program a bit by letting $x = \frac{w}{m}$, $y = \frac{w_L}{m}$, and $z = \frac{w_R}{m}$.

$$\begin{aligned} \rho = \max(x + \alpha \cdot y + (1-\alpha) \cdot z) \\ x + y + (1-\alpha) \cdot \beta \cdot z \leq 1 \\ (1 - \alpha \cdot \beta) \cdot (x + z) \leq 1 \\ x, y, z \geq 0 \\ \alpha, \beta \in (0,1) \ . \end{aligned}$$

For any fixed values of α and β, we consider the above program as a linear program, whose target function depends on α and β: $\rho = \rho(\alpha, \beta)$. Now, in order to simplify our analysis, we drop the non-negativity constraints on x and z, obtaining an upper bound on $\rho(\alpha, \beta)$.

$$\begin{aligned} \rho(\alpha, \beta) \leq \max(x + \alpha \cdot y + (1-\alpha) \cdot z) \\ x + y + (1-\alpha) \cdot \beta \cdot z \leq 1 \\ (1 - \alpha \cdot \beta) \cdot x + (1 - \alpha \cdot \beta) \cdot z \leq 1 \\ y \geq 0 \ . \end{aligned}$$

Note that the above LP is feasible and bounded for $\alpha, \beta \in (0,1)$, since the first constraint implies an upper bound of $\frac{1}{\beta}$ on the objective function. Hence, we may assume without loss of generality that we seek an extreme-point solution, in which case all three constraints must be tight since we have three variables. Thus $y = 0$, and we may solve the following linear system to obtain x and z in terms of α and β:

$$\begin{aligned} x + (1-\alpha) \cdot \beta \cdot z = 1 \\ (1 - \alpha \cdot \beta) \cdot x + (1 - \alpha \cdot \beta) \cdot z = 1 \ . \end{aligned}$$

Taking into account these values, the objective function becomes,

$$\begin{aligned} f(\alpha,\beta) &= x + \alpha\cdot y + (1-\alpha)\cdot z \\ &= (1-\alpha)\cdot(x+z) + \alpha\cdot x \\ &= \frac{(1-\alpha)\cdot[1-(1-\alpha)\cdot\beta] + \alpha\cdot(1-\beta)}{(1-\alpha\cdot\beta)\cdot[1-(1-\alpha)\cdot\beta]} \ . \end{aligned}$$

Thus we obtain an upper bound on our worst-case approximation ratio by maximizing this quantity with respect to α and β:

$$\rho = \max_{\alpha,\beta\in[0,1]} \rho(\alpha,\beta) \leq \max_{\alpha,\beta\in[0,1]} f(\alpha,\beta) \ .$$

Maximizing $f(\alpha,\beta)$ is equivalent to maximizing

$$g(\alpha,\beta) = 1 - \frac{1}{f(\alpha,\beta)} = \frac{\alpha\cdot(1-\alpha)\cdot\beta\cdot(1-\beta)}{(1-\alpha)\cdot[1-(1-\alpha)\cdot\beta]+\alpha\cdot(1-\beta)} \ .$$

Maximizing $g(\alpha,\beta)$ is equivalent to minimizing

$$h(\alpha,\beta) = \frac{1}{g(\alpha,\beta)} = \frac{1}{1-\beta} + \frac{1}{\alpha\cdot(1-\alpha)\cdot\beta} \ .$$

In minimizing $h(\alpha,\beta)$ for any fixed value of $\beta \in (0,1)$, we see that $\alpha = \frac{1}{2}$, and some elementary calculus yields that $h(\frac{1}{2},\beta)$ is minimized over $\beta \in (0,1)$ at $\beta = \frac{2}{3}$. Evaluating f at these worst-case values of $\alpha = \frac{1}{2}$ and $\beta = \frac{2}{3}$ yields $\rho \leq \frac{9}{8}$.

Remark 1. It is natural to wonder whether the $\frac{8}{9}$-approximation factor in the above algorithm can be improved. It turns out that the integrality gap of the linear program [22] that we considered is $\frac{8}{9}$. It is an indication that using this linear program alone, one cannot improve the approximation factor. In order to see this, consider the tree from figure 3, and let the vertex budget $k = 2$. Observe that in this tree with two vertices one can cover four edges. However, the linear program corresponding to this tree has a fractional solution $(\hat{x},\hat{z})$ whose cost is $\frac{9}{2}$. In order to construct such a solution, for each vertex u let $\hat{x}_u = \frac{1}{2}$, if u is not a leaf, and is zero, otherwise. For each edge e let $\hat{z}_e = \frac{1}{2}$, if e is incident to a leaf, and is one, otherwise. It is not hard to see that this fractional solution is a feasible solution, and its cost is $\frac{9}{2}$, which implies that the integrality gap of our linear program is $\frac{4}{\frac{9}{2}} = \frac{8}{9}$.

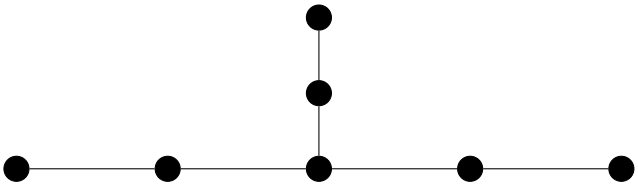


Fig. 3. An example showing that the integrality gap is $\frac{8}{9}$

7 Conclusion

In this paper, we studied the partial vertex cover, the budgeted maximum coverage problems and their weighted analogues on bipartite graphs. We proved that the PVCB problem is **NP-hard**. Then we presented an $\frac{8}{9}$-approximation algorithm for the EBMCB problem, which is computationally identical to the PVCB problem.

From our perspective, the following lines of research appear promising:

- Determining whether the PVCB problem is **APX-hard** or not.
- Obtaining an α-approximation algorithm for the PVCB problem where $\alpha \leq \frac{4}{3}$.
- Considering the main problems in the class of bounded-degree graphs.

References

1. Ageev, A.A., Sviridenko, M.I.: Approximation algorithms for maximum coverage and max cut with given sizes of parts. In: Cornuéjols, G., Burkard, R.E., Woeginger, G.J. (eds.) IPCO 1999. LNCS, vol. 1610, pp. 17–30. Springer, Heidelberg (1999)
2. Bar-Yehuda, R.: Using homogeneous weights for approximating the partial cover problem. J. Algorithms 39(2), 137–144 (2001)
3. Bar-Yehuda, R., Flysher, G., Mestre, J., Rawitz, D.: Approximation of Partial Capacitated Vertex Cover. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 335–346. Springer, Heidelberg (2007)
4. Bläser, M.: Computing small partial coverings. Inf. Process. Lett. 85(6), 327–331 (2003)
5. Bshouty, N.H., Burroughs, L.: Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In: Morvan, M., Meinel, C., Krob, D. (eds.) STACS 1998. LNCS, vol. 1373, pp. 298–308. Springer, Heidelberg (1998)
6. Caskurlu, B., Gehani, A., Bilgin, C.C., Subramani, K.: Analytical models for risk-based intrusion response. Computer Networks (Special issue on Security/Identity Architecture) 57(10), 2181–2192 (2013)
7. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Ann. of Math. (2) 162(1), 439–485 (2005)
8. Feige, U.: A threshhold of $ln(n)$ for approximating Set Cover. In: Proc. of the 28th Annual ACM Symposium on the theory of Computing, pp. 314–318 (1996)
9. Gandhi, R., Khuller, S., Srinivasan, A.: Approximation algorithms for partial covering problems. J. Algorithms 53(1), 55–84 (2004)
10. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized Complexity of Generalized Vertex Cover Problems. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 36–48. Springer, Heidelberg (2005)
11. Hochbaum, D.S.: The t-Vertex Cover Problem: Extending the Half Integrality Framework with Budget Constraints. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 111–122. Springer, Heidelberg (1998)
12. Karakostas, G.: A better approximation ratio for the vertex cover problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1043–1050. Springer, Heidelberg (2005)
13. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
14. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\epsilon$. J. Comput. Syst. Sci. 74, 335–349 (2008)

15. Khuler, S., Moss, A., Naor, J.(S.): The Budgeted Maximum Coverage problem. Inform. Process. Lett. 70(1), 39–45 (1999)
16. Kneis, J., Langer, A., Rossmanith, P.: Improved upper bounds for partial vertex cover. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 240–251. Springer, Heidelberg (2008)
17. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Intuitive algorithms and t-vertex cover. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 598–607. Springer, Heidelberg (2006)
18. Kneis, J., Mölle, D., Rossmanith, P.: Partial vs. Complete Domination: t-Dominating Set. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 367–376. Springer, Heidelberg (2007)
19. Könemann, J., Parekh, O., Segev, D.: A unified approach to approximating partial covering problems. Algorithmica 59(4), 489–509 (2011)
20. Mestre, J.: A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. Algorithmica 55(1), 227–239 (2009)
21. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. System Sci. 43(3), 425–440 (1991)
22. Vazirani, V.V.: Approximation Algorithms. Springer-Verlag New York, Inc., New York (2001)

Perfect Pipelining for Streaming Large File in Peer-to-Peer Networks

Fei Chen and Xiaowei Wu

The University of Hong Kong

Abstract. We study the efficiency of large file streaming in a peer-to-peer network in which a large file is cut into many pieces of equal size, and initially all pieces are known only by one source node. We analyze the number of rounds required, called the *finishing time*, for all nodes in the network to collect all pieces in the default order.

Based on the basic PUSH-PULL protocol, we design the Constant Out-degree Protocol (COP). At the beginning of the protocol, each node selects a constant number of neighbors, with only whom communication will be initiated. We focus our analysis on the performance of COP on preferential attachment graphs, which are believed to model peer-to-peer networks well. We show that a tight bound of $\Theta(B + \log n)$ rounds can be achieved with high probability for streaming B pieces in preferential attachment graphs with n nodes. Moreover, we show that there is a dichotomy in the results depending on how neighbors are contacted in each round; specifically, when each node avoids repeating initiation with neighbors in the previous $M \geq 2$ rounds, then the finishing time is improved to $\Theta(B + \frac{\log n}{\log \log n})$ with high probability.

For lower bounds, we show that there is a class of regular graphs in which perfect pipelining is impossible for any PUSH-PULL protocols using random neighbor selection.

1 Introduction

The problem of information disseminating in a distributed network has been extensively studied. It is assumed that this information, usually called a rumor, was known by only one node in a large connected network. The goal is to disseminate this rumor to all nodes in the network, with bounded bandwidth and number of communications. Many protocols have been proposed based on various network topologies [7,14,15]. Among these protocols, distributed algorithms have gained more attention since each node does not require global information about the network.

A gossip based protocol was first introduced by Demers et al. [7] for maintaining replicated databases. In this protocol, each node is only allowed to communicate with its own neighbors in each round. A similar distributed protocol called PUSH-PULL protocol was introduced later. In this protocol, each node initiates communication with one of its neighbors per round; the node *pushes* the rumor to the neighbor if it is informed of the rumor, or *pulls* the rumor from

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 27–38, 2014.

the neighbor if the neighbor is informed. A common performance measure of protocols is the number of rounds required to have all nodes informed, which is called the *finishing time*.

We consider the rumor spreading problem when the rumor is not a tiny piece of information, but a large file which can not be transmitted from one node to another in one round. In this case, we divide the file into B small pieces so that each piece can be transmitted in one round. This problem can also be considered as the multiple-rumor version of the rumor spreading problem with streaming requirement. We analyse the finishing time for all nodes in the network to collect all B pieces in the correct order. For instance, in peer-to-peer video streaming, a node needs to receive earlier portions first and the next portion needs to arrive before all currently received portions have been played. We consider two properties of a protocol that are important for streaming multiple pieces: *perfect pipelining* and *ordered pieces transmission*, which we explain below.

1.1 Our Contribution and Results

We study the large file streaming problem in which B ordered pieces of information known to one source node are to be spread in a graph with n nodes. We consider protocols that have the following properties.

1. Loosely speaking, a (randomized) protocol for multiple pieces spreading achieves *perfect pipelining* on $G(V, E)$ if the following holds: if the finishing time for spreading one piece is T (with high probability), then the finishing time for spreading B pieces is $O(B + T)$ (with high probability) in G.
2. A protocol employs *ordered pieces transmission* if the pieces are transmitted and received in the correct order.

Our results and contribution are summarized as follows.

Achieving perfect pipelining with COP. In Section 4, we modify the basic PUSH-PULL protocol and design the Constant Out-degree Protocol (COP). At the beginning of the protocol, each node selects a constant number of neighbors, with only whom communications will be initiated.

We show that using COP, steaming B pieces in preferential attachment graphs requires $O(B + \log n)$ rounds, which achieves perfect pipelining. We also prove a tight lower bound $\Omega(B + \log n)$ for the finishing time. As considered by Doerr et al. [8], if each node does not repeat neighbor initiation from the previous 2 rounds, COP can achieve finishing time $O(B + \frac{\log n}{\log \log n})$, which is also tight. This theoretical gap in the finishing time between the two cases suggests that after the random neighbor selection phase to identify out-going neighbors, deterministic round-robin for the pieces streaming phase is optimal.

We briefly outline our technical contributions. According to Arthur and Panigrahy [1], spreading B pieces in a graph with diameter d and maximum degree D using random PUSH requires $O(D(B + \max\{d, \log n\}))$ rounds with high probability (w.h.p.), no matter how the pieces are chosen for transmission. For preferential attachment graphs, the resulting upper bound $O(\sqrt{n}(B + \log n))$ is loose

because the maximum degree is as large as $\Omega(\sqrt{n})$. However, the large degree nodes are crucial for single piece spreading [8] since they are highly connected. To overcome this technical hurdle, our protocol (COP) is based on the following two observations: (1) for single piece spreading, even if each node is "active" for only a constant number of rounds, w.h.p., the piece can still reach all nodes via paths of length $O(\frac{\log n}{\log \log n})$; (2) we can direct the edges to achieve constant out-degree, even when the underlying undirected graph has a high maximum degree; this allows us to achieve the same result as in [1] if we use both PUSH and PULL.

Implication on the Performance of Streaming. We show that our protocol achieves tight upper bound $\Theta(B + \log n)$ when the *ordered pieces transmission* is adopted. In this case, all nodes will receive all pieces of the file in the correct order and perfect pipelining is achieved. As far as we know, this is the first time perfect pipelining is proved for streaming large file in peer-to-peer network.

Impossibility of Perfect Pipelining in Some Cases. In Section 3 we consider the basic PUSH-PULL protocol for streaming B pieces in complete graphs. Unfortunately, perfect pipelining cannot be achieved with this simple protocol. In Section 5, we give a lower bound for the finishing time in a particular class of D-regular graphs, called *Necklace Graphs*. We show that spreading B pieces under any PUSH-PULL protocol in which communication partners are chosen uniformly at random in each round (no matter how those pieces are selected for transmission) in Necklace Graph requires $\Omega(D(B + d))$ rounds, where d is the diameter, for any $\omega(1) \leq D \leq o(n)$.

1.2 Related Work

The single piece (usually called a rumor) spreading problem has been extensively studied. Using the PUSH-PULL protocol, the finishing time (also called stopping time) is $\Theta(\log n)$ for both complete graphs [16,13] and random graphs [9,10]. Rumor spreading in preferential attachment graphs (PA-graphs) was also studied and two tight upper bounds $O(\log n)$ and $O(\frac{\log n}{\log \log n})$ for different protocols were shown by Doerr et al. [8]. Given a graph with conductance ϕ, an $O(\phi^{-1} \log n)$ upper bound was proved by Giakkoupis [11], which is tight. For rumor spreading in general graphs, Censor-Hillel et al. [4] gave an algorithm which constructs subgraphs along with the rumor spreading process and showed that the finishing time is at most $O(d+\text{poly} \log n)$, where d is the diameter of the underlying graph, with no dependence on the conductance.

Although observed by experiments [5] and extensively applied on the Internet, the high efficiency of multiple-rumor spreading is hardly analyzed. Deb et al. [6] presented a protocol based on random linear network coding and showed that in the complete graph spreading $B = \Theta(n)$ rumors requires at most $O(n)$ rounds, which is asymptotically optimal. Also using network coding, Haeupler [12] gave a protocol that achieves perfect pipelining in many other graphs. In their protocol, communication partners exchange a random linear combination of rumors and they show that after receiving enough combinations, all rumors can be decoded locally.

Given a graph with diameter $d = \Omega(\log n)$ and maximum out-degree D, an $O(D(B + d))$ upper bound on the finishing time was proved by Arthur and Panigrahy [1] for spreading B rumors with PUSH. Since $\Omega(B + d)$ is a trivial lower bound, this upper bound is tight when D is a constant. This result actually inspired the construction of our protocol COP, which attempts to find a subgraph that has constant out-degree and small diameter in the first step.

2 Preliminaries

Consider a graph G with n nodes and B pieces indexed by $\{1, 2, \ldots, B\}$; for streaming application, we assume that smaller index means higher priority. Throughout this paper we suppose that initially a single *source node* $u_0 \in V$ has all B pieces while other nodes have no pieces. We say a node is *waiting* if it has no pieces, *downloading* if it has at least one piece, and *finished* if it has all the pieces. We use log to denote the logarithm with base 2.

Pieces are transmitted in a round-by-round manner. During a round of the pieces streaming process, we say there is a piece *transmission* between two nodes if some piece is transmitted from one node to the other. The *finishing time* is the number of rounds required until all nodes in the graph become finished.

The Basic PUSH-PULL Protocol. For each node $u \in V$ and integer $t \geq 1$, let p_t^u be the current downloading position of u at time t. Note that $p_0^u = B$ if $u = u_0$, and $p_0^u = 0$ otherwise. In each round each node independently selects a neighbor uniformly at random, and initiates communication with that neighbor. In round $t \geq 1$, for an instance of communication initiated by u between u and v, the transmission follows the following rules.

1. If $p_{t-1}^u = p_{t-1}^v$, then no transmission is made;
2. If $p_{t-1}^v > p_{t-1}^u$, then node u *pulls* from node v the piece with index $p_{t-1}^u + 1$;
3. If $p_{t-1}^v < p_{t-1}^u$, then node u *pushes* to node v the piece with index $p_{t-1}^v + 1$.

Note that each node initiates communication with exactly one neighbor in each round and at most one piece is transmitted. Also note that all B pieces are initiated by the source node at the beginning and hence each node in the graph will receive all the pieces in the fixed order $1, 2, \ldots, B$.

In Section 3, we show that the basic PUSH-PULL protocol cannot achieve perfect pipelining on the complete graph; in Section 4, we modify the basic PUSH-PULL protocol and design the Constant Out-degree Protocol, which achieves perfect pipelining for directed random graphs and preferential attachment graphs.

3 Low Efficiency of Basic PUSH-PULL

To motivate why we consider COP, we analyze the performance of the basic PUSH-PULL protocol on complete graph K_n with source node u_0 having all B pieces. We show that perfect pipelining can not be achieved since (on average) each node has to wait for $\Omega(\log n)$ rounds to get the next piece.

Theorem 1. *The finishing time for streaming B pieces from one source node in a complete graph with n nodes using the basic PUSH-PULL protocol is $\Omega(B \log n)$ with probability $1 - \exp(-\Theta(B))$.*

Note that by experiments, Theorem 1 is not true if the pieces are spread randomly. We conjecture that the finishing time for spreading B pieces with random pieces selection is $O(B + \log n)$. The proof of Theorem 1 is included in the full version. Intuitively, when all nodes in the graph are waiting for the same piece, the efficiency is low since most of the communications are useless. However, if we can guarantee that at least a constant fraction of communications are useful, then after $O(B)$ rounds, all nodes will become finished. Following this idea, we define in Section 4 a protocol that constructs a constant maximum out-degree subgraph before applying the basic PUSH-PULL protocol. We show that in several cases, the subgraph constructed by our protocol has small diameter and using basic PUSH-PULL protocol, perfect pipelining can be achieved.

4 Constant Out-Degree Protocol

Constant Out-degree Protocol. Let D be a fixed constant parameter used in the protocol. Suppose we are given an input graph with one source node.
(1) Out-going Edge Selection Phase. A directed subgraph H is formed, where each node u chooses independently its *outgoing edges* and *outgoing neighbors*, with only whom node u will initiate communications in the next phase. We consider all input graphs as multi-graphs. Each node u chooses $\min\{D, deg(u)\}$ (out-going) edges incident to u uniformly at random without replacement. The chosen edges are the outgoing edges and the incident nodes (excluding u) are the outgoing neighbors of u in H.
(2) Pieces Streaming Phase. In each round, each node chooses one of its sampled outgoing edges uniformly at random to initiate communication. In round $t \geq 1$, for an instance of communication initiated by u between nodes u and v, the transmission follows as the basic PUSH-PULL protocol.

In other words, we construct a directed subgraph H and apply the basic PUSH-PULL protocol to H, where the directions of initiation coincide with the directions of edges. Since the only difference between COP and the basic PUSH-PULL protocol is an extra neighbor selection process, we know that if perfect pipelining can be achieved in the (random) subgraph H using basic PUSH-PULL, then perfect pipelining is achieved by COP in the input graph. We show in this section that perfect pipelining can be achieved by applying COP to stream multiple pieces in directed random graphs and preferential attachment graphs. Unless otherwise specified, the *diameter* of a graph refers to that of the underlying induced **undirected** graph.

Deterministic Rumor Spreading Phase. Suppose we consider the case that after the random subgraph H is constructed in the first phase each node initiates communication with its outgoing neighbors in a round-robin way. Recall that since the out-degree of the subgraph H is at most D, there is communication

between two neighboring nodes at most every D rounds. It follows that if the diameter of H is at most d, all nodes will become finished after $D(B+d)$ rounds.

Fact 1. *Given any input graph with n nodes, if the underlying undirected graph of the subgraph constructed by COP with constant parameter D has diameter d and each node contacts its neighbors in a round-robin way, then for all $B \geq 1$, the finishing time for streaming B pieces starting from one source node u_0 in the input graph using COP with parameter D is at most $D(B+d) = O(B+d)$.*

Note that Fact 1 holds with probability 1 since it is deterministic. If the diameter of the graph is $\Omega(\log n)$, the random neighbor chosen process achieves a similar upper bound w.h.p.. We use the following fact [1, Proposition 4.1] on streaming multiple pieces along a path.

Fact 2 (Streaming Multiple Pieces on a Path). *Suppose there exists a path $P(v_0, v_s) = \{v_0, v_1, \ldots, v_s\}$ of length s between two nodes v_0 and v_s such that in each round, communication between any two adjacent nodes v_i and v_{i+1} is performed with probability at least p independently for each pair of adjacent nodes. Assume only v_0 has all B pieces and all other nodes are waiting. Then if $s' \geq s$, with probability $1 - 2\exp(-\frac{s'}{2})$, node v_s will become finished after $\frac{4(B+4s')}{p}$ rounds.*

The fact above implies that by using COP protocol, perfect pipelining can be achieved in any graph if the subgraph H constructed has a small diameter d.

Theorem 2. *Suppose the subgraph constructed by COP has diameter d, then for all $B \geq 1$, the finishing time for streaming B pieces starting from one source node u_0 is at most $O(B + \max\{d, \log n\})$ with probability $1 - O(\frac{1}{n})$.*

Proof. We denote the subgraph constructed by COP with constant parameter D by H, which has diameter d. Hence for any node v, there exists a path $P = (u_0, u_1, \ldots, u_{s-1}, u_s = v)$ in the undirected graph induced by H between u_0 and v, where $s \leq d$ is the length of the path.

Hence, in the directed graph H, for any $i = \{0, 1, \ldots, s-1\}$, there is a directed edge between nodes u_i and u_{i+1}. Since both nodes u_i and u_{i+1} have out-degree at most D, which is a constant, communication between u_i and u_{i+1} will be performed with probability at least $\frac{1}{D}$ in each round. However, if there exists two directed edges (u_i, u_{i+1}) and (u_i, u_{i-1}) that start at u_i, then the communications between u_i and u_{i+1}, u_i and u_{i-1} are not independent (only one of them can happen in each round). To make the communications independent, we define *phases* such that each phase consist of two rounds. In each phase, for each $i = \{1, \ldots, s-1\}$, we assume that the communication between u_i and u_{i+1} is invalid in the first round and the communication between u_i and u_{i-1} is invalid in the second round. Note that the assumption will only make the downloading process slower. Under this assumption, we know that for each $i = \{0, 1, \ldots, s-1\}$, in each phase, communication between u_i and u_{i+1} will be performed with probability at least $\frac{1}{D}$, independently.

Hence, for each node v in H, there is a path of length $s \leq d$ between u_0 and v such that in each phase, communications between any two adjacent nodes

will be performed with probability at least $\frac{1}{D}$, independently for each pair of adjacent nodes. Hence by Fact 2, if $s' \geq s$, with probability $1 - 2\exp(-\frac{s'}{2})$, node v will become finished after $4D(B + 4s')$ phases. If $s \geq 4\log n$, then set $s' = s$ and node v will become finished after $4D(B + 4s) = O(B + d)$ phases with probability $1 - O(\frac{1}{n^2})$. If $s < 4\log n$, then set $s' = 4\log n$ and node v will become finished after $4D(B + 16\log n) = O(B + \log n)$ phases with probability $1 - O(\frac{1}{n^2})$. Therefore, with probability $1 - O(\frac{1}{n^2})$, node v will become finished after $O(B + \max\{d, \log n\})$ phases, which means $O(B + \max\{d, \log n\})$ rounds. A union bound on $n - 1$ nodes in H yields the result.

By Theorem 2, we know that the finishing time of COP is highly related to the diameter d of the subgraph constructed in the first step. If $d = \Omega(\log n)$, then using round-robin in the pieces streaming phase has the same bound as random neighbor selection. However, as we shall later see, the PA-graphs have diameter $O(\frac{\log n}{\log\log n})$ w.h.p., and indeed we show that there is a dichotomy in the finishing times between deterministic and random pieces streaming phases.

COP on Directed Random Graphs. We apply COP with $D = 8$ to stream B pieces in directed random graphs $G_{n,p}$ with $p \geq \frac{6\log n}{n}$. Observe that the neighbor selection phase actually constructs a random D-out-degree subgraph. For space reasons, we show in our full version that a random D-out-degree graph induces an expander with diameter $O(\log n)$. Hence the following theorem follows immediately; its proof is also presented in the full version.

Theorem 3 (Finishing Time of COP in $G_{n,p}$). *For all $B \geq 1$, the finishing time for streaming B pieces starting from one source node in directed random graphs $G_{n,p}$ with n nodes and $p \geq \frac{6\log n}{n}$ using the constant out-degree protocol with $D = 8$ is at most $O(B + \log n)$ with probability $1 - O(\frac{1}{n})$.*

The intuition behind the proof can be summarized as follows.

1. When $p \geq \frac{6\log n}{n}$, the random subgraph construction can be equivalently modelled as in complete graph.
2. For any node u, the number of nodes reachable from u grows exponentially as the distance increases.
3. For any two nodes u and v, if both of them have many reachable nodes within a distance of $O(\log n)$, then it is very likely that there exists a common node w that is reachable from both u and v by paths with length $O(\log n)$.

Next we consider the finishing time of multiple-piece streaming in *Preferential Attachment graphs* (PA-graphs). The notion of preferential attachment graphs was first introduced by Barabási and Albert [2], and they have been used to model social and peer-to-peer networks. In this work, we follow the formal definition by Bollobás et al. [3]. Let G_m^n be a PA-graph. We denote by $deg_G(v)$ the degree of a vertex v in G_m^n.

Definition 1 (Preferential Attachment Graphs). *Let $m \geq 2$ be a fixed constant parameter. The random graph G_m^n is an undirected graph on the vertex set $V := \{1, \ldots, n\}$ inductively defined as follows.*

1. G_m^1 *consists of a single vertex with* m *self-loops.*
2. *For all* $n > 1$, G_m^n *is built from* G_m^{n-1} *by adding the new node* n *together with* m *edges* $e_n^1 = \{n, v_1\}, \ldots, e_n^m = \{n, v_m\}$,*inserted one after the other in this order. Let* $G_{m,i-1}^n$ *denote the graph right before the edge* e_n^i *is added. Let* $M_i = \sum_{v \in V} deg_{G_{m,i-1}^n}(v)$ *be the sum of the degrees of all the nodes in* $G_{m,i-1}^n$*. The endpoint* v_i *is selected randomly such that* $v_i = u$ *with probability* $\frac{deg_{G_{m,i-1}^n}(u)}{M_i+1}$*, except for* n *that is selected with probability* $\frac{deg_{G_{m,i-1}^n}(n)+1}{M_i+1}$.

It can be easily seen that for $m = 1$ the graph is disconnected w.h.p.; so we focus on the case $m \geq 3$. For each node i and $k \in [m]$, let $\tau_{i,k}$ be the node chosen by i in the k-th rounds after i is added to G_m^n. We call the first node added to G_m^n node v_1. Bollobás et al. [3] introduced an equivalent way of generating G_m^n that is more accessible. In that model, each node i is assigned a random variable $\omega_i \in (0, 1)$ following some distribution and edges are added to node i with probability proportional to ω_i. The formal definition of the new model is given in [3]. Under this equivalent model, a node i is called *useful* if $\omega_i \geq \frac{\log^2 n}{n}$ and *non-useful* otherwise.

We apply COP with $D = 3$ to stream B pieces in PA-graph and show that the finishing time in a PA-graph is at most $O(B + \log n)$. Note that the PA-graph is a multi-graph and the outgoing neighbors of a node is a multiset in the subgraph constructed by COP.

Theorem 4 (Finishing Time of COP In PA-graphs). *For all* $B \geq 1$*, the finishing time for streaming* B *pieces starting from any source node in a PA-graph* G_m^n *using the constant out-degree protocol with* $D = 3$ *is at most* $O(B + \log n)$ *with probability* $1 - o(1)$.

Communication Initiation without Repeat. In each round of the downloading process, each node initiates communication uniformly at random with an outgoing neighbor that is not initiated with in the last $\min\{M, deg_G(u) - 1\}$ rounds; for a multi-set of outgoing neighbors, multiplicities are respected. For $D = 3$ and $M = 2$, this is equivalent to round-robin neighbor selection.

Lemma 1. *[8, Lemma 5.8] There exists a property* $\mathcal{P}$ *of* G_m^n *that holds with probability* $1 - o(1)$ *such that conditioning on* $\mathcal{P}$*, with probability at least* $1 - n^{-\Omega(n)}$*, using only* PUSH *on* G_m^n *(avoiding repeated neighbor initiation from the previous* $M \geq 2$ *rounds), a piece initiated at a node* u *reaches a useful node in* $O(\log \log n)$ *rounds.*

Lemma 2. *[8, Lemma 5.11] If each node* v *in* G_m^n *receives an independent random variable* x_v *such that* $x_v = 1$ *with constant probability* p *and* $x_v = 0$ *otherwise, then with probability* $1 - o(1)$*, for each useful node* u*, there exists a path of length* $O(\frac{\log n}{\log \log n})$ *between* u *and node* v_1 *such that every second node* v *in the path has degree* m*, its two neighbors in the path are* $\tau_{v,1}$ *and* $\tau_{v,2}$ *and has* $x_v = 1$.

Note that all x_v's are independent and the randomness of x_v comes from the behavior of node v. In the rest of the section, we assume the PA-graph G_m^n under consideration has the properties stated above.

4.1 Upper Bound for the Stopping Time

First, it is obvious that each node in G_m^n has $deg_G(u) \geq m \geq 3$ and the out-degree of each node in H is $\min\{D, deg_G(u)\} = 3$, for the case $D = 3$. A node v in H is called a *fast* node if $deg_G(v) = m$ and it chooses both $\tau_{v,1}$ and $\tau_{v,2}$ as its outgoing neighbors in H.

In Lemma 3, we show that for each non-useful node v, there is at least one useful node u that can be reached from v by a directed path of length $O(\log \log n)$. In Lemma 4, we show that all useful nodes are connected to node v_1 by paths of length $O(\frac{\log n}{\log \log n})$ in H and every second node in the paths is fast. We include the proof of Lemma 4 in our full version. These two lemmas imply that the diameter of H is at most $O(\frac{\log n}{\log \log n})$. Given a node u and an integer k, we denote by H_k^u the set of nodes that can be reached by a directed path of length at most k in the subgraph H.

Lemma 3 (Useful Node in $O(\log \log n)$-Neighborhood). *With probability $1 - o(1)$, for each non-useful node u, at least one useful node will be included in H_K^u for some $K = O(\log \log n)$ following the constant out-degree protocol.*

Proof. We shall use Lemma 1 [8, Lemma 5.8] to prove this result. We condition on the same property $\mathcal{P}$ on G_m^n as in Lemma 1. In the proof of [8, Lemma 5.8], the authors consider a restricted version of PUSH strategy on G_m^n with one piece starting at a fixed node $u \in V$. We show that there is a correspondence between their restricted piece streaming process and our outgoing neighbor selection process. Note that in the proof of [8, Lemma 5.8], the authors assume that each node performs only PUSH and each node is active for only 3 rounds after it start downloading. With the memory property, this means the node contacts three random distinct neighbors in those 3 rounds (if the node has degree at least 3); this corresponds to the selection of $D = 3$ outgoing neighbors uniformly at random in COP.

Hence, the single rumor spreading process in [8, Lemma 5.8] can be coupled with our outgoing neighbor selection phase such that if a node v becomes downloading after $3K$ rounds in [8, Lemma 5.8], then there is a path from u to v of length K such that the next node in the path is an outgoing neighbor of the previous node selected by COP. With this coupling argument, we know that conditioning on property $\mathcal{P}$, with probability $1 - n^{-\Omega(n)}$, a useful node will be included in H_K^u for some $K = O(\log \log n)$. By union bound on all nodes in V, the conditional probability that this holds for all H_K^u's is at least $1 - n^{-\Omega(n)} \cdot n$. Since property $\mathcal{P}$ holds with probability at least $1 - o(1)$, our result follows.

Lemma 4 (Distances Between Useful Nodes). *With probability $1 - o(1)$, for each useful node u, there exists a path of length $O(\frac{\log n}{\log \log n})$ between u and 1 such that every second node in the path is fast.*

Proof of **Theorem 4:** By Lemma 3 and 4, we know that with probability $1 - o(1)$, each non-useful node can reach a useful node by a path of length at most $O(\log \log n)$ and all useful nodes are connected to node v_1 by paths of length

at most $O(\frac{\log n}{\log\log n})$ in H, which implies that the underlying undirected graph of H has diameter at most $O(\log\log n) + O(\frac{\log n}{\log\log n}) = O(\frac{\log n}{\log\log n})$. By Theorem 2, the finishing time of streaming B pieces in a PA-graph G_m^n using COP with $D = 3$ is at most $O(B + \log n)$ with probability $1 - o(1) - O(\frac{1}{n}) = 1 - o(1)$. □

4.2 Lower Bound for the Stopping Time

We have shown that streaming B pieces in a PA-graph G_m^n using COP with $D = 3$ requires at most $O(B + \log n)$ rounds. However, the PA-graph and the subgraph H of the PA-graph have diameter $O(\frac{\log n}{\log\log n})$, which is $o(\log n)$. Note that by Fact 2, each node v will become finished after $O(B + \frac{\log n}{\log\log n})$ with probability $1 - \exp(-\frac{\log n}{\log\log n})$. However, since the failure probability is $\omega(\frac{1}{n})$ and there are n nodes, we cannot use union bound to show that w.h.p., all nodes will be finished after $O(B + \frac{\log n}{\log\log n})$ rounds. Indeed, we show that the finishing time of streaming B pieces in a PA-graph G_m^n using COP with $D = 3$ needs $\Omega(B + \log n)$ rounds, which implies that the upper bound we have proved in the last section is tight and perfect pipelining is achieved. More specifically, we prove the following result.

Theorem 5 (Lower Bound For COP in PA-Graphs). *With probability $1 - o(1)$, COP with any constant parameter D needs $\Omega(B + \log n)$ rounds to get all nodes in a PA-graph G_m^n finished.*

Proof. It is obvious that if the protocol needs $\Omega(\log n)$ rounds to spread 1 piece to all nodes in G_m^n, then $\Omega(B + \log n)$ rounds are required for the same protocol to stream B pieces to all nodes in G_m^n. Hence, we only consider the case when $B = 1$ in the following proof.

By the proof of [8, Theorem 3.3], we know that with probability $1 - o(1)$, there are $\Omega(n)$ edges in G_m^n whose incident nodes are of degree at most $m + c$, where c is a constant. We call those edges *link*s and let $L \subseteq E$ be the collection of links, where $|L| = \Omega(n)$. Note that for any two edges e_1 and e_2 in L, e_1 and e_2 may have common endpoints or the endpoints of e_1 and e_2 may have common neighbors. To avoid this and make the proof simpler, we remove one of those two edges if such a situation happens. Let $\tilde{L}$ be the collection of links after the removals. Since each endpoint of a link is of degree at most $m + c$, we conclude that $|\tilde{L}| \geq \frac{1}{(m+c-1)^2+1}|L| = \Omega(n)$ (to keep one edge, at most $(m + c - 1)^2$ edges will be deleted). Note that any two links in $\tilde{L}$ have no common endpoint and the endpoints of any two edges have no common neighbor.

Fix one link in $\tilde{L}$, denoted by (u, v), u will choose v as an outgoing neighbor with probability at least $\frac{D}{m+c}$ and v will choose u as an outgoing neighbor also with probability at least $\frac{D}{m+c}$, independently in COP. Hence for each of those links, with constant probability, the two endpoints of the link will choose each other as outgoing neighbor in the protocol, independently for all links in $\tilde{L}$. If that happens, we call the link *preserved* in H. By Chernoff bound, we know that with probability $1 - \exp(-\Omega(n))$, there are $\Omega(n)$ preserved links.

Fix one pair of nodes (u, v) that are connected by a preserved link, in each round, if the two nodes are waiting, then with probability at least $(\frac{1}{D})^2(1 - \frac{1}{D})^{2(D-1)} = \delta$, they will remain waiting. Note that δ is a constant. The probability that (u, v) remains waiting for $\frac{\log n}{2 \ln \delta^{-1}}$ is therefore at least $n^{-\frac{1}{2}}$. Since each of those pairs are disjoint and the choices of communications are made independently, the probability that none of the $\Omega(n)$ pairs remains waiting after $\frac{\log n}{2 \ln \delta^{-1}}$ is at most $(1 - n^{-\frac{1}{2}})^{\Omega(n)} \leq \exp(-n^{-\frac{1}{2}}\Omega(n)) \leq \exp(-\Omega(n^{\frac{1}{2}}))$. Hence with probability $1 - o(1) - \exp(-\Omega(n)) - \exp(-\Omega(n^{\frac{1}{2}})) = 1 - o(1)$, COP needs $\Omega(\log n)$ rounds to spread one piece to all nodes in G_m^n and $\Omega(B + \log n)$ rounds to stream B pieces to all nodes in G_m^n using COP with any constant parameter D.

4.3 Improving the Upper Bound

It is established by Doerr et al. [8] that if each node in G_m^n avoid repeated neighbor initiation from the previous $M \geq 2$ rounds, the upper bound of the finishing time of spreading one piece in G_m^n can be improved to $O(\frac{\log n}{\log \log n})$, which is tight up to a constant factor. Note that with $M = 2$ and $D = 3$, the pieces streaming phase of COP is the same as round-robin and by Fact 1, the next theorem follows immediately.

Theorem 6. *For all $B \geq 1$, the finishing time for streaming B pieces starting from any source node in a PA-graph G_m^n using COP with round-robin pieces streaming phase is at most $O(B + \frac{\log n}{\log \log n})$ with probability $1 - o(1)$.*

5 Lower Bound

In Section 4 we show that perfect pipelining can be achieved using protocols based on the basic PUSH-PULL protocol when streaming multiple pieces in directed random graphs and preferential attachment graphs. In this section we give a class of D-regular graphs, where $\omega(1) \leq D \leq o(n)$, with diameter d and show that perfect pipelining is impossible in those regular graphs for any protocols in which communication partners are chosen uniformly at random in each round. We show that streaming B pieces from one source node in those regular graphs requires at least $\Omega(D(B + d))$ rounds, no matter how the pieces are chosen to be transmitted. We denote those protocols by PUSH-PULL based protocols. Note that by the upper bound proved by [1], our lower bound is actually tight.

The Necklace Graph. A D-regular necklace graph $N(V, E)$ with n nodes consists of $\ell := \frac{n}{D+1}$ units $U_1, U_2, \ldots, U_\ell$. Each unit U_i for $i \in [\ell]$ contains $D + 1$ nodes, numbered by $1, 2, \ldots, D + 1$. Within a unit U_i, there is an edge between each pair of nodes except the pair $\{1, D + 1\}$. That is, the subgraph of $N(V, E)$ induced by nodes in U_i is $K_{D+1} \backslash 1, D + 1$. For each $i \in \{1, 2, \ldots, \ell - 1\}$, the two units U_i and U_{i+1} are connected via an edge between node $D + 1$ of U_i and node 1 of U_{i+1}. Units U_ℓ and U_1 are connected via an edge between node $D + 1$ of $U_{\ell+1}$ and node 1 of U_1.

Theorem 7 (Lower Bound for PUSH-PULL). *Let D be an integer such that $\omega(1) \leq D \leq o(n)$. Let $N(V, E)$ be a D-regular necklace graph with $|V| = n$ and diameter d. Then for all $B \geq 1$, the finishing time for streaming B pieces starting from one source node in $N(V, E)$ using any PUSH-PULL based protocol is at least $\Omega(D(B + d))$ with probability at least $1 - o(1)$.*

For space reasons, we put its proof in the full version. Theorem 7 indicates that perfect pipelining is impossible when streaming multiple pieces in necklace graphs using any PUSH-PULL based protocol.

References

1. Arthur, D., Panigrahy, R.: Analyzing bittorrent and related peer-to-peer networks. In: SODA 2006, pp. 961–969. ACM, New York (2006)
2. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science 286, 509–512 (1999)
3. Bollobás, B., Riordan, O.: The diameter of a scale-free random graph. Combinatorica 24(1), 5–34 (2004)
4. Censor-Hillel, K., Haeupler, B., Kelner, J., Maymounkov, P.: Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In: STOC 2012. ACM, New York (2012)
5. Chan, H.-L., Lam, T.-W., Wong, P.W.H.: Efficiency of data distribution in bitTorrent-like systems. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 378–388. Springer, Heidelberg (2007)
6. Deb, S., Médard, M., Choute, C.: Algebraic gossip: a network coding approach to optimal multiple rumor mongering. IEEE/ACM Trans. Netw. 14(SI), 2486–2507 (2006)
7. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: PODC 1987. ACM Press, New York (1987)
8. Doerr, B., Fouz, M., Friedrich, T.: Social networks spread rumors in sublogarithmic time. In: STOC 2011. ACM, New York (2011)
9. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. In: Asano, T., Imai, H., Ibaraki, T., Nishizeki, T. (eds.) SIGAL 1990. LNCS, vol. 450, pp. 128–137. Springer, Heidelberg (1990)
10. Fountoulakis, N., Huber, A., Panagiotou, K.: Reliable broadcasting in random networks and the effect of density. In: INFOCOM 2010, pp. 2552–2560. IEEE Press, Piscataway (2010)
11. Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: STACS 2011, vol. 9, pp. 57–68. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl (2011)
12. Haeupler, B.: Analyzing network coding gossip made easy. In: STOC 2011, pp. 293–302. ACM, New York (2011)
13. Karp, R., Schindelhauer, C., Shenker, S., Vocking, B.: Randomized rumor spreading. In: FOCS 2000, p. 565. IEEE Computer Society, Washington, DC (2000)
14. Kempe, D., Kleinberg, J., Demers, A.: Spatial gossip and resource location protocols. In: STOC 2001, pp. 163–172. ACM, New York (2001)
15. Kempe, D., Kleinberg, J.M.: Protocols and impossibility results for gossip-based communication mechanisms. In: FOCS 2002, pp. 471–480. IEEE Computer Society, Washington, DC (2002)
16. Pittel, B.: On spreading a rumor. SIAM J. Appl. Math. 47(1) (March 1987)

The Shortest Path Game: Complexity and Algorithms*

Andreas Darmann[1], Ulrich Pferschy[2], and Joachim Schauer[2]

[1] University of Graz, Institute of Public Economics, Universitaetsstr. 15, 8010 Graz, Austria
andreas.darmann@uni-graz.at
[2] University of Graz, Department of Statistics and Operations Research, Universitaetsstr. 15, 8010 Graz, Austria
{pferschy,joachim.schauer}@uni-graz.at

Abstract. In this work we address a game theoretic variant of the shortest path problem, in which two decision makers (agents/players) move together along the edges of a graph from a given starting vertex to a given destination. The two players take turns in deciding in each vertex which edge to traverse next. The decider in each vertex also has to pay the cost of the chosen edge. We want to determine the path where each player minimizes its costs taking into account that also the other player acts in a selfish and rational way. Such a solution is a subgame perfect equilibrium and can be determined by backward induction in the game tree of the associated finite game in extensive form.

We show that finding such a path is PSPACE-complete even for bipartite graphs both for the directed and the undirected version of the game. On the other hand, we can give polynomial time algorithms for directed acyclic graphs and for cactus graphs in the undirected case. The latter is based on a decomposition of the graph into components and their resolution by a number of fairly involved dynamic programming arrays.

Keywords: shortest path problem, game theory, computational complexity, cactus graph.

1 Introduction

We are given a directed graph $G = (V, A)$ with vertex set V and arc set A with positive costs $c(u, v)$ for each arc $(u, v) \in A$ and two designated vertices $s, t \in V$. The aim of Shortest Path Game is to find a directed path from s to t in the following setting: The game is played by two players (or agents) A and B who start in s and always move together along arcs of the graph. In each

* Ulrich Pferschy and Joachim Schauer were supported by the Austrian Science Fund (FWF): [P 23829-N13]. Andreas Darmann was supported by the Austrian Science Fund (FWF): [P 23724-G11]. We would like to thank Christian Klamler (University of Graz) for fruitful discussions and valuable comments.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 39–53, 2014.

vertex the players take turns to select the next vertex to be visited among all neighboring vertices of the current vertex with player A taking the first decision in s. The player deciding in the current vertex also has to pay the cost of the chosen arc. Each player wants to minimize the total arc costs it has to pay. The game continues until the players reach the destination vertex t.Later, we will also consider the same problem on an undirected graph $G = (V, E)$ with edge set E which is quite different in several aspects.

To avoid that the players get stuck at some point, we restrict the players in every decision to choose an arc (or edge, in the undirected case) which still permits a feasible path from the current vertex to the destination t.

(R1) No player can select an arc which does not permit a path to vertex t.

In classical game theory the above scenario can be seen as a finite game in extensive form. All feasible decisions for the players can be represented in a game tree, where each node corresponds to the decision of a certain player in a vertex of the graph G.

The standard procedure to determine equilibria in a game tree is *backward induction* (see (Osborne, 2004, ch. 5)). This means that for each node in the game tree, whose child nodes are all leaves, the associated player can reach a decision by simply choosing the best of all child nodes w.r.t. their allocated total cost, i.e. the cost of the corresponding path in G attributed to the current player. Then these leaf nodes can be deleted and the pair of costs of the chosen leaf is moved to its parent node. In this way, we can move upwards in the game tree towards the root and settle all decisions along the way.

This backward induction procedure implies a strategy for each player, i.e. a rule specifying for each node of the game tree associated with this player which arc to select in the corresponding vertex of G. *Always choose the arc according to the result of backward induction.* Such a strategy for both players is a *Nash equilibrium* and also a so-called *subgame perfect equilibrium* (a slightly stronger property), since the decisions made in the underlying backward induction procedure are also optimal for every subtree.[1]

The outcome, if both players follow this strategy, is a unique path from s to t in G corresponding to the unique subgame perfect equilibrium (SPE) which we will call ***spe*-path**. A *spe*-path for Shortest Path Game is the particular solution in the game tree with minimal cost for both selfish players under the assumption that they have complete and perfect information of the game and know that the opponent will also strive for its own selfish optimal value.

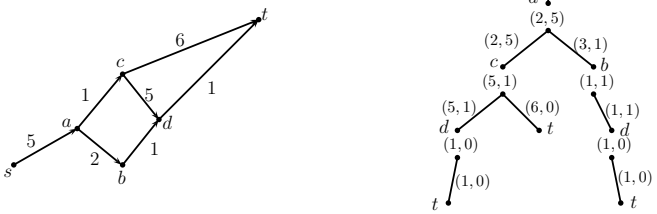
Clearly, such a *spe*-path path can be computed in exponential time by exploring the full game tree. It it is the main goal of this paper to study the complexity

[1] In order to guarantee a unique solution of such a game and thus a specific subgame perfect Nash equilibrium, we have to define a tie-breaking rule. We will use the "optimistic case", where in case of indifference a player chooses the option with lowest possible cost for the other player. If both players have the same cost, the corresponding paths in the graph are completely equivalent. Assigning arbitrary but fixed numbers to each vertex in the beginning, e.g. $1, \ldots, n$, we choose the path to a vertex with lowest vertex number.

status of finding this *spe*-path. In particular, we want to establish the hardness of computation for general graphs and identify special graph classes where a *spe*-path can be found in polynomial time.

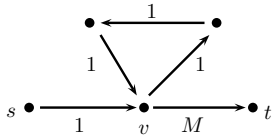
Note that in general game theory one considers only outcomes of strategies and their payoffs, i.e. costs of paths from s to t in our scenario. In this paper we will consider in each node of the game tree the cost for each player for moving from the corresponding vertex v of G towards t, since the cost of the path from s to v does not influence the decision in v. This allows us to solve identical subtrees that appear in multiple places of the game tree only once and use the resulting subpath of the *spe*-path on all positions.

Consider the following example: The *spe*-path is determined by backward induction and represented by ordered pairs of cost values (x, y) meaning that the decider in a given vertex has to pay a total value of x whereas the follower has to pay a value of y.



In this setting finding the *spe*-path for the two players is not an optimization problem as dealt with in combinatorial optimization but rather the identification of two sequences of decisions for the two players fulfilling a certain property in the game tree.

Note that there is a conceptual problem in this finite game model. There may occur cases where the game is infinite as illustrated by the following example. Player B has to decide in vertex v whether to pay the cost $M \gg 2$ or enter the cycle of length 3. In the latter case, the players move along the cycle and then A has to decide in v with the same two options as before for player B. In order to avoid paying M both players may choose to enter the cycle whenever it is their turn to decide in v leading to an infinite game.



Since there is no concept in game theory to construct an equilibrium for an infinite game in extensive form (only bargaining models such as the Rubinstein bargaining game are well studied, cf. (Osborne, 2004, ch. 16)) we have to impose reasonable conditions to guarantee finiteness of the game. An obvious restriction would be to rule out any cycle by requiring that each vertex may be visited at most once. Indeed, a cycle of even length can not be seen as a reasonable choice for any

player since it only increases the total cost of both players. Thus, a *spe*-path in a finite game will never contain a cycle of even length. However, in the above example it would be perfectly reasonable for B to enter the cycle of *odd* length and thus switch the role of the decider in v. Therefore, a second visit of a vertex may well make sense. However, if also A enters the cycle in the next visit of v, two rounds through the odd cycle constitute a cycle of even length which we rejected before. Based on these arguments we will impose the following restriction:

(R2) The players can not select an arc which implies necessarily a cycle of even length.

Note that (R2) implies that an odd cycle may be part of the solution path, but it may not be traversed twice since this would constitute a cycle of even length. In the remainder of the paper we use "cycle" for any closed walk, also if vertices are visited multiple times. It also follows that each player can decide in each vertex at most once and any arc can be used at most once by each agent.

1.1 Related Literature

A closely related game is known as Geography (see Schaefer (1978)). It is played on a directed graph with no costs. Starting from a designated vertex $s \in V$, the two players move together and take turns in selecting the next vertex. The objective of the game is quite different from Shortest Path Game, namely, the game ends as soon as the players get stuck in a vertex and the player who has no arc left for moving on loses the game. Moreover, there is a further restriction that in Geography each arc may be used at most once.

Schaefer (1978) already showed PSPACE-completeness of Geography. Lichtenstein and Sipser (1980) proved that the variant Vertex Geography, where each vertex cannot be visited more than once, is PSPACE-complete for planar bipartite graphs of bounded degree. This was done as an intermediate step for showing that Go is PSPACE-complete. Fraenkel and Simonson (1993) gave polynomial time algorithms for Geography and Vertex Geography when played on directed acyclic graphs. In Fraenkel et al. (1993) it was proved that also the undirected variant of Geography is PSPACE-complete. However, if restricted to bipartite graphs they provided a polynomial time algorithm by using linear algebraic methods on the bipartite adjacency matrix of the underlying graph. Note that this result is in contrast to the PSPACE-completeness result of Section 3 for Shortest Path Game on bipartite undirected graphs. Bodlaender (1993) showed that directed Vertex Geography is linear time solvable on graphs of bounded treewidth. For directed Geography such a result was shown under the additional restriction that the degree of every vertex is bounded by a constant - the unrestricted variant however is still open.

Recently, the *spe*-path of Shortest Path Game was used in Darmann et al. (2013) as a criterion for sharing the cost of the shortest path (in the classical sense) between two players. A different variant of two players taking turns in the decision on a combinatorial optimization problem and each of them optimizing its own objective function was recently considered for the Subset Sum problem by Darmann et al. (2014).

1.2 Our Contribution

We introduce the concept of *spe*-path resulting from backward induction in a game tree with full information, where two players pursue the optimization of their own objective functions in a purely rational way. Thus, a solution concept for the underlying game is determined which incorporates in every step all anticipated decisions of future steps. The main question we ask in this work concerns the complexity status of computing such a *spe*-path, if the game consists in the joint exploration of a path from a source to a sink. We believe that questions of this type could be an interesting topic also for other problems on graphs and beyond.

We can show in Section 2.1 that for *directed graphs* Shortest Path Game is PSPACE-complete even for bipartite graphs, while for acyclic directed graphs a linear time algorithm is given in Section 2.2. These results are in line with results from the literature for the related game Geography.

On the other hand, for *undirected graphs* we can show in Section 3 that again Shortest Path Game is PSPACE-complete even for bipartite graphs by a fairly complicated reduction from Quantified 3-Sat while the related problem Geography is polynomially solvable on undirected bipartite graphs. This surprising difference shows that finding paths with minimal costs can lead to dramatically harder problems than paths concerned only with reachability.

In Section 4 we give a fairly involved algorithm to determine the *spe*-path on undirected cactus graphs in polynomial time. It is based on several dynamic programming arrays and a partitioning of the graph into components. Using an advanced auxiliary data structure, its running time can be bounded by $O(n \log n)$.

2 Spe-Paths for Shortest Path Game on Directed Graphs

The following decision problem basically asks whether the unique subgame perfect equilibrium remains below given cost bounds C_A and C_B.

Shortest Path Game DEC:
Given a weighted graph G and two positive values C_A, C_B, does the *spe*-path in the above game tree yield costs $c(A) \leq C_A$ and $c(B) \leq C_B$?

2.1 PSPACE-completeness

The PSPACE-completeness of Shortest Path Game on general graphs can be shown by constructing an instance of Shortest Path Game, such that the *spe*-path decides the winner of Vertex Geography. In fact, we can give an even stronger result with little additional effort.

Theorem 1. SHORTEST PATH GAME DEC *is* PSPACE*-complete even for bipartite directed graphs.*

Proof. Inclusion in PSPACE can be shown easily by considering that the height of the game tree is bounded by $2|A|$. Hence, we can determine the *spe*-path in polynomial space by exploring the game tree in a DFS-way. In every node currently under consideration we have to keep a list of decisions (i.e. neighboring vertices in the graph) still remaining to be explored and the cost of the currently preferred subpath among all the options starting in this node that were already explored. By the DFS-processing there are at most $2|A|$ nodes on the path from the root to the current node for which this information has to be kept.

We provide a simple reduction from VERTEX GEOGRAPHY, which is known to be PSPACE-complete for planar bipartite directed graphs where the in-degree and the out-degree of a vertex is bounded by two and the degree is bounded by three (Lichtenstein and Sipser (1980)). For a given instance of VERTEX GEOGRAPHY we construct an instance of SHORTEST PATH GAME, such that the *spe*-path decides the winner of VERTEX GEOGRAPHY: Given the planar bipartite directed graph $G = (V, A)$ of VERTEX GEOGRAPHY with starting vertex s, we can two-color the vertices of V because G is bipartite. For the two-coloring, we use the colors red and green and color the vertices such that s is a green vertex. We create a new graph H for SHORTEST PATH GAME DEC as follows: First we assign a cost ε to every arc $e \in E$. Then we introduce a new vertex t which we color red, and an arc of weight $M \gg 0$ from each green vertex to t. Next, introduce a green vertex z and an arc of weight $M \gg 0$ from each red vertex to z. Finally, introduce an arc of cost ε from z to t. The constants C_A and C_B are set to $C_A = 2\,\varepsilon$ and $C_B = M$. This means that a "yes"-instance corresponds to player A winning VERTEX GEOGRAPHY. It is not hard to see that H is a bipartite directed graph. Note that since the constructed graph is bipartite the rule of VERTEX GEOGRAPHY saying that each arc can be used at most once is equivalent to (R2) in SHORTEST PATH GAME.

Whenever a player gets stuck in a vertex playing VERTEX GEOGRAPHY, it would be possible to continue the path in H towards t (possibly via z) by choosing the arc of cost M. On the other hand, both players will avoid to use such a costly arc as long as possible and only one such arc will be chosen. Thus, the *spe*-path for SHORTEST PATH GAME will incur $\leq 2\,\varepsilon$ cost to one player and exactly cost M to the other player, who is thereby identified as the loser of VERTEX GEOGRAPHY. This follows from the fact that if both players follow the *spe*-path, they can anticipate the loser. If it is A, then this player will immediately go from s to t. If it is player B, then A will choose an arc with cost ε, then B will go to z paying M, and A from z to t at cost ε. □

Note that the result of Theorem 1 also follows from Theorem 3 be replacing each edge in the undirected graph by two directed arcs. However, we believe that the connection to GEOGRAPHY established in the above proof is interesting in its own right.

2.2 Directed Acyclic Graphs

If the underlying graph G is acyclic we can devise a strongly polynomial time dynamic programming algorithm. It is related to a dynamic programming scheme for the longest path problem in acyclic directed graphs. For each vertex $v \in V$ we define $S(v) := \{u \mid (v,u) \in A\}$ as the set of successors of v. Then we define the following two dynamic programming arrays for each vertex v:

$d_1(v)$: minimal cost to go from v to t for the player deciding at v.
$d_2(v)$: minimal cost to go from v to t for the player **not** deciding at v.

The two arrays are initialized as follows:
$d_1(t) = d_2(t) = 0$, $d_1(v) = d_2(v) = \infty$ for all $v \in V, v \neq t$.

Algorithm 1. Optimal strategies of the shortest path game on acyclic graphs

1: **repeat**
2: find $v \in V$ with $d_1(v) = \infty$ such that $d_1(u) \neq \infty$ for all $u \in S(v)$
3: let $u' := \arg\min\{c(v,u) + d_2(u) \mid u \in S(v)\}$
4: $d_1(v) := c(v,u') + d_2(u')$
5: $d_2(v) := d_1(u')$
6: **until** $d_1(s) \neq \infty$

Starting from the destination vertex t and moving backwards in the graph we iteratively compute the values of $d_i(v)$. Note that each such entry is computed only once and never updated later. In each iteration of the **repeat**-loop one entry $d_1(v)$ is reduced from ∞ for some vertex $v \in V$. Thus the algorithm terminates after $|V| - 1$ iterations, but we have to show that in each iteration a vertex v is found in line 2. Assume otherwise: If no vertex v remains with $d_1(v) = \infty$, then also $d_1(s) \neq \infty$ and we would have stopped the algorithm before. If for all vertices $v \in V$ with $d_1(v) = \infty$ there exists a vertex $u \in S(v)$ with $d_1(u) = \infty$, then we could apply the same argument to u. Thus, also u has a successor u_s with $d_1(u_s) = \infty$. Iterating this argument we can build a path from v to u and to u_s and so on. By construction, this path never ends, because otherwise the last vertex of the path would fulfill the conditions of line 2. But this means that the path is a cycle in contradiction to the assumption that G is acyclic. Taking a closer look at the running time details we can state with same additional effort:

Theorem 2. *The* spe-*path of* SHORTEST PATH GAME *on acyclic directed graphs can be computed in* $O(|A|)$ *time.*

3 Spe-Paths for Shortest Path Game on Undirected Graphs

We will provide a reduction from Quantified 3-Sat which is known to be PSPACE-complete (Stockmeyer and Meyer (1973)).

Definition (Quantified 3-Sat)**:**

> GIVEN: Set $X = \{x_1, \ldots, x_n\}$ of variables and a quantified Boolean formula $F = (\exists x_1)(\forall x_2)(\exists x_3) \ldots (\forall x_n)\, \phi(x_1, \ldots, x_n)$ where ϕ is a propositional formula over X in 3-CNF (i.e., in conjunctive normal form with exactly three literals per clause).
> QUESTION: Is F true?

Let $C_1, \ldots, C_m$ denote the clauses that make up ϕ, i.e., $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots C_m$, where each C_i, $1 \leq i \leq m$, contains exactly three literals. Quantified 3-Sat can be interpreted as the following game (cf. Fraenkel and Goldschmidt (1987)): There are two players (the existential- and the universal-player) moving alternately, starting with the existential-player. The i-th move consists of assigning a truth value ("true" or "false") to variable x_i. After n moves, the existential-player wins if and only if the produced assignment makes ϕ true.

3.1 PSPACE-completeness

The following result shows a notable difference between Shortest Path Game and Geography, since Fraenkel et al. (1993) showed that Geography is polynomially solvable on undirected bipartite graphs (while PSPACE-complete on general undirected graphs).

Theorem 3. Shortest Path Game DEC *on undirected graphs is* PSPACE*-complete, even for bipartite graphs.*

Proof. Inclusion in PSPACE follows from a similar argument as in the proof of Theorem 1. Given an instance $\mathcal{Q}$ of Quantified 3-Sat we construct an instance $\mathcal{S}$ of Shortest Path Game by creating an undirected graph $G = (V, E)$ as follows. The vertices are 2-colored (using the colors red and green) to show that G is bipartite. To construct G, we introduce (see the figure below):

- green vertices d, p, r, red vertices w, q, t
- edges $\{p, q\}$, $\{r, t\}$, $\{w, d\}$ and $\{d, t\}$
- for each clause C_j, a green vertex c_j

– for each even i, $2 \leq i \leq n$, an "octagon", i.e.,
  - red vertices $v_{i,0}, v_{i,2}, v_{i,4}, v_{i,6}$
  - green vertices $v_{i,1}, v_{i,3}, v_{i,5}, v_{i,7}$
  - edges $\{v_{i,\ell}, v_{i,\ell+1}\}$, $0 \leq \ell \leq 6$, and edge $\{v_{i,7}, v_{i,0}\}$
– for each odd i, $1 \leq i \leq n$, a "hexagon", i.e.,
  - green vertices $v_{i,0}, v_{i,2}, v_{i,4}$
  - red vertices $v_{i,1}, v_{i,3}, v_{i,5}$
  - edges $\{v_{i,\ell}, v_{i,\ell+1}\}$, $0 \leq \ell \leq 4$, and edge $\{v_{i,5}, v_{i,0}\}$

In order to connect these parts, we introduce:

– for each even i, $2 \leq i \leq n$
  - a green vertex u_i
  - edges $\{v_{i-1,3}, u_i\}$, $\{u_i, v_{i,0}\}$ and the edges $\{v_{i,2}, r\}$, $\{v_{i,6}, r\}$
  - edge $\{v_{i,4}, v_{i+1,0}\}$, where $v_{n+1,0} := p$
  - for each clause C_j, the edge $\{v_{i,2}, c_j\}$ if $x_i \in C_j$ and $\{v_{i,6}, c_j\}$ if $\bar{x}_i \in C_j$
– for each odd i, $1 \leq i \leq n$
  - the edges $\{v_{i,1}, r\}$, $\{v_{i,5}, r\}$
  - for each clause C_j, the edge $\{v_{i,1}, c_j\}$ if $x_i \in C_j$ and $\{v_{i,5}, c_j\}$ if $\bar{x}_i \in C_j$
– for each j, $1 \leq j \leq m$,
  - edges $\{q, c_j\}$ and $\{w, c_j\}$

Abusing notation, for $1 \leq i \leq n$, let $x_i := \{v_{i,0}, v_{i,1}\}$, and $\bar{x}_i := \{v_{i,0}, v_{i,5}\}$ if i is odd resp. $\bar{x}_i := \{v_{i,0}, v_{i,7}\}$ if i is even. I.e., we identify a literal with an edge of the some label. For illustration, in the figure below we assume $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

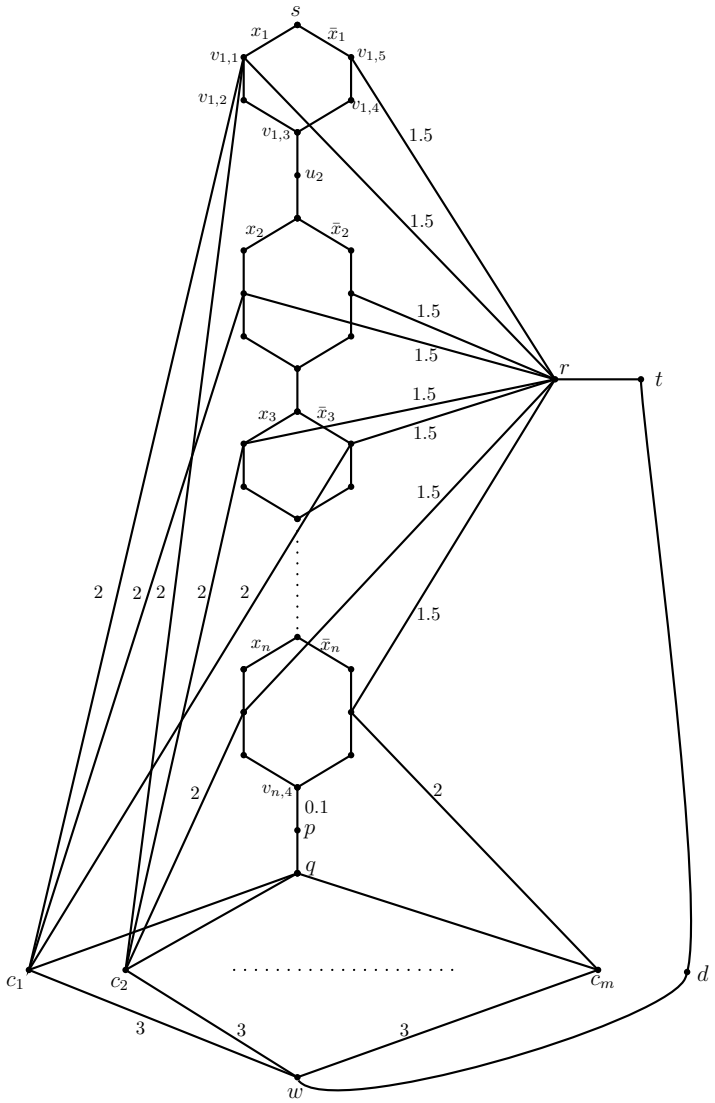
Finally, we define the edge costs.[2] We start with the edges emanating from vertex w: The cost of edge $\{w, d\}$ is $c\{w, d\} = 0$, all other edges emanating from w have cost 3. The edge $\{r, t\}$ has cost $c(\{r, t\}) = 0$, while each other edge emanating from r has cost 1.5. The edge $\{v_{n,4}, p\}$ has cost $c(\{v_{n,4}, p\}) = 0.1$. For each $1 \leq j \leq m$, the edges emanating from vertex c_j which do not correspond to $\{c_j, w\}$ or $\{c_j, q\}$ have cost 2. The remaining edges have zero cost.

Note that from the fact that each edge connects a green with a red vertex, it immediately follows that G is a bipartite graph. Now, in $\mathcal{S}$ the players A, B try to find a path from s to t in the graph G, each one aiming at minimizing her own total cost.

Claim. $\mathcal{Q}$ is a "yes"-instance of Quantified 3-Sat $\Leftrightarrow$ Instance $\mathcal{S}$ of Shortest Path Game ends with A having to carry zero cost.

Proof of Claim. omitted.

[2] In the introduction edge costs were defined to be strictly positive. For simplicity we use zero costs in this proof, but these could be easily replaced by some small $\varepsilon > 0$.



4 Shortest Path Game on Undirected Cactus Graphs

In this section we will show that a cactus graph still allows a polynomial solution of Shortest Path Game. This is mainly due to a decomposition structure which allows to solve components of the graph to optimality independently from the solution in the remaining graph.

A *cactus graph* is a graph where each edge is contained in at most one simple cycle. Equivalently, any two simple cycles have at most one vertex in common. Considering Shortest Path Game, it is easy to see that the union of all simple paths from s to t constitutes a unique sequence of edges (which are bridges of

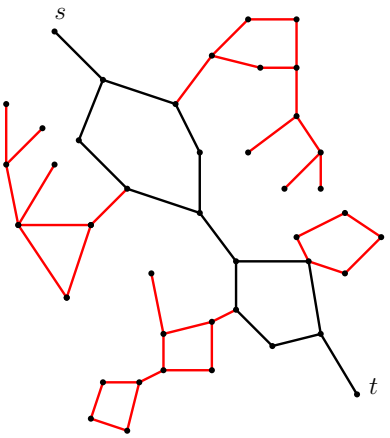


Fig. 1. Graph G with connection strip G' (in black) and branches $G \setminus G'$ (in red)

the graph) and simple cycles which defines a subgraph G', i.e. the *connection strip* between s and t. All other vertices of the graph are "dead end streets", i.e. edges and cycles branching off from G' (see Figure 1). In the *spe*-path vertices in $G \setminus G'$ could be included only to change the role of the decision maker in a vertex of G'. Clearly, any such deviation from G' must be a cycle rooted in some vertex of G'. Moreover, by (R2) only cycles (not necessarily simple) of odd length might be traversed in this way. This structural property gives rise to a preprocessing step where all vertices in $G \setminus G'$ are contracted into a *swap option* in a vertex $v \in G'$ with cost $(sw_d(v), sw_f(v))$ meaning that if the path of the two players reaches a certain vertex $v \in G'$, the current decider has the option to switch roles (by entering an odd cycle in $G \setminus G'$ rooted in v) at cost of $sw_d(v)$ for himself (the decider) and $sw_f(v)$ for the other player (the follower).

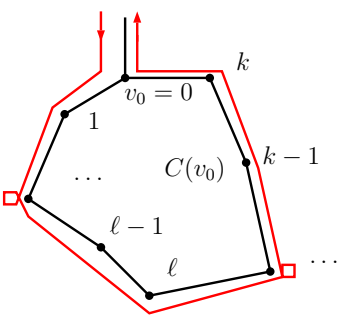
Our algorithm will first compute these swap costs by recursively traversing the components of $G \setminus G'$ in Section 4.1. Then, in the second step, the *spe*-path in G' is computed by moving backwards from t towards s in Section 4.2. Because of the space restriction, we will only describe the contraction of branches in algorithmic detail to give the reader a flavor of the required computation. The main part of the algorithm is sketched only briefly. Its details are quite involved and follow in some sense the technique described in Section 4.1.

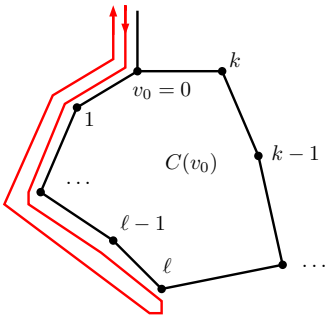
4.1 Contraction of the Branches

Consider a cycle $C(v_0)$ which is connected to the remaining graph only via v_0 and all other vertices of $C(v_0)$ have degree 2, i.e. all other edges and cycles incident to these vertices were contracted into swap options before. For simplicity of notation we refer to vertices by their index number and assume that $C(v_0) = C(0)$ consists of a sequence of vertices $0, 1, 2, \ldots, k-1, k, 0$.

There are four possibilities how to use the cycle for a swap: The players could enter the cycle by the edge $(0, 1)$ and go around the full length of the cycle

(possibly using additional swaps in vertices of the cycle) (see left figure below). Or after edge $(0,1)$ the players could move up to some vertex $\ell \in \{1,2,\ldots,k\}$, turn around by utilizing a swap option in ℓ and go back to 0 (see right figure below). In the latter case, the players can not use any additional swaps in vertices $1,\ldots,\ell-1$ (resp. $k,k-1,\ldots,\ell+1$) since in that case the swap vertex would be visited three times in violation of (R2). Thus, we have to distinguish in each vertex whether such a turn around is still possible or whether it is ruled but by a swap in a previously visited vertex of the cycle.





These two configurations can also be used in a laterally reversed way moving on the cycle in the different direction starting with the edge $(0,k)$ which yields four cases in total.

Let $D \in \{A,B\}$ be the decision maker in 0. We use a generic notation for dynamic programming arrays, where $d_P^{\pm}(i)$ denotes the cost of a certain path starting in vertex i and ending in a fixed specified vertex, with subscript $P \in \{d,f\}$, where $P = d$ signifies that the cost occurs for the player deciding in i and $P = f$ refers to the cost of the follower. Superscript $\pm \in \{+,-\}$ shows that the decider in i is equal to D if $\pm = +$, or whether the other player decides in i, i.e. $\pm = -$. We use cost $\top$ if a path is infeasible.

Following this system we use:
$tc_P^{\pm}(i)$: minimal cost to move from i back to 0, if a turn around is still possible.
$rc_P^{\pm}(i)$: minimal cost to move from i back to 0, if no turn around is possible and the path has to go around the cycle, i.e. visit vertices $i+1, i+2, \ldots, k, 0$, with possible swaps on the way. If one player decides to turn around at some vertex i, the cost of the path back towards vertex 0 is completely determined since no choices remain open. The corresponding costs are independent from D and will be recorded as $\text{path}_P(i)$ in analogy to above.

Now we can state the appropriate update recursion for the case where D chooses $(0,1)$ as a first edge. We go backwards along this path and settle the minimal costs for vertices $k, k-1, \ldots, 1$. The case where D moves into the other direction of the cycle is completely analogous and the final swap costs $sw(v_0)$ are given by the cheaper alternative.

The decision tree in any vertex i is depicted in Figure 2.

1. move on along the cycle to vertex $i+1$:
   $rc_d^+(i) := c(i,i+1) + rc_f^-(i+1)$, $rc_f^+(i) = rc_d^-(i+1)$
   $rc_d^-(k) := c(i,i+1) + rc_f^+(i+1)$, $rc_f^-(i) = rc_d^+(i+1)$

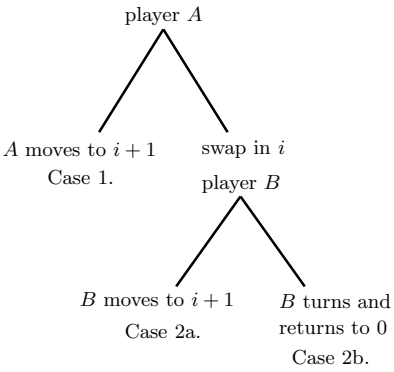


Fig. 2. Decision tree for player A deciding in vertex i

$tc_d^+(i) := c(i,i+1) + tc_f^-(i+1)$, $tc_f^+(i) = tc_d^-(i+1)$
$tc_d^-(k) := c(i,i+1) + tc_f^+(i+1)$, $tc_f^-(i) = tc_d^+(i+1)$

2. make a swap (if available): Then the other player has (at most) two possibilities and chooses the one with the lower cost between 2a. and 2b. (or the only feasible choice), which implies the cost for the decider in i.

2a. move on to vertex $i+1$:
$rc_f^+(i) = sw_f(i) + c(i,i+1) + rc_f^+(i+1)$, $rc_d^+(i) := sw_d(i) + rc_d^+(i+1)$
$rc_f^-(i) = sw_f(i) + c(i,i+1) + rc_f^-(i+1)$, $rc_d^-(i) := sw_d(i) + rc_d^-(i+1)$
$tc_f^+(i) := sw_f(i) + c(i,i+1) + rc_f^+(i+1)$, $tc_d^+(i) := sw_d(i) + rc_d^+(i+1)$
$tc_f^-(i) := sw_f(i) + c(i,i+1) + rc_f^-(i+1)$, $tc_d^-(i) := sw_d(i) + rc_d^-(i+1)$.

2b. turn around (if possible): Since the decider at the end of the return path in vertex 0 must be different from D, the feasibility of a turn around depends on the number of edges between 0 and i.
If i is even:
$tc_d^+(i) := sw_d(i) + \text{path}_f(i)$, $tc_f^+(i) := sw_f(i) + \text{path}_d(i)$
$tc_d^-(i) := \top$, $tc_f^-(i) := \top$
If i is odd:
$tc_d^+(i) := \top$, $tc_f^+(i) := \top$
$tc_d^-(i) := sw_d(i) + \text{path}_f(i)$, $tc_f^-(i) := sw_f(i) + \text{path}_d(i)$

Now the decider in vertex i can anticipate the potential decision of the other player in case 2., since the other player will choose the better outcome between cases 2a. and 2b. Hence, the decision maker in vertex i chooses the minimum between case 1. and case 2. independently for all four dynamic programming entries. This immediately implies the cost for the other player.

It remains to discuss the initialization of the arrays for $i = k$, i.e. the last vertex in the cycle and thus the first vertex considered in the recursion. To avoid the repetition of all three cases implied by a possible swap at vertex k, we add two artificial vertices $k+1$ and $k+2$ and three artificial edges $(k,k+1)$, $(k+1,k+2)$ and $(k+2,0)$ replacing the previous edge $(k,0)$. We set $c(k,k+1) = c(k,0)$ and the other two artificial edges have cost 0. It is easy to see that this extension of

the cycle does not change anything. Now we can start the recursive computation at vertex $k+2$ which has no swap option. We get:

$rc_d^+(k+2) := c(k+2,0)$, $rc_f^+(k+2) = 0$, $rc_d^-(k+2) := \top$, $rc_f^-(k+2) = \top$
$tc_d^+(k+2) := c(k+2,0)$, $tc_f^+(k+2) = 0$, $tc_d^-(k+2) := \top$, $tc_f^-(k+2) = \top$

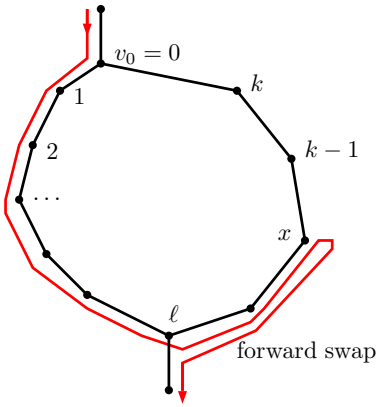
4.2 Main Part of the Algorithm

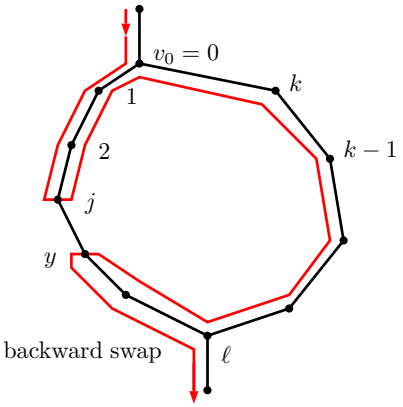
In the main part of the algorithm we determine the *spe*-path from s to t along the connection strip G' after contracting the reminder of the graph into swap options in G'. We traverse the connection strip backwards starting in t and moving up towards s. In the following we will focus on the computation of a *spe*-path for one cycle of this sequence. Each such cycle has two designated vertices which all paths from s to t have to traverse, an "upper vertex" v_0 through which every path starting in s enters the cycle, and a "lower vertex" v_ℓ through which all paths connecting the cycle with t have to leave the cycle.

If the decider in 0 takes the edge $(0,1)$ there are two main possibilities for the path from 0 to ℓ (the alternative case starting with edge $(0,k)$ is completely symmetric and the decider will finally take the better of the two options).

Case (i): The two players may move along the vertices $0,1,\ldots,\ell$, possibly with a few swaps on the way. After reaching ℓ, they may either exit the cycle or continue to $\ell+1,\ldots,x$, make a *forward swap* in x and return back via $x-1, x-2,\ldots$ back to ℓ and finally exit the cycle (see left figure below). As a special variant, they players may also never swap in some vertex x but go back to 0 thus traversing the full cycle and then taking the path $0,1,\ldots,\ell$ a second time.

Case (ii): As a second, more complicated possibility, the two players may also move along vertices $0,1,\ldots,j$, $j<\ell$, and then utilize a swap in j and return to 0. Then they are forced to move on from 0 to $k, k-1,\ldots,\ell$. After reaching ℓ they may either exit the cycle directly or they may also continue to $\ell-1,\ell-2,\ldots,y$ with $y>j$, make a *backward swap* in y and return via $y+1,\ldots,\ell$ where they finally exit the cycle (see right figure below).





To determine the *spe*-path in G' we move backwards through the cycle starting with the decision in v_ℓ and going back up to v_0. In each vertex j we consider the possibilities of moving on to $j+1$ or taking a swap and anticipating the further moves of the other player. To do so, the cost of potential paths from j to v_ℓ after a swap have to be computed beforehand in auxiliary arrays in the spirit of Section 4.1. In Case (ii) this turns out to be fairly complicated since the costs of the path from the current vertex j back up to v_0 and on to k, $k-1$, etc., downto v_ℓ with a possible backward swap have to be taken into account. Note that the turning point y of a backward swap is limited by j. Thus, we have to compute the cost of a backward swap for all bounds j.

In the initialization of all paths leading to v_ℓ we also have to take the *spe*-path from v_ℓ to t into account which was computed before. The correct combination of solutions of two adjacent cycles contains another subtle detail, since the joining vertex v_ℓ of the upper cycle can be traversed at most twice and thus only certain combinations are feasible.

While the main procedure can be done in linear time over all vertices of a cycle, the auxiliary array to compute the backward swap would be quadratic in a straightforward implementation. However, we can do better by utilizing an advanced data structure and thus reach the following statement:

Theorem 4. *The* spe*-path of* Shortest Path Game *on undirected cactus graphs can be computed in* $O(n \log n)$ *time.*

References

Bodlaender, H.: Complexity of path-forming games. Theoretical Computer Science 110(1), 215–245 (1993)

Darmann, A., Klamler, C., Pferschy, U.: Sharing the cost of a path. Technical report (2013), `http://ssrn.com/abstract=2287875` (submitted)

Darmann, A., Nicosia, G., Pferschy, U., Schauer, J.: The subset sum game. European Journal on Operational Research 233, 539–549 (2014)

Fraenkel, A.S., Goldschmidt, E.: PSPACE-hardness of some combinatorial games. Journal of Combinatorial Theory 46(1), 21–38 (1987)

Fraenkel, A.S., Scheinerman, E.R., Ullman, D.: Undirected edge geography. Theoretical Computer Science 112(2), 371–381 (1993)

Fraenkel, A.S., Simonson, S.: Geography. Theoretical Computer Science 110(1), 197–214 (1993)

Lichtenstein, D., Sipser, M.: Go is polynomial-space hard. Journal of the ACM 27(2), 393–401 (1980)

Osborne, M.J.: An Introduction to Game Theory. Oxford Univ. Press (2004)

Schaefer, T.J.: On the complexity of some two-person perfect-information games. Journal of Computer and System Sciences 16(2), 185–225 (1978)

Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proc. of the 5th Symp. on Theory of Computing, STOC 1973, pp. 1–9 (1973)

Online Scheduling of Unit Length Jobs with Commitment and Penalties

Stanley P.Y. Fung

Department of Computer Science, University of Leicester, Leicester, United Kingdom
pyf1@le.ac.uk

Abstract. We consider the online scheduling of unit length jobs with two models of commitment. In immediate notification, the acceptance of a job must be decided as soon as it is released. In immediate decision, the actual time slot allocated to the job must also be fixed at the job's arrival as well. Failure to honour a commitment will result in a penalty. The non-commitment version has been extensively studied. In this paper we give algorithms and lower bounds for the two models of commitment. For immediate decision, we give an $O(m(1+\rho)^{1/m})$-competitive algorithm where m is the number of machines and ρ is the penalty factor, and when m is large we give an $O(\log(1+\rho))$ upper bound. This is matched by a lower bound of $\Omega(\log \rho)$ on the competitive ratio. For immediate notification we give a lower bound of $\Omega(\log \rho / \log\log \rho)$. We also give some better bounds when $m = 1$ or when ρ is small. Finally we give considerations to the case of separate arrival and start times.

1 Introduction

The Model. The basic setting of our problem is as follows: there is a set of jobs, where each job j is specified by a triple $(r(j), d(j), w(j))$ representing its release time, deadline and weight respectively. Each job has length 1 and all release times and deadlines are integers; also, time is discrete and is divided into time slots of unit length. A *slot* is defined as a tuple (i, t) which is a combination of a machine i and a time step $[t, t+1)$. Thus, each job will fit into a slot and there will be no interruption while a job is being executed.

Jobs arrive online, so a job is only known to the algorithm upon its release time, at which point all information about the job is known. All jobs arriving at the same time step is known to the algorithm at the same time. The objective of the algorithm is to schedule these jobs on m identical machines so as to maximize the profit, i.e. the total weight of jobs completed before their deadlines. This is known as the *unit job scheduling* problem in the literature.

As the algorithms are online, they do not have knowledge of future information and have to make scheduling decisions based on what is known. This inevitably leads to suboptimal algorithms, and the standard way to analyze such online algorithms is *competitive analysis*. An algorithm (for a maximization problem) is *R-competitive* if the value returned by the algorithm is always at least $1/R$ times the value returned by an offline optimal algorithm, for any input instance.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 54–65, 2014.

In this paper we will use $\mathcal{OPT}$ to denote the offline optimal algorithm, and $\mathcal{ONL}$ to denote the online algorithm in question. Also, we use $\mathcal{OPT}(i,t)$ to denote the job scheduled by $\mathcal{OPT}$ at slot (i,t); similar definition holds for $\mathcal{ONL}(i,t)$.

In the above setting, whether a job will be completed on time is only known until its deadline is reached. The algorithm is not required to announce upfront whether it intends to complete the job. In many application areas where customer service is crucial, it is important to let the customers (users supplying the jobs) know as early as possible whether (or when) the job will be completed. Ideally this should be made known immediately, at the same time the job is presented; if the job is rejected the customer can then find another company to fulfill the job, and not be left in a situation where he/she only finds out that the job is not completed at the last minute (and therefore is too late to do anything about it). Such 'commitment' in job scheduling can be modelled in a number of ways, two of which we consider in this paper:

Immediate Notification (IMM NOTIF): When a job is released, the algorithm has to decide immediately whether to accept the job. An accepted job must be completed before its deadline. A job not accepted is gone forever. This model was introduced in [14].

Immediate Decision (IMM DEC): In addition to IMM NOTIF, here the exact time to execute an accepted job, and also the machine to execute the job (if there are more than one machine) must also be announced at the same time that the job is accepted, and this cannot be changed later. This model was introduced in [9].

While it would be ideal to always keep to one's commitments, unfortunately it is not uncommon for (for example) online shopping or delivery companies to delay or miss orders altogether. In our model, an accepted job can be *evicted*[1] later, but such eviction incurs a penalty that is proportional to the weight of the job evicted. This is in addition to losing the weight of the job. A job once evicted is gone and will not come back. In the case of IMM DEC, reallocating an accepted job to a different slot is not allowed, even after paying penalties, so the job is gone forever. The objective of the algorithm is then to maximize the total weight of all completed jobs, minus the sum of all penalties incurred. Observe that $\mathcal{OPT}$ does not pay penalties as clearly it can decide in advance whether to accept a job and when to schedule it, and to announce it upfront, without damaging its profit.

We use ρ to denote the *penalty factor*, where $\rho \geq 0$, and the penalty of evicting a job j of weight $w(j)$ is then equal to $\rho w(j)$. There are several motivations of studying such a penalty model. Incorporating a penalty factor allows us to study quantitatively the cost of breaking a commitment. It may be tempting to suggest that ρ should be at most 1, so that the penalty is at most the value of the job, and it may well be true in practice. While we do study small ρ values, we also allow ρ

[1] We use the term 'eviction' rather than 'preemption' to distinguish with the case where a job is interrupted in the middle of its execution (which will not happen in our setting.)

to grow arbitrarily large and then study how the competitive ratio depends on ρ; this is because when $\rho \to \infty$ it models the situation when eviction is not allowed, which was the case in earlier works. A finite but large ρ models an intermediate situation where it is very costly but not impossible to break a commitment. This penalty model also allows the study of scheduling weighted jobs, to model the different importance of jobs; in nonpreemptive models like [8] allowing weighted jobs would lead to trivial bad results.

The idea of giving penalties for not processing jobs is not new; for example there are a lot of work on scheduling with rejection (see e.g. [19] for a survey). Also, there are many works in management science and operations research concerning *quoted lead time*, i.e. it is the supplier, on receiving a demand, who quote a time when the order will be met, and typically the profit received is a non-increasing function of this quoted lead time; see e.g. [16]. Most of these were not studied with competitive analysis, but [17] studied a model where the profit decreases by a fixed amount per unit of added time (this is the penalty) but commitment is not breakable (called 100% reliable).

The need of immediate commitment and revocation penalties can also be found in mechanism design in auctions. In [6] an auction problem for unit sized time slots is considered. The auction must be truthful, i.e. players are incentivized to bid their true values. Existing algorithms for unit job scheduling [18] already satisfy this property, but [6] considered an additional requirement of *promptness*, which can be guaranteed by restricting the algorithm to allocate a job to a fixed time slot on its arrival and not change afterwards (although the job can be removed later); it is therefore identical to the immediate decision problem (although there is no penalty so $\rho = 0$.)

Additionally, banner advertisement auctions were studied in [1] and [7] where request acceptance decisions must be made immediately but it is possible to 'buy back' or 'bump' an accepted request by paying a penalty; the papers explained the motivations for the need for immediate commitment and penalties. In Section 4 we discuss in more detail how this relates to our problem with a separation of arrival and starting times.

Previous Results. The online scheduling of unit length jobs (without any commitment or penalty) has been a very active area of scheduling research in the past ten years or so. It originated from a problem in buffer management [18] but it soon attracted a lot more attention. We refer readers to the survey [13] for a comprehensive literature review of the area. Here we list only the latest results. For a single machine, the current best deterministic algorithm is 1.828-competitive [11], and the lower bound is 1.618 [15,5,20]. The randomized upper and lower bounds are respectively 1.582 and 1.25 [4,5]. For multiple machines the upper and lower bounds are $(1 - (\frac{m}{m+1})^m)^{-1}$ and 1.25 respectively [4].

This unit job scheduling problem with IMM DEC or IMM NOTIF has not been studied before, but there are other existing work on scheduling with commitment (but not penalty), where a committed job must always be completed. Goldwasser and Kerbikov [14] considered IMM NOTIF where jobs have unequal lengths and the objective is to maximize the total length of completed jobs.

They showed that in many cases there are algorithms with competitive ratios that are not worse than their non-commitment counterparts. Ding et al. [8] considered IMM DEC with equal length (but non-unit length) and unweighted jobs (i.e. the objective is to maximize the number of completed jobs), and gave an algorithm with a competitive ratio that tends to $e/(e-1) \approx 1.58$ when m is large. A $e/(e-1)$ lower bound (again for large m) from [10] shows that this algorithm is optimal; in fact the lower bound holds even for unit length jobs. In [2] an *immediate dispatch* model was proposed, in which the machine allocated for an accepted job must be announced upfront but the machine is free to move these jobs around. In [3] an online scheduling problem with commitment (IMM NOTIF) was studied where the penalty is equal to the length of the unfinished part of the job, with application in the charging of electric vehicles.

Allowing a penalty factor with commitment was considered in [12], where it studied longer (non-unit length) jobs and also addressed a related problem of *preemption penalty*. They considered two cases where (i) jobs have different lengths but their values are proportional to their lengths, or (ii) jobs have equal lengths but arbitrary weights. Several algorithms with competitive ratios depending on the penalty factor ρ were given. As already mentioned, commitment with penalty also arises in auctions; [1] and [7] gave algorithms but the items to be auctioned do not spread over a time period (even though the requests do.)

Our Results. In this paper we consider a setting similar to [12] but with unit length jobs. This may look like a simpler problem, but it allows us to remove any consideration of interruption due to job arrival in the middle of job execution and thus study purely the effect of commitment on the scheduling algorithms' performance. As we will shortly see, this allows sublinear (in ρ) competitive ratios which have not yet been achieved in the longer job case.

For IMM DEC we give an upper bound of $O(m(1+\rho)^{1/m})$, and a lower bound of $\Omega(\log \rho)$ (that applies to any number of machines). Note that when $m = \Theta(\log \rho)$ the bounds asymptotically match. When $m > \log \rho$ we modify the algorithm so that the competitive ratio remains $O(\log \rho)$. In the case of one machine we give $\Theta(\rho)$ bounds. For IMM NOTIF, we give an $\Omega(\log \rho / \log \log \rho)$ lower bound on the competitive ratio for any number of machines. For small values of ρ and for a single machine, we give another lower bound. Table 1 summarizes the results. Finally we tighten the bound for IMM NOTIF if arrival times and starting times are separate. Because of space constraints some proofs are omitted.

2 Immediate Decision

2.1 Upper Bound

The algorithm maintains a *provisional schedule* S to record which slot a pending job is scheduled to be executed. Each time a job is accepted, it will be provisionally assigned a slot in S. For each machine i and time step t on or after the current time, $S(i,t)$ denotes the job that is provisionally assigned to slot (i,t)

Table 1. Some typical values of the upper bounds (UB) and lower bounds (LB) on the competitive ratios

m	1								large
ρ	0	0.1	0.2	0.5	1	1.5	2	large	large
IMM DEC UB	2	3.727	4.760	6	8	10	12	$O(\rho)$	$O(\log \rho)$
IMM DEC LB	2	2	2	2	2	2.5	3	$\Omega(\rho)$	$\Omega(\log \rho)$
IMM NOTIF UB	2	3.117	3.727	5.236	7.464	9.583	11.657	$O(\rho)$	$O(\log \rho)$
IMM NOTIF LB	1.618	1.691	1.766	2	2	2	2	$\Omega(\log \rho / \log\log \rho)$	

in S. If no job is assigned to a slot, we imagine that a null job with weight 0 is assigned there.

A simple algorithm would be to accept a job j if it is 'sufficiently heavy':

Algorithm 1. Let $\beta = 2(1+\rho)$ if $\rho > (\sqrt{2}-1)/2$, and $\beta = 1+\rho+\sqrt{\rho^2+\rho}$ otherwise. At every time step t, the algorithm runs an admission procedure to update S, and then simply executes the jobs $S(i,t)$, $1 \le i \le m$. The admission procedure considers each new job j arriving in that time step in turn (the order in which they are considered does not matter). It finds the slot (i,u) before $d(j)$ with the minimum $w(S(i,u))$. Let k be the job in this slot, i.e., $k = S(i,u)$. If $w(j) > \beta w(k)$, accept j in slot (i,u) and k is *evicted*. As this is IMM DEC, k is lost. Otherwise j is rejected.

Theorem 1. *Algorithm 1 is* $\min(4(1+\rho), 4\rho+2+4\sqrt{\rho^2+\rho})$*-competitive for any* m*.*

The competitive ratio of Algorithm 1 grows linearly with ρ. In fact we can give a better algorithm if $m > 1$. The key idea is to make sure that for all jobs committed to the same time step, their weights should differ 'substantially.'

Algorithm 2. Let $\beta = (2\rho+2)^{1/m}$. For each t, let $S'(i,t), 1 \le i \le m$, be the jobs of $S(i,t)$ sorted in decreasing order of job weights, i.e., $S'(i,t)$ denotes the i-th largest-weight job scheduled to run at time step t in the current provisional schedule. (As already noted, if there are fewer than m such jobs, we assume the provisional schedule is filled with weight-0 jobs.) The algorithm maintains the property that $w(S'(i,t)) \ge \beta w(S'(i+1,t))$ for all i and t.

Only the admission procedure is changed from Algorithm 1. For each new job j arriving at this time step t (again the order is not important), the algorithm tries to find a time $u \ge t$ such that $u < d(j)$ and $w(j) \ge \beta w(S'(1,u))$, i.e., j is heavy enough relative to the heaviest job committed to time u. If there are more than one such u, choose anyone. Evict the lightest job at that time step, $S'(m,u)$, replacing it with j. The job $S'(m,u)$ is called the job *evicted* by j. As this is IMM DEC, this evicted job is lost forever and will not be considered again. If no such u exists, reject j. Note that after the eviction, j becomes the new $S'(1,u)$, the old $S'(1,u)$ becomes the new $S'(2,u)$ and so on. This is only for notational convenience; the

jobs stay at the same machines and do not actually move. The property $w(S'(i,u)) \geq \beta w(S'(i+1,u))$ is maintained.

Theorem 2. *Algorithm 2 is $O(m(1+\rho)^{1/m})$-competitive.*

Proof. For brevity, let $x_i(t) = w(\mathcal{OPT}(i,t))$ and $y_i(t) = w(S'(i,t))$ at the end of the execution. Note that at the end of the execution, S is actually the final schedule and so $y_i(t)$ represents $w(\mathcal{ONL}(i,t))$ but in sorted order of weights.

We map each job $\mathcal{OPT}(i,t)$ to a time step of $\mathcal{ONL}$. If $\mathcal{OPT}(i,t)$ is completed in $\mathcal{ONL}$ as well, we map it to the time step at which it appears in $\mathcal{ONL}$. Otherwise, it is either rejected immediately or is evicted later on. If it is rejected on its arrival, then at the time of its arrival, $S'(1,t)$ must be a job with weight larger than $x_i(t)/\beta$. Furthermore, observe that for any slot, the weight of the job in it can only increase during the course of execution of the algorithm. Thus, $y_1(t)$ must also be at least $x_i(t)/\beta$. We map $\mathcal{OPT}(i,t)$ to time step t. Finally, if $\mathcal{OPT}(i,t)$ is evicted in $\mathcal{ONL}$, we associate it with the job that evicts it. If this job is in turn evicted later, the association is transferred to the new evicting job. Thus each evicted job is associated with, possibly via a chain of evictions, a job completed by $\mathcal{ONL}$. Job $\mathcal{OPT}(i,t)$ is then mapped to the time step where it is executed in $\mathcal{ONL}$.

Consider a time step t. We bound the total weight of jobs mapped to t. All the $\mathcal{OPT}(i,t)$ may be rejected by $\mathcal{ONL}$ and thus mapped to t. All the $\mathcal{ONL}(i,t)$ may be completed by $\mathcal{OPT}$ and will be mapped here. Moreover all jobs transitively evicted by $\mathcal{ONL}(i,t)$ may also be mapped here. Note that whenever a job j_1 evicts another job j_2, it must be that $w(j_1) \geq \beta^m w(j_2)$ because this j_2 must be the lightest job $S'(m,u)$ in its time step u and $w(j_1) \geq \beta w(S'(1,u)) \geq \ldots \geq \beta^m w(S'(m,u))$. Thus, the total weight of evicted jobs associated with a job of weight $y_i(t)$ is at most $y_i(t)/\beta^m + y_i(t)/\beta^{2m} + \cdots < y_i(t)/(\beta^m - 1)$. These jobs are all mapped to t. No other jobs are mapped to t. At time t, $\mathcal{ONL}$ gets a profit of $\sum_{i=1}^m y_i(t)$, but as each of the jobs may have evicted a chain of other jobs, there is a penalty of at most $\rho y_i(t)(1/\beta^m + 1/\beta^{2m} + \ldots) < \rho y_i(t)/(\beta^m - 1)$ associated with a job with weight $y_i(t)$. The ratio of mapped $\mathcal{OPT}$ job weights to $\mathcal{ONL}$ job weights, minus penalties, in this time step is therefore at most

$$\frac{\sum_{i=1}^m x_i(t) + \sum_{i=1}^m y_i(t) + \sum_{i=1}^m \frac{y_i(t)}{\beta^m - 1}}{\sum_{i=1}^m y_i(t) - \sum_{i=1}^m \frac{y_i(t)\rho}{\beta^m - 1}} \leq \frac{m\beta y_1(t) + \sum_{i=1}^m y_i(t)\frac{\beta^m}{\beta^m - 1}}{\sum_{i=1}^m y_i(t)(1 - \frac{\rho}{\beta^m - 1})}$$

$$\leq \frac{m(\beta(\beta^m - 1) + \beta^m)y_1(t)}{y_1(t)(\beta^m - 1 - \rho)} = \frac{m((2\rho+2)^{1/m}(2\rho+1) + 2\rho + 2)}{\rho + 1}$$

$$< 2m(2\rho+2)^{1/m} + 2m$$

where we used the fact that $0 \leq y_i(t) \leq y_1(t)$. □

Observe that the competitive ratio $O(m(1+\rho)^{1/m})$ decreases with increasing m, from $O(\rho)$ when $m = 1$, to $O(\log(1+\rho))$ when $m = \ln(2+2\rho)$ (noting that $x^{1/\ln x} = e$ for any x). But the competitive ratio then increases with increasing m. This is due to the fact that the algorithm effectively uses only one slot 'seriously'

in each time step. In fact, for example, if m jobs of the same weight and of tight deadline[2] arrives, the algorithm will only schedule one of them, which clearly makes it not better than m-competitive.

For large m, we can modify the algorithm to retain the $O(\log(1+\rho))$ competitive ratio. For convenience, assume $m' = \ln(2+2\rho)$ is an integer and that m is a multiple of m'. We partition the m machines into m/m' bands, each with m' machines. Each band then effectively runs the algorithm indpendently. Within each band, the machines maintain a provisional schedule with the property that $w(S'(i,t)) \geq \beta w(S'(i+1,t))$ for all i and t, where $\beta = (2\rho+2)^{1/m'}$. When a new job j arrives, it tries to get accepted by testing if $w(j) \geq \beta w(S'(1,u))$ for some u and some band, and evict $S'(m',u)$ if so. Job j is only rejected if it cannot be accepted in any band. (If more than one band can accept the job, choose anyone.)

Theorem 3. *The modified Algorithm 2 is $O(\log(1+\rho))$-competitive.*

Proof. Suppose the bands are numbered 1, 2, ... and $y_i^z(t)$ denotes $w(S'(i,t))$ for band z. The scheme that we use to map jobs in $\mathcal{OPT}$ to time steps in $\mathcal{ONL}$ remain the same as in Theorem 2. In this case, if the job $OPT(i,t)$ is rejected on arrival, then for each band, $S'(1,t)$ must contain a job of weight larger than $x_i(t)/\beta$. We group every m' such jobs and associate them with a $y_1^z(t)$ of a band. Also, if a job j_1 evicts another job j_2 then $w(j_1) \geq \beta^{m'} w(j_2)$. Similar to Theorem 2 the competitive ratio is then

$$\begin{aligned}
&\frac{\sum_{i=1}^{m} x_i(t) + \sum_z \sum_{i=1}^{m'} y_i^z(t) + \sum_z \sum_{i=1}^{m'} y_i^z(t) \frac{\beta^{m'}}{\beta^{m'}-1}}{\sum_z \sum_{i=1}^{m'} y_i^z(t)(1 - \frac{\rho}{\beta^{m'}-1})} \\
\leq\ &\frac{m' \sum_z \beta y_1^z(t) + \sum_z m' y_1^z(t) \frac{\beta^{m'}}{\beta^{m'}-1}}{\sum_z y_1^z(t)(1 - \frac{\rho}{\beta^{m'}-1})} = \frac{m'(\beta(\beta^{m'}-1) + \beta^{m'})}{(\beta^{m'} - 1 - \rho)} \\
=\ &\frac{m'((2\rho+2)^{1/m'}(2\rho+1) + 2\rho + 2)}{\rho+1} < m'(2(2\rho+2)^{1/m'} + 2)
\end{aligned}$$

which is $O(\log(1+\rho))$ as $(2\rho+2)^{1/m'} = (2\rho+2)^{1/\ln(2\rho+2)} = e$. □

2.2 Lower Bounds

Next we prove a lower bound of $\Omega(\log \rho)$ for any number of machines. The bound only works for large ρ. For large number of machines ($m = \Omega(\log \rho)$) it follows from Theorem 3 that the bound is optimal.

Theorem 4. *Any deterministic algorithm has competitive ratio $\Omega(\log \rho)$ for the immediate decision model with m machines.*

[2] We call a job j tight if $d(j) = r(j) + 1$, i.e. it must be scheduled immediately.

Proof. Define a sequence of integers D_i as follows: $D_1 = D$ for a large D, and $D_i = 2(D_{i+1} + 1)$ for $i > 1$. The D is chosen in such a way that $D_1, D_2, \ldots, D_j$ are all integers for some sufficiently large j. Note that if the D_i's are large then all D_i/D_{i+1} are very close to (but above) 2. Moreover, observe that $2 + D_{i+1} = 1 + D_i/2$ and hence $(2 + D_{i+1})2^{i-1} = (1 + D_i/2)2^{i-1} = (2 + D_i)2^{i-2} = \ldots = (2 + D_2)2^0 = 1 + D_1/2$ for all i.

Suppose $\mathcal{ONL}$ is R-competitive. The adversary construction consists of a large number of rounds. In the construction, the minimum job weight is 1 and we will make sure the maximum job weight W is less than ρ, so that there is no point in evicting a job to accept another job because even if we evict a minimum weight job to accept a maximum weight one, this will still result in reduced total profit of the schedule. Then without loss of generality we can assume that no eviction takes place.

Let $I_1 = [0, 1 + D_1)$. Round 1 begins at time $t_1 = 0$, when $m(1 + D_1)$ jobs of deadline $1 + D_1$ and weight 1 arrives. Suppose $\mathcal{ONL}$ accepts x_1 of these jobs. As $\mathcal{OPT}$ can accept all jobs, for $\mathcal{ONL}$ to be R-competitive it must be that $x_1 \geq m(1 + D_1)/R$. At most m of these accepted jobs can be scheduled in $[0, 1)$. Each of the remaining $x_1 - m$ accepted jobs is committed to either $[1, 2 + D_2)$ or $[2 + D_2, 1 + D_1)$, both of length $1 + D_2$. One of them is the denser interval, i.e. the one that contains at least $(x_1 - m)/2$ accepted jobs. Let $I_2 = [t_2, t_2 + D_2 + 1)$ be this denser interval, where $t_2 \in \{1, 2 + D_2\}$. As $\mathcal{ONL}$ obeys immediate decision, the location of I_2 is known.

Then Round 2 begins at t_2, when $m(D_2 + 1)$ jobs of deadline $t_2 + D_2 + 1$ and weight 2 arrives. Clearly they can only be scheduled in I_2. Suppose $\mathcal{ONL}$ accepts x_2 of these jobs. $\mathcal{OPT}$ can always fill slots in I_2 with weight-2 jobs, and fill slots in $I_1 - I_2$ with weight-1 jobs. Thus, $\mathcal{OPT}$ can get a profit of $m(2+D_2)+2m(1+D_2)$. So in order for $\mathcal{ONL}$ to be R-competitive, it must be that $x_1 + 2x_2 \geq m(4 + 3D_2)/R$. There are now at least $(x_1 - m)/2 + x_2 - m$ accepted jobs in the interval $[t_2+1, t_2+D_2+1)$, and we can again divide it into two halves, with one of them being denser, i.e. containing at least half of the accepted jobs. Let $I_3 = [t_3, t_3 + D_3 + 1)$ be this denser interval, with $t_3 \in \{t_2+1, t_2+D_3+2\}$. In this denser interval, there are at least $(x_1 - m)/4 + (x_2 - m)/2$ accepted jobs. We then proceed to the next round where jobs will arrive into this denser interval.

In general, just after round $i-1$ finishes, $\mathcal{ONL}$ has accepted x_j jobs of weight 2^{j-1}, for $j = 1, 2, \ldots, i-1$. At least $(x_1 - m)/2^{i-1} + (x_2 - m)/2^{i-2} + \ldots + (x_{i-1} - m)/2$ jobs are in an interval $I_i = [t_i, t_i + D_i + 1)$. In round i, $m(D_i + 1)$ new jobs, each of weight 2^{i-1}, arrives at time t_i with deadline $t_i + D_i + 1$. $\mathcal{OPT}$ can fill all the slots in I_i with the new weight-2^{i-1} jobs, fill all slots in in $I_{i-1} - I_i$ with weight-2^{i-2} jobs, and so on, and fill all slots in $I_1 - I_2$ with weight-1 jobs. $\mathcal{OPT}$ therefore gets a profit of

$$
\begin{aligned}
& m(2 + D_2) + m(2 + D_3)(2) + \ldots + m(2 + D_i)2^{i-2} + m(1 + D_i)2^{i-1} \\
&= m((2 + D_2) + 2(2 + D_3) + \ldots + 2^{i-2}(2 + D_i) + 2^{i-1}(2 + D_i) - 2^{i-1}) \\
&= m((i + 1)(1 + D/2) - 2^{i-1}) \text{ (since } (2 + D_{j+1})2^{j-1} = 1 + D/2 \text{ for any } j) \\
&> m((i + 1)D/2 - 2^{i-1})
\end{aligned}
$$

So if $\mathcal{ONL}$ accepts x_i of these new jobs, it must be that $x_1+2x_2+\ldots+2^{i-1}x_i \geq (m/R)((i+1)D/2-2^{i-1})$. Now, the interval $[t_i+1, t_i+D_i+1)$ can be partitioned into two halves, and one of them will contain at least $(x_1-m)/2^i+\ldots+(x_i-m)/2$ accepted jobs. Let $I_{i+1} = [t_{i+1}, t_{i+1}+D_{i+1}+1)$, where $t_{i+1} \in \{t_i+1, t_i+2+D_{i+1}\}$, be this denser interval. Proceed to the next round.

If after some round i we have $x_1/2^i+x_2/2^{i-1}+\ldots+x_i/2-m > mD/2^i$, i.e. $x_1+2x_2+\ldots+2^{i-1}x_i > m(D+2^i)$, then as $mD/2^i > mD_{i+1}$ and $1/2^i+1/2^{i-1}+\ldots+1/2 < 1$, this would imply $(x_1-m)/2^i+(x_2-m)/2^{i-1}+\ldots+(x_i-m)/2 > mD_{i+1}$, which means there would not have been enough timeslots to accept all these jobs for it to stay R-competitive, so $\mathcal{ONL}$ loses. We show that this must happen after some finite number of rounds. In fact, we already have the constraints that for all i, $x_1+2x_2+\ldots+2^{i-1}x_i \geq (m/R)((i+1)D/2-2^{i-1})$, so all we need to prove is that there exists a finite i such that $(m/R)((i+1)D/2-2^{i-1}) > m(D+2^i)$. This inequality is true if and only if $i > 2R(1+2^i/D)+2^i/D-1$. Note that we can choose $D \gg 2^i$, so the right hand side in the expression above is at most $2R(1+\epsilon)+\epsilon-1$ for some arbitrarily small $\epsilon > 0$. So when $i = 2R$, i.e. after $2R$ rounds, $\mathcal{ONL}$ must lose. By choosing $R = \log\rho/2$, we have $W = 2^{i-1} = 2^{2R-1} < \rho$, so indeed $W < \rho$ as stated. □

For one machine, the following shows that the $O(\rho)$ upper bound in Theorem 1 is asymptotically optimal.

Theorem 5. *No deterministic algorithm has a competitive ratio better than* $\max(\rho+1, 2)$ *in the immediate decision model with one machine.*

3 Immediate Notification

3.1 Upper Bound

Clearly, the algorithm for immediate decision can also be applied in the immediate notification model. Here we give a similar algorithm that utilizes the ability to reschedule accepted jobs and give a slightly smaller competitive ratio (which is more significant when $m = 1$ and ρ is small). Again we maintain a provisional schedule S (even though in IMM NOTIF the algorithm is not required to announce this information upfront), and define S' similarly.

Algorithm 3. Let $\beta = (2\rho+1)^{1/m}$. We again maintain the property that $S'(i,u) \geq \beta S'(i+1,u)$. For each newly arriving job j arriving at time t, run the following admission procedure:

- Step 1. Find the earliest time u, where $u < d(j)$, such that $w(S'(1,u)) \leq w(j)/\beta$. If no such u exists then reject j. Otherwise, remove the job $k = S'(m,u)$, i.e. the smallest-weight job committed to time u, accept j, and assign the slot vacated by k to j. Note that now j becomes the new $S'(1,u)$. If k is a null job, the procedure finishes. Otherwise we say that k is *displaced* by j, and we proceed to Step 2.

– Step 2. Find the earliest time u such that $u < d(k)$ and $w(k) > w(S'(1,u))$. If no such u exists then k is evicted and the procedure finishes. Otherwise remove the job $k' = S'(1,u)$ from S, replace it with k, and then this k' becomes the new k and Step 2 is repeated.

Intuitively, after k is removed in Step 1, it tries to reinsert itself back into S by following a similar procedure, except that this time β effectively becomes 1 and that we only uses $S'(1,u)$, i.e. the heaviest jobs, not the lightest ones. At the end of this chain of displacements, there may eventually be a job that will not find a place in S. We call it the job *evicted by* j. Note that when we refer to evicted jobs, it is the last job in a chain of displaced jobs, not the job directly displaced by j's insertion.

We show that this algorithm is also $O(m(1+\rho)^{1/m})$-competitive with $\beta = (2\rho+1)^{1/m}$; however for $m = 1$ and ρ small it gives some improvement over Algorithm 1; see Table 1.

Theorem 6. *Algorithm 3 is $O(m(1+\rho)^{1/m})$-competitive with $\beta = (2\rho+1)^{1/m}$. When $m = 1$, the competitive ratio becomes $(2\rho + 2 + 2\sqrt{\rho^2+2\rho})$ with $\beta = 1+\rho+\sqrt{\rho^2+2\rho}$.*

In a way similar to Theorem 3 the algorithm can be modified to retain the $O(\log(1+\rho))$-competitiveness when m is large.

3.2 Lower Bounds

We first consider the case of a single machine. The 1.618 lower bound in [15,5,20] for the no-commitment, no-penalty model carries over here. The following theorem gives a stronger lower bound for $\rho > 0$.

Theorem 7. *In the immediate notification model with one machine, no deterministic algorithm can give a competitive ratio better than* $\min((1+\rho+\sqrt{(1+\rho)^2+4})/2, 2)$.

The above theorem cannot give any lower bound larger than 2 even if ρ is large. The next theorem gives a lower bound that increases with ρ. It only works for large ρ, but it works for any number of machines.

Theorem 8. *Any deterministic algorithm has competitive ratio $\Omega(\log\rho/\log\log\rho)$ for the immediate notification model with m machines.*

4 Separate Arrival and Start Times

Finally, we consider a situation when the arrival time and the earliest allowed start time of a job is not necessarily the same; for example, an order is placed at a certain time but the delivery should only take place during a certain time window that does not begin immediately. There are other situations where this

may be useful; see e.g. [1,7] where it is even assumed that all job arrivals are completed before execution takes place, i.e. they are separate processes.

Formally, a job is represented by a release time $r(j)$, its earliest possible start time $s(j)$, its deadline $d(j)$, and its weight $w(j)$, where $r(j) \leq s(j) < d(j)$. All of the algorithms in this paper can be straightforwardly adapted to this model with the same competitive ratio and essentially identical proof, which we omit. At first sight it might appear this model is easier as it gives 'advance notice' of jobs for the online algorithm. However, it turns out that in this model we can prove tighter (in fact, optimal) lower bounds. The separation of arrival and starting times allows job sequences to behave like the 'online-list' model (where jobs arriving in the same time step are presented to the online algorithm one by one, the next only after the previous has been dealt with), in contrast to the 'online-time' model where all jobs arriving at the same time are presented together. (For a discussion of these models see e.g. [19].) In the case with penalties the online-list model is disadvantageous to the online algorithm.

In IMM NOTIF there is a gap between the upper bound $O(\log \rho)$ and the lower bound $\Omega(\log \rho / \log\log \rho)$. If we separate arrival and start times however we can close this gap by giving a matching lower bound:

Theorem 9. *For the immediate mnotification model with separate arrival and starting times, any deterministic online algorithm is $\Omega(\log \rho)$-competitive.*

Note that in [1] the case where items for auction are considered. In the case where the constraints are represented by a uniform matroid of rank m, it can be interpreted as a special case of our scheduling problem where all jobs have the same $s(j)$ and $d(j) = s(j)+1$, all arrivals are well before $s(j)$ and jobs arrive one by one. The items correspond to the m slots offered by m machines in a single time step. They give matching upper and lower bounds of $1+2\rho+2\sqrt{\rho(1+\rho)} = \Theta(\rho)$ when $m = 1$. Our results here (Theorems 2 and 9) can be seen as results for $m > 1$.

References

1. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling ad campaigns: online algorithms with cancellations. In: Proc. 10th ACM Conference on Electronic Commerce, pp. 61–70 (2009)
2. Bunde, D.P., Goldwasser, M.H.: Dispatching equal-length jobs to parallel machines to maximize throughput. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 346–358. Springer, Heidelberg (2010)
3. Chen, S., Tong, L., He, T.: Optimal deadline scheduling with commitment. In: 49th Annual Allerton Conference on Communication, Control and Computing, pp. 111–118 (2011)
4. Chin, F.Y.L., Chrobak, M., Fung, S.P.Y., Jawor, W., Sgall, J., Tichý, T.: Online competitive algorithms for maximizing weighted throughput of unit jobs. Journal of Discrete Algorithms 4(2), 255–276 (2006)
5. Chin, F.Y.L., Fung, S.P.Y.: Online scheduling with partial job values: Does timesharing or randomization help? Algorithmica 37(3), 149–164 (2003)

6. Cole, R., Dobzinski, S., Fleischer, L.K.: Prompt mechanisms for online auctions. In: Monien, B., Schroeder, U.-P. (eds.) SAGT 2008. LNCS, vol. 4997, pp. 170–181. Springer, Heidelberg (2008)
7. Constantin, F., Feldman, J., Muthukrishnan, S., Pal, M.: An online mechanism for ad slot reservations with cancellations. In: Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1265–1274 (2009)
8. Ding, J., Ebenlendr, T., Sgall, J., Zhang, G.: Online scheduling of equal-length jobs on parallel machines. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 427–438. Springer, Heidelberg (2007)
9. Ding, J., Zhang, G.: Online scheduling with hard deadlines on parallel machines. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 32–42. Springer, Heidelberg (2006)
10. Ebenlendr, T., Sgall, J.: A lower bound for scheduling of unit jobs with immediate decision on parallel machines. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 43–52. Springer, Heidelberg (2009)
11. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 209–218 (2007)
12. Fung, S.P.Y.: Online preemptive scheduling with immediate decision or notification and penalties. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 389–398. Springer, Heidelberg (2010)
13. Goldwasser, M.H.: A survey of buffer management policies for packet switches. SIGACT News 45(1), 100–128 (2010)
14. Goldwasser, M.H., Kerbikov, B.: Admission control with immediate notification. Journal of Scheduling 6, 269–285 (2003)
15. Hajek, B.: On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In: Proc. Conference on Information Sciences and Systems, pp. 434–438 (2001)
16. Kaminsky, P., Hochbaum, D.: Due date quotation models and algorithms. In: Leung, J.Y.-T., Henderson, J.H. (eds.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, ch. 20. Chapman and Hall/CRC (2004)
17. Keskinocak, P., Ravi, R., Tayur, S.: Scheduling and reliable lead-time quotation for orders with availability intervals and lead-time sensitive revenues. Management Science 47(2), 264–279 (2001)
18. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. In: Proc. 33th ACM Symposium on Theory of Computing, pp. 520–529 (2001)
19. Pruhs, K., Sgall, J., Torng, E.: Online Scheduling. In: Leung, J.Y.-T., Henderson, J.H. (eds.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, ch. 15. Chapman and Hall/CRC (2004)
20. Zhu, A.: Analysis of queueing policies in QoS switches. Journal of Algorithms 53, 137–168 (2004)

The Lazy Matroid Problem*

Laurent Gourvès[1,2], Jérôme Monnot[1,2], and Aris T. Pagourtzis[3]

[1] CNRS, UMR 7243 (LAMSADE)
[2] PSL Université Paris Dauphine
Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16, France
[3] National Technical University of Athens (NTUA)
School of Electrical and Computer Engineering
Heroon Politechniou 9, GR-15780 Zographou, Greece
{laurent.gourves,jerome.monnot}@dauphine.fr, pagour@cs.ntua.gr

Abstract. This article introduces the LAZY MATROID PROBLEM, which captures the goal of saving time or money in certain task selection scenarios. We are given a budget B and a matroid $\mathcal{M}$ with weights on its elements. The problem consists in finding an independent set F of minimum weight. In addition, F is feasible if its augmentation with any new element x implies that either $F + x$ exceeds B or $F + x$ is dependent. Our first result is a polynomial time approximation scheme for this **NP**-hard problem which generalizes a recently studied version of the LAZY BUREAUCRAT PROBLEM. We next study the approximability of a more general setting called LAZY STAFF MATROID. In this generalization, every element of $\mathcal{M}$ has a multidimensional weight. We show that approximating this generalization is much harder than for the LAZY MATROID PROBLEM since it includes the INDEPENDENT DOMINATING SET PROBLEM.

Keywords: approximation algorithms, matroids, independent dominating set.

1 Introduction

Imagine that the Minister of Public Works has to select some projects to fund, among a pool of proposed ones. She has a certain budget that she can spend on these projects and she wants to select projects in such a way that as much money as possible are saved (remain unused), yet not enough for any left-out project. This is in fact a 'reincarnation' of the LAZY BUREAUCRAT PROBLEM [1–5] in which a lazy worker wants to select a set of tasks of minimum total duration in such a way that his remaining working time does not suffice to add any task.

* Aris Pagourtzis was supported for this research by the project ALGONOW, cofinanced by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) – Research Funding Program: THALES, Investing in knowledge society through the European Social Fund.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 66–77, 2014.

Assume further that the Minister has to deal with additional constraints, e.g. if the country is divided into regions and there is a maximum number of projects that should be allocated per region. Such constraints can often be described by a matroid on the set of tasks: for example, the above case can be described by a partition matroid. To address such scenarios, we define a generalization of the Lazy Bureaucrat Problem, which we call Lazy Matroid: given a weighted matroid of tasks $(S, \mathcal{F})$ and a budget B, we want to select a set of tasks $S' \subseteq S$ such that adding every left-out task to S' would violate either the budget B or the matroid constraint (or both); the goal is to minimize the weight of S'.

Another situation that can be well described by the Lazy Matroid problem concerns a network design problem: one wants to connect several parts of a network by, say, optical fibers, without exceeding a given cost budget, and respecting two constraints: it is not allowed to create cycles (this is considered unnecessary spending) and it is not acceptable to avoid establishing a connection between unconnected components, if the remaining budget suffices. It is reasonable to assume that the network manager would like to spend as little as possible, without violating the constraints. This is an instance of the Lazy Matroid Problem on graphic matroids.

Our Contribution. In this work we formally define and study the Lazy Matroid problem, which is **NP**-hard, since so is the Lazy Bureaucrat Problem; the latter is a special case of the former as pointed out above. Our first result is a *PTAS* for Lazy Matroid. The proposed algorithm involves careful employment of two well known greedy algorithms for weighted matroids, in conjunction with appropriately designed matroid contraction and restriction operations.

We next consider a more general setting, in which each task has to be carried out by several workers, who collectively wish to minimize their total work load; we call this variant Lazy Staff Matroid. In the Public Works scenario, this would correspond to projects associated with multiple weights, each representing an estimation of the project's negative impact in some domain of increased importance: environment, cultural heritage, unemployment, to name a few; then, one might want to bound the total impact of selected projects in each of the considered domains.

In contrast to the one-worker case, we show that Lazy Staff Matroid is highly inapproximable. We do this by reduction from the independent dominating set problem (ISDS in short). Along the way, we obtain some new (to the best of our knowledge) inapproximability results for ISDS on regular graphs. We finally present a $2m$-approximation algorithm for Lazy Staff Matroid on free matroids. Some proofs are omitted due to space limitation.

Related Work. To the best of our knowledge, this is the first study on the matroidal version of the Lazy Bureaucrat scheduling problem; the latter was defined by Arkin, Bender, Mitchell and Skiena [6, 1] under various optimization objectives. In fact, Lazy Matroid is a generalization of Lazy Bureaucrat

with common arrivals and deadlines, which was shown to admit an *FPTAS* in [2]; note that in the common arrivals case the two most studied objectives, namely `makespan` and `time-spent` coincide. The (weak) **NP**-hardness of this case was shown by Gai and Zhang [7, 3].

Earlier results on Lazy Bureaucrat include approximations for the common deadline case: first a tight 2-approximation algorithm working under both objectives was given by Esfahbod, Ghodsi and Sharifi [8] and later two *PTAS*'s, one for each objective, were presented in [7, 3].

Note that the Lazy Bureaucrat with common arrivals and deadlines is a knapsack-like problem with an inverted objective function since one tries to *minimize* the total value of the solution. Camerini and Vercellis have studied a matroidal version of the knapsack with its classical objective function of *maximizing* the total value of the solution [9].

Coming to the Lazy Staff Matroid problem, we show in Section 5 that it includes well-known problems as special cases, most notably the independent dominating set problem (ISDS). ISDS is not approximable within $n^{1-\varepsilon}$ for any $\varepsilon > 0$ on graphs of n vertices [10] (unless $\mathbf{P} = \mathbf{NP}$). In addition, it is **NP**-hard to approximate ISDS in graphs of degree at most 3 within a factor $\frac{681}{680}$ [11]. Regarding regular graphs no approximability hardness results for ISDS can be found in the literature, up to our best knowledge.

2 Basic Notions on Matroids

This section comprises basic notions on matroids, see [12, 13] for more details. We use the shorthand notation $X + x := X \cup \{x\}$ and $X - y := X \setminus \{y\}$. A matroid $\mathcal{M} = (X, \mathcal{F})$ is a finite set of elements X and a collection $\mathcal{F}$ of subsets of X satisfying the following properties: (i) $\emptyset \in \mathcal{F}$; (ii) if $F_2 \subseteq F_1$ and $F_1 \in \mathcal{F}$ then $F_2 \in \mathcal{F}$; (iii) for every F_1, $F_2 \in \mathcal{F}$ where $|F_1| < |F_2|$, $\exists\ e \in F_2 \backslash F_1$ such that $F_1 + e \in \mathcal{F}$.

The elements of $\mathcal{F}$ and $2^X \setminus \mathcal{F}$ are called *independent* sets and *dependent* sets, respectively. The *bases* of a matroid are its inclusion-wise maximal independent sets. All bases of a matroid $\mathcal{M}$ have the same cardinality $r(\mathcal{M})$, defined as the *rank* of $\mathcal{M}$.

In the presence of a weight function $w : X \to \mathbb{R}$, we use the shorthand notation $w(X') = \sum_{x \in X'} w(x)$ for all $X' \subseteq X$. A matroid $(X, \mathcal{F})$ where each element e has a weight $w(e)$ is a *weighted matroid*; it is denoted by $(X, \mathcal{F}, w)$.

Given $(X, \mathcal{F}, w)$, a classical optimization problem consists in computing a base of minimum weight. This problem is solved by Min-Greedy (see Algorithm 1). Computing a base of *maximum weight* can be done with a similar algorithm called Max-Greedy (the elements of $\mathcal{M}$ are scanned by non-increasing weight); the output of Max-Greedy($\mathcal{M}$) is denoted by $\max -Gr(\mathcal{M})$.

The time complexity of matroid algorithms depends on the difficulty of testing if a set F belongs to $\mathcal{F}$. We deliberately neglect this test when the time complexity of an algorithm is provided. Thus, Algorithm 1 runs in polynomial time.

Given a matroid $\mathcal{M} = (X, \mathcal{F})$ and $Y \subseteq X$, the *restriction* of $\mathcal{M}$ to Y, denoted by $\mathcal{M}|Y$, is the structure $(Y, \mathcal{F}')$ where $\mathcal{F}' = \{Z \subseteq Y : Z \in \mathcal{F}\}$. If $Y \in \mathcal{F}$, the *contraction* of $\mathcal{M}$ by Y, denoted by $\mathcal{M}/Y$, is the structure $(X \setminus Y, \mathcal{F}')$ where $\mathcal{F}' = \{F \subseteq X \setminus Y : F \cup Y \in \mathcal{F}\}$. It is well known that both $\mathcal{M}|Y$ and $\mathcal{M}/Y$ are matroids.

Next Theorem is satisfied by any matroid $\mathcal{M}$.

Theorem 1. *[14] Let B and B' be bases and let $x \in B - B'$. Then there exists $y \in B' - B$ such that both $B - x + y$ and $B' - y + x$ are bases of $\mathcal{M}$.*

Matroids are known to model several structures in combinatorial optimization. For instance, the *free matroid* is defined on a set X, each subset $F \subseteq X$ is independent and the unique base is X. A second example is the *graphic matroid* which is defined on the set of edges of a graph G, the independent sets are the forests of G (subsets of edges without cycles). A base of the graphic matroid is a spanning tree if the graph G is connected. A third example is the *partition matroid*; this matroid is defined on a set X partitioned into k disjoint sets $X_1, \ldots, X_k$ for $k \geq 1$. Given k integers $b_i \geq 0$ $(i = 1, ..., k)$, the independent sets are all the sets $F \subseteq X$ satisfying $|F \cap X_i| \leq b_i$ for all $i = 1, \ldots, k$.

Algorithm 1. MIN-GREEDY

```
Data: a weighted matroid M = (X, F, w)
1 Rename X = {x_1, ..., x_n} such that w(x_i) ≤ w(x_{i+1}), i ≤ n − 1
2 Gr(M) ← ∅
3 for i = 1 to n do
4     if Gr(M) ∪ {x_i} ∈ F then
5         Gr(M) ← Gr(M) ∪ {x_i}
6 return Gr(M)
```

Note that MIN-GREEDY and MAX-GREEDY can also be used to complete an independent set $F \in \mathcal{F}$ into a base. Instead of starting with the empty set as in step 2 of the algorithms, we begin with F. Thus, the completion of F with MIN-GREEDY and MAX-GREEDY provides a base of minimum and maximum weight, respectively, within the set of bases which contain F.

3 Problem Definition and Properties

LAZY MATROID PROBLEM
Input: a weighted matroid $\mathcal{M} = (X, \mathcal{F}, w)$, where w is a positive weight function $w : X \to \mathbb{R}^+$ and a positive bound B.
Output: $F \in \mathcal{F}$ with $w(F) \leq B$ and s.t. $\forall x \in X \setminus F, F + x \in \mathcal{F} \Rightarrow w(F + x) > B$.
Objective: minimize $w(F)$.

Note that a feasible solution to Lazy Matroid Problem must satisfy a constraint of maximality which counterbalances the fact that the weight of a solution must be minimized. This constraint refers to the *busy requirement* of the Lazy Bureaucrat Problem [1].

In what follows, all solution sets will be assumed to be sorted in non decreasing order of weight, unless otherwise stated. For $t \leq n$, $X_t = \{x_1, \ldots, x_t\}$ is the restriction of X to the t smallest elements and $\mathcal{M}_t$ is the restriction of $\mathcal{M}$ to X_t. It is well known that $\mathcal{M}_t$ remains a matroid.

Let $OPT(\mathcal{M}, B) = \{x_{\pi(1)}, \ldots, x_{\pi(p)}\}$ be an optimal solution to the Lazy Matroid Problem on instance $(\mathcal{M}, B)$. We will omit $(\mathcal{M}, B)$ when the context is clear ; $p = |OPT|$. For $t \leq p$, $OPT_t = \{x_{\pi(1)}, \ldots, x_{\pi(t)}\}$ is the restriction of OPT to the t smallest elements.

$Gr(\mathcal{M})$ is the solution returned by the greedy algorithm Min-Greedy with weighted matroid $\mathcal{M}$, see Algorithm 1. It is well known that $Gr(\mathcal{M})$ is a base of $\mathcal{M}$ and has a minimum weight among all bases of $\mathcal{M}$. Actually more generally, if $Gr_t(\mathcal{M})$ denotes the restriction of $Gr(\mathcal{M})$ to the t first elements taken by Min-Greedy, then $Gr_t(\mathcal{M})$ has a minimum weight among all independent sets of $\mathcal{M}$ with size exactly t. Finally $\max -Gr(\mathcal{M})$ is a base of maximum weight of $\mathcal{M}$ and it is returned by Max-Greedy algorithm.

Lazy Greedy is an adaptation of Greedy for the Lazy Matroid Problem and it is described in Algorithm 2.

Algorithm 2. Lazy Greedy

```
Data: a weighted matroid M = (X, F, w) and a bound B
Rename X = {x_1, ..., x_n} such that w(x_i) ≤ w(x_{i+1}), i ≤ n − 1
LazyGr(M, B) ← ∅
for i = 1 to n do
    if LazyGr(M, B) ∪ {x_i} ∈ F and w(LazyGr(M, B)) ≤ B then
        LazyGr(M, B) ← LazyGr(M, B) ∪ {x_i}
return LazyGr(M, B)
```

On free matroids, Lazy Greedy coincides with the shortest job first scheduling policy introduced in [8] for the common deadline case of the Lazy Bureaucrat Problem and provides, in the worst case, a 2-approximation [8]. As proved in [2], a slight modification of this greedy algorithm gives a 4/3-approximation for Lazy Bureaucrat Problem in linear time.

Note that $LazyGr(\mathcal{M}, B)$ is a feasible solution to the Lazy Matroid Problem, but it does not guarantee any constant approximation ratio.

Now, we give some properties on Lazy Greedy which will be useful later. We will suppose that $LazyGr(\mathcal{M}, B) = \{x_{f(1)}, \ldots, x_{f(s)}\}$ is the solution returned by Lazy Greedy; $s = |LazyGr(\mathcal{M}, B)|$. Note that we have $LazyGr(\mathcal{M}, B) = Gr_s(\mathcal{M})$ under the previous notations.

Lemma 1. *Let $k \geq 1$ be an integer. If $w(OPT) \geq B\frac{k}{k+1}$ then* Lazy Greedy *is a $\frac{k+1}{k}$-approximation.*

Lemma 2. *For any instance $(\mathcal{M}, B)$, we have $s \geq p$.*

Actually, the case $s = p$ is polynomially solvable by Lazy Greedy. Henceforth, we focus on the case $s > p$.

4 A *PTAS*

Let us give an overview of the PTAS. Given k in input, the algorithm consists in testing every possible subset of *at most* k elements. Each of these sets that satisfies the feasibility constraint of the lazy matroid problem is stored in a set denoted by Sol. If the optimum uses at most k elements then it must belong to Sol. Otherwise, one tries to guess $A^* = \{x_{g^{A^*}(1)}, \ldots, x_{g^{A^*}(k)}\}$, the k elements of OPT with largest weight. Then $\mathcal{M}$ is contracted by A^* and restricted to the elements of X whose weight does not exceed the weight of the lightest element of A^*. This matroid is denoted by $\mathcal{M}^{A^*}$. max-greedy is run on $\mathcal{M}^{A^*}$ in order to get a set $\{x_{h^{A^*}(1)}, \ldots, x_{h^{A^*}(s^{A^*})}\}$ of s^{A^*} elements. Then $s^{A^*}+1$ sets F_0, F_1, $\ldots F_{s^{A^*}}$ are constructed as follows. For $t \in \{0, 1, \ldots, s^{A^*}\}$, min-greedy is run to complete $\{x_{h^{A^*}(s^{A^*}-t+1)}, \ldots, x_{h^{A^*}(s^{A^*})}\}$ into a base F_t of $\mathcal{M}^{A^*}$. Every set $F_t + A^*$ that satisfies the feasibility constraint of the lazy matroid problem on $(\mathcal{M}, B)$ is added to Sol. Finally, the solution of minimum weight stored in Sol is returned. The algorithm, formally described in Algorithm 3, is shown to be $\frac{k+1}{k}$-approximate.

Note that Sol $\neq \emptyset$ because PTAS-Lazy contains at least the Lazy Greedy solution on initial instance $(\mathcal{M}, B)$. Indeed, when A' denotes the set of the k heaviest elements of $LazyGr(\mathcal{M}, B)$, we have $A' + U_0^{A'} + Gr(\mathcal{M}^{A',0}) = LazyGr(\mathcal{M}, B)$ for iteration $t = 0$.

Let us fix an integer $k \geq 1$ and let us prove that PTAS-Lazy (Algorithm 3) with input k is a $\frac{k+1}{k}$-approximation. Let APX be the solution returned by PTAS-Lazy on input $(\mathcal{M}, B, k)$. Let $OPT = \{x_{\pi(1)}, \ldots, x_{\pi(p)}\}$ be an optimal solution satisfying $\pi(1) < \ldots < \pi(p)$ and $|OPT| = p$. If $|OPT| \leq k$ then $OPT \in$ Sol and the algorithm is 1-approximate. Suppose from now on that $p = |OPT| > k$. Let A^* be the k heaviest elements of OPT, i.e. $A^* = \{x_{\pi(p-k+1)}, \ldots, x_{\pi(p)}\}$. Following the notations of Algorithm 3, we can also define A^* as $\{x_{g^{A^*}(1)}, \ldots, x_{g^{A^*}(k)}\}$. Let $\mathcal{M}^{A^*}$ denote $(X_{g^{A^*}(1)-1}, \mathcal{F}^{A^*}, w)$ where $\mathcal{F}^{A^*}$ is the restriction of $\mathcal{F}$ to the subsets of $X_{g^{A^*}(1)-1} = \{x_1, \ldots, x_{g^{A^*}(1)-1}\}$.

Lemma 3. *If $w(OPT) < B\frac{k}{k+1}$ then $OPT - A^*$ is a base of $\mathcal{M}^{A^*}$.*

For $t = 0, \ldots, s^{A^*}$, let F_t be a base of $\mathcal{M}^{A^*}$ defined as $F_t = U_t^{A^*} \cup Gr(\mathcal{M}^{A^*,t})$. Following Algorithm 3, $U_t^{A^*}$ consists of the t heaviest elements of $\max -Gr(\mathcal{M}^{A^*})$ whereas $Gr(\mathcal{M}^{A^*,t})$ is obtained by running min-greedy on $\mathcal{M}^{A^*,t}$.

Note that $F_{s^{A^*}} = \max -Gr(\mathcal{M}^{A^*})$ and $F_0 = Gr(\mathcal{M}^{A^*})$. Moreover, by a property of Min-Greedy, we know that for any t, F_t is a base of $\mathcal{M}^{A^*}$ with minimum weight among the bases of $\mathcal{M}^{A^*}$ containing $U_t^{A^*}$ (note that $U_0^{A^*} = \emptyset$).

Algorithm 3. PTAS-LAZY

Data: a weighted matroid $\mathcal{M} = (X, \mathcal{F}, w)$, a bound B and an integer $k \geq 1$
Rename $X = \{x_1, \cdots, x_n\}$ such that $w(x_i) \leq w(x_{i+1})$, $i \leq n-1$
for *all* $A \subseteq X$ *of size at most* k **do**
  if A *is a feasible solution to the* LAZY MATROID *problem for instance* $(\mathcal{M}, B)$ **then**
    $\text{Sol} \leftarrow \text{Sol} + A$
for *all* $A = \{x_{g^A(1)}, \ldots, x_{g^A(k)}\} \subseteq X$ *and* $A \notin \text{Sol}$ *with* $|A| = k$, $w(A) < B$ *and* $g^A(1) < \cdots < g^A(k)$ **do**
  Let $\mathcal{M}^A$ be the matroid restricted to $X_{g^A(1)-1}$ and contracted to A
  Compute $\max -Gr(\mathcal{M}^A) = \{x_{h^A(1)}, \ldots, x_{h^A(s^A)}\}$, a maximum weight base of $\mathcal{M}^A$ where $h^A(1) < \ldots < h^A(s^A)$
  for $t = 0$ *to* $s^A = |\max -Gr(\mathcal{M}^A)|$ **do**
    Let U_t^A be the t heaviest elements of $\max -Gr(\mathcal{M}^A)$
    For $t \geq 1$, let $\mathcal{M}^{A,t} = \mathcal{M}_{h^A(s^A-t)}|U_t^A$ be $\mathcal{M}^A$ restricted to $X_{h^A(s^A-t)}$ and contracted to U_t^A; for $t = 0$, let $\mathcal{M}^{A,0} = \mathcal{M}^A$ and $U_0^A = \emptyset$
    if $A + U_t^A + Gr(\mathcal{M}^{A,t})$ *is a feasible solution to the* LAZY MATROID *problem on instance* $(\mathcal{M}, B)$ **then**
      $\text{Sol} \leftarrow \text{Sol} + \left(A + U_t^A + Gr(\mathcal{M}^{A,t})\right)$
return the best solution within Sol

Lemma 4. *If* $w(OPT) < B\frac{k}{k+1}$ *then either* $F_0 + A^*$ *is an optimal solution to the* LAZY MATROID *problem on* $(\mathcal{M}, B)$*, or there exists* $t > 1$ *such that* $w(F_t) \geq w(OPT - A^*)$ *and* $w(F_{t-1}) < w(OPT - A^*)$.

Since both F_t and F_{t-1} are bases of $\mathcal{M}^{A^*}$, we can use Theorem 1 with $x_{h^{A^*}(s^{A^*}-t+1)} \in F_t \setminus F_{t-1}$ to state that there must be $a \in F_{t-1} \setminus F_t$ such that $U := F_{t-1} + x_{h^{A^*}(s^{A^*}-t+1)} - a$ is also a base of $\mathcal{M}^{A^*}$.

Lemma 5. *If* $w(OPT) < B\frac{k}{k+1}$ *then* $F_t + A^*$ *is a* $\frac{k+1}{k}$*-approximate solution to the* LAZY MATROID *problem on* $(\mathcal{M}, B)$.

Proof. Both F_t and U contain $U_t^{A^*}$ and they are both bases of $\mathcal{M}^{A^*}$. Since F_t is a base of $\mathcal{M}^{A^*}$ with minimum weight among the bases of $\mathcal{M}^{A^*}$ containing $U_t^{A^*}$, we get that $w(F_t) \leq w(U)$. Since neither F_t nor U contains A^*, we deduce that

$$w(F_t + A^*) \leq w(U + A^*) \tag{1}$$

We can also see that $w(U) \leq w(F_{t-1}) + w(x_{h^{A^*}(s^{A^*}-t+1)}) < w(OPT - A^*) + \frac{1}{k}w(OPT)$. The first inequality follows from the definition of U. The second inequality is due to $w(F_{t-1}) < w(OPT - A^*)$ (Lemma 4) and the fact that $\forall x \in X_{g^{A^*}(1)-1}$, $kw(x) \leq w(A^*) \leq w(OPT)$. Hence,

$$w(U + A^*) \leq w(OPT) + \frac{1}{k}w(OPT) \tag{2}$$

Using Inequalities (1) and (2), we obtain $w(F_t) \leq \frac{k+1}{k} w(OPT) < B$ because $w(OPT) < B\frac{k}{k+1}$ by hypothesis.

It remains to show that $F_t + A^*$ is a feasible solution to the LAZY MATROID problem. By contradiction, suppose there exists $a \in X \setminus X_{g^{A^*}(1)-1}$ (because F_t is a base of $\mathcal{M}^{A^*}$) such that $F_t + A^* + a \in \mathcal{F}$ and $w(F_t + A^*) + w(a) \leq B$. Note that $|F_t + A^* + a| > |OPT|$ because $|F_t| = |OPT - A^*|$) and $\forall x \in F_t + a$, $w(a) \geq w(x)$. Therefore there exists $b \in (F_t + A^* + a) - OPT$ such that $OPT + b \in \mathcal{F}$ and $w(OPT + b) \leq w(F_t) + w(a) \leq B$, contradicting the feasibility of OPT. □

Theorem 2. PTAS-LAZY *with input k is a polynomial $\frac{k+1}{k}$-approximation for* LAZY MATROID PROBLEM *on $(\mathcal{M}, B)$ where $\mathcal{M} = (X, \mathcal{F}, w)$. The time complexity is $O(|X|^{k+2})$.*

Proof. Let APX be the solution returned by PTAS-LAZY with input k and suppose $|OPT| > k$ (otherwise $w(APX) = w(OPT)$). By construction, $w(APX) \leq B$ because it contains at least one solution (the one returned by LAZY GREEDY). If $w(OPT) \geq B\frac{k}{k+1}$ then $w(APX) \leq \frac{k+1}{k} w(OPT)$. If $w(OPT) < B\frac{k}{k+1}$ then $w(APX) \leq w(F_t + A^*) \leq \frac{k+1}{k} w(OPT)$ by using both $F_t + A^* \in$ Sol and Lemma 5. In any case, we get the expected result. □

5 The Lazy Staff Matroid Problem

LAZY STAFF MATROID PROBLEM
Input: an m-weighted matroid $\mathcal{M} = (X, \mathcal{F}, w)$, where $w : X \to \mathbb{R}_+^m$ is a positive weight function on m dimensions ($w_i(x)$ denotes the i-th component of $w(x)$) and a positive bound B.
Output: $F \in \mathcal{F}$ with $w_i(F) \leq B$ for every $i \in \{1, \ldots, m\}$ and s.t. $\forall x \in X \setminus F, F + x \in \mathcal{F} \Rightarrow w_i(F + x) > B$ for some $i \in \{1, \ldots, m\}$.
Objective: minimize $\sum_{i=1}^{m} w_i(F)$.

For example, dealing with free matroids, a staff is composed of m lazy bureaucrats who have to execute some given jobs. A job is a vector of m non negative integers. Each coordinate k of a job corresponds to the time that worker k would spend for doing his part. In a feasible solution, i.e. a subset of jobs, the constraint of maximality imposes that every additional job would exceed the working time of *at least* one worker.

The LAZY STAFF MATROID PROBLEM is a generalization of the LAZY MATROID PROBLEM; the latter corresponds to the case $m = 1$. The LAZY STAFF MATROID PROBLEM is much harder than the LAZY MATROID PROBLEM.

For instance, it is not difficult to see that the restriction of this problem to binary inputs (i.e., $B, w_i(x) \in \{0, 1\}$) already contains the minimum maximal matching. Given a graph $G = (V, E)$ with n vertices and m edges, instance of MINIMUM MAXIMAL MATCHING, we consider the free matroid on E, $\mathcal{M} = (E, 2^E)$ and we set $B = 1$. There are n mappings w_v for $v \in V$ described by: $w_v(e) = 1$ if v is incident to e in G and $w_v(e) = 0$ otherwise. Since, any Lazy Staff solution corresponds to a maximal matching in G and vice versa, the result follows.

We will prove that the LAZY STAFF MATROID PROBLEM contains the independent dominating set problem (ISDS) in regular graphs. This latter problem is also known as *minimum maximal independent set*. Given a graph $G = (V, E)$, we want to find $S \subset V$ which is *independent* (no two vertices in S are joined by an edge) and *dominating* (every vertex of $V \setminus S$ is adjacent to some vertex of S) of minimum size. ISDS is one of the hardest, well-known, **NP**-hard graph problems. In [10], it is shown that this problem is not approximable within $n^{1-\varepsilon}$ for any $\varepsilon > 0$ on graphs of n vertices (assuming $\mathbf{P} \neq \mathbf{NP}$). In addition, it is **NP**-hard to approximate ISDS in graphs of degree at most 3 within a factor $\frac{681}{680}$ while a 2-approximation algorithm exists [11]. Up to our best efforts, we were not able to find in the literature any complexity results dealing with regular graphs, but some results can be deduced from existing ones.

Lemma 6. ISDS *is* **APX***-complete in cubic graphs and it is not constant approximable in regular graphs, unless* $\mathbf{P} = \mathbf{NP}$.

Proof. For the first part of the lemma, we prove that the reduction given in [15, Theorem 13] for the **NP**-completeness of ISDS is actually an L-reduction [16].

First, we start from the dominating set problem (DS) which is known to be **APX**-complete in cubic graphs [17]. Given a cubic graph $G = (V, E)$ with n vertices and m edges, instance of DS, we obtain a cubic graph $H = (V', E')$ by replacing each edge $e = [u, v] \in E$ by a gadget $H(e)$. This transformation is illustrated in Figure 1 and we are going to show that it is an L-reduction.

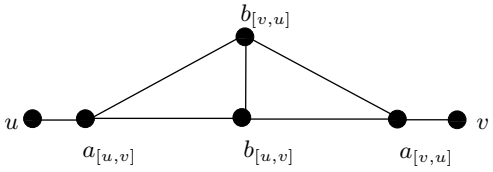


Fig. 1. Local replacement of $[u, v]$ by $H([u, v])$ where four new vertices are added

Let $\mathcal{D}^*$ be an optimal dominating set of G and let $ds(G)$ denote its size. One can build an independent dominating set $\mathcal{ID}$ of H based on $\mathcal{D}^*$ as follows. Begin with $\mathcal{ID} = \emptyset$. If $v \in \mathcal{D}^*$ then add v to $\mathcal{ID}$. For every edge $[u, v]$ do: if $\mathcal{D}^* \cap \{u, v\} = \emptyset$ then add $b_{[u,v]}$ to $\mathcal{ID}$; if $\mathcal{D}^* \cap \{u, v\} = \{u\}$ then add $a_{[v,u]}$ to $\mathcal{ID}$; if $\mathcal{D}^* \cap \{u, v\} = \{u, v\}$ then add $b_{[u,v]}$ to $\mathcal{ID}$. Thus, an independent dominating set $\mathcal{ID}$ for H is obtained and its size is $ds(G) + m$. Let $isds(H)$ denote the size of an optimal independent dominating set of H. We have:

$$isds(H) \leq ds(G) + m \tag{3}$$

Since G is cubic, we know that $m = 3n/2$ and $ds(G) \geq \frac{n}{4}$ (a node can cover four nodes: itself and its three neighbors). From inequality (3), we get that

$$isds(H) \leq ds(G) + \frac{3n}{2} \leq 7ds(G)$$

From any independent dominating set $\mathcal{ID}$ of H with value $apx(H)$, we can polynomially obtain a dominating set $\mathcal{D}$ of G with value $apx(G)$ satisfying

$$apx(G) \leq apx(H) - m \tag{4}$$

Inequality (4) is obtained as follows. For any edge $[u, v] \in E$, we first observe that we can always suppose that $|\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\}| = 1$. Indeed $|\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\}| \neq 0$ otherwise $b_{[u,v]}$ and $b_{[v,u]}$ are not dominated; $|\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\}| < 3$ otherwise $\mathcal{ID}$ is not independent. It can be $|\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\}| = 2$ only when $\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\} = \{a_{[u,v]}, a_{[v,u]}\}$ and one can modify $\mathcal{ID}$ in order to reduce its size. Let $E'' := \{[u, v] \in E : \mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\} = \{a_{[u,v]}, a_{[v,u]}\}\}$. Proceed as follows until $E'' = \emptyset$:

- take a vertex $s \in V$ endpoint of at least one edge of E'';
- denote by E_s the edges $[s, t] \in E \backslash E''$ such that $\mathcal{ID} \cap \{a_{[s,t]}, a_{[t,s]}, b_{[s,t]}, b_{[t,s]}\} = \{a_{[s,t]}\}$;
- $\mathcal{ID} \leftarrow \mathcal{ID} + s$;
- for every $[s, t] \in E''$, do $\mathcal{ID} \leftarrow \mathcal{ID} - a_{[s,t]}$;
- for every $[s, t] \in E_s$, do $\mathcal{ID} \leftarrow \mathcal{ID} - a_{[s,t]} + b_{[s,t]}$;
- update E'' by deleting all edges incident to s.

The modification is such that $\mathcal{ID}$ remains an independent dominating set with no greater size and $|\mathcal{ID} \cap \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]}\}| = 1$ for any edge $[u, v] \in E$. Moreover $\mathcal{D} = \mathcal{ID} \setminus \{a_{[u,v]}, a_{[v,u]}, b_{[u,v]}, b_{[v,u]} : [u, v] \in E\}$ is a dominating set of G.

Hence, using inequalities (3) and (4) we deduce that $apx(G) - ds(G) \leq apx(H) - isds(H)$. In conclusion, the reduction is an L-reduction from DS in cubic graphs to ISDS in cubic graphs. Since the former is **APX**-complete [17], we obtain that the latter is (also) **APX**-complete. This implies that no PTAS for ISDS in cubic graphs exists unless $\mathbf{P} = \mathbf{NP}$.

For the second part of the lemma, we use the self improvement of ISDS based on the graph composition as it is done for the independent set problem, see for instance Theorem 6.12, page 146, of [18].

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *composition* $G_1[G_2]$ is the graph that has vertex set $V_1 \times V_2$ and edge set $\{[(u_1, u_2), (v_1, v_2)] :$ either $[u_1, v_1] \in E_1$ or $u_1 = v_1$ and $[u_2, v_2] \in E_2\}$.

Given a *regular* graph $G = (V, E)$ on n vertices and degree $\Delta(G)$, its composition with itself, that is $G[G]$, is denoted by $G'' = (V'', E'')$

It is not difficult to see that G'' is also a regular graph of degree $\Delta(G'') = (n + 1)\Delta(G)$ and we have:

$$isds(G'') = isds^2(G) \tag{5}$$

Moreover, from any independent dominating set $IDS(G'')$ of G'' with value $apx(G'')$, we can polynomially find an independent dominating set $IDS(G)$ of G of value $apx(G)$ such that:

$$apx^2(G) \leq apx(G'') \tag{6}$$

Let $D_1 = \{u \in V : (u,v) \in IDS(G'')$ for some $v \in V\}$ and for $u \in D_1$, $D_2^u = \{v \in V : (u,v) \in IDS(G'')\}$. It is easy to check that D_1 and D_2^u for $u \in D_1$ are independent dominating sets of G. Thus, if $IDS(G)$ is the set of smallest cardinality in $\{D_1\} \cup \{D_2^u : u \in D_1\}$ then $apx(G'') = \sum_{u \in D_1} |D_2^u| \geq |IDS(G)|^2 = apx^2(G)$.

In conclusion, any constant approximation of ISDS allows us to obtain a polynomial-time approximation scheme which is a contradiction with the first claim. □

Theorem 3. *Unless* $\mathbf{P} = \mathbf{NP}$*, the* Lazy Staff matroid Problem *is not constant approximable even for the free matroid and binary weights (i.e.,* $B, w_i(x) \in \{0,1\}$*).*

Proof. We propose an approximation-preserving reduction from the independent dominating set problem in regular graphs. Let $G = (V,E)$ be a regular graph of degree $\Delta(G)$, with n vertices and m edges, instance of ISDS. Let $\mathcal{M} = (V, 2^V)$ be a free matroid on V and let $B = 1$. There are m mappings w_e for $e \in E$ described by: $w_e(v) = 1$ if v is incident to e in G and $w_e(v) = 0$ otherwise.

Clearly, $S \subseteq V$ is a lazy solution iff S is an independent dominating set of G. Moreover, $\sum_{e \in E} w_e(S) = \Delta(G)|S|$. Thus, using Lemma 6, the result follows. □

Using the proof of Theorem 3, we can deduce that any approximation ratio of Lazy Staff matroid Problem might depend on parameter m.

Let us now study the generalization of Lazy Greedy in the context of the Lazy Staff matroid Problem with the free matroid. Let Lazy Staff Greedy be the algorithm which first renames the elements by non-decreasing sum of their coordinates (ties are broken arbitrarily). At the beginning $I = \emptyset$ and there is a pointer t on the first element. While $t \leq n$, if $I \cup \{t\}$ is a feasible Lazy Staff solution, then $I \leftarrow I \cup \{t\}$, $t \leftarrow t + 1$.

Lemma 7. Lazy Staff Greedy *is* $2m$*-approximate on free matroids.*

Proof. Let OPT be the value of an optimal solution while APX denotes the value of the solution returned by Lazy Staff Greedy. Suppose $OPT \geq \frac{B}{2}$. Since Lazy Staff Greedy returns a feasible solution, we get that $APX \leq mB \leq 2mOPT$.

Now suppose $OPT < \frac{B}{2}$. It follows that for every element of the optimum, the sum of its coordinates is at most $OPT < B/2$. Moreover, every element whose sum of its coordinates is at most $B/2$ must be in the optimum (by the maximality constraint). Hence Lazy Staff Greedy builds the optimum by taking all elements whose sum of its coordinates is at most $B/2$.

Consider the instance with 3 elements whose weights are $(\frac{B}{2} + \varepsilon, 0, \cdots, 0)$, $(\frac{B}{2} + \varepsilon, \varepsilon, 0, \cdots, 0)$ and $(\frac{B}{2} - \varepsilon, B, \cdots, B)$. Lazy Staff Greedy returns a solution which contains elements 1 and 3 while the optimum consists of the second element. The ratio is $mB/(B/2 + 2\varepsilon)$ which tends to $2m$ as ε tends to 0. □

References

1. Arkin, E.M., Bender, M.A., Mitchell, J.S.B., Skiena, S.: The lazy bureaucrat scheduling problem. Inf. Comput. 184, 129–146 (2003)
2. Gourvès, L., Monnot, J., Pagourtzis, A.T.: The lazy bureaucrat problem with common arrivals and deadlines: Approximation and mechanism design. In: Gąsieniec, L., Wolter, F. (eds.) FCT 2013. LNCS, vol. 8070, pp. 171–182. Springer, Heidelberg (2013)
3. Gai, L., Zhang, G.: Common deadline lazy bureaucrat scheduling revisited. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 515–523. Springer, Heidelberg (2006)
4. Gai, L., Zhang, G.: On lazy bureaucrat scheduling with common deadlines. J. Comb. Optim. 15, 191–199 (2008)
5. Hepner, C., Stein, C.: Minimizing makespan for the lazy bureaucrat problem. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, pp. 40–50. Springer, Heidelberg (2002)
6. Arkin, E.M., Bender, M.A., Mitchell, J.S.B., Skiena, S.S.: The Lazy Bureaucrat Scheduling Problem. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 122–133. Springer, Heidelberg (1999)
7. Gai, L., Zhang, G.: Hardness of lazy packing and covering. Oper. Res. Lett. 37, 89–92 (2009)
8. Esfahbod, B., Ghodsi, M., Sharifi, A.: Common-deadline lazy bureaucrat scheduling problems. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 59–66. Springer, Heidelberg (2003)
9. Camerini, P.M., Vercellis, C.: The matroidal knapsack: A class of (often) well-solvable problems. Operations Research Letters 3, 157–162 (1984)
10. Halldórsson, M.M.: Approximating the minimum maximal independence number. Information Processing Letters 46, 169–172 (1993)
11. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems in bounded degree graphs. Inf. Comput. 206, 1264–1275 (2008)
12. Oxley, J.G.: Matroid Theory. Oxford University Press (1992)
13. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Heidelberg (2003)
14. Brualdi, R.: Comments on bases in different structures. Bull. Austral. Math. Soc. 1, 161–167 (1969)
15. Manlove, D.F.: On the algorithmic complexity of twelve covering and independence parameters of graphs. Discrete Applied Mathematics 91, 155–175 (1999)
16. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes (extended abstract). In: Simon, J. (ed.) STOC, pp. 229–234. ACM (1988)
17. Alimonti, P., Kann, V.: Some apx-completeness results for cubic graphs. Theor. Comput. Sci. 237, 123–134 (2000)
18. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

Treewidth Computation and Kernelization in the Parallel External Memory Model

Riko Jacob[1], Tobias Lieber[1], and Matthias Mnich[2]

[1] Institute for Theoretical Computer Science, ETH Zürich, Switzerland
{rjacob,lieberto}@inf.ethz.ch
[2] Cluster of Excellence MMCI, Saarbrücken, Germany
m.mnich@mmci.uni-saarland.de

Abstract. We present a randomized algorithm which computes, for any fixed k, a tree decomposition of width at most k of any input graph. We analyze it in the parallel external memory (PEM) model that measures efficiency by counting the number of cache misses on a multi-CPU private cache shared memory machine. Our algorithm has sorting complexity, which we prove to be optimal for a large parameter range.

We use this algorithm as part of a PEM-efficient kernelization algorithm. Kernelization is a technique for preprocessing instances of size n of NP-hard problems with a structural parameter κ by compressing them efficiently to a kernel, an equivalent instance of size at most $g(\kappa)$. An optimal solution to the original instance can then be recovered efficiently from an optimal solution to the kernel. Our main results here is an adaption of the linear-time randomized protrusion replacement algorithm by Fomin et al. (FOCS 2012). In particular, we obtain efficient randomized parallel algorithms to compute linear kernels in the PEM model for all separable contraction-bidimensional problems with finite integer index (FII) on apex minor-free graphs, and for all treewidth-bounding graph problems with FII on topological minor-free graphs.

1 Introduction

Many practically relevant computational problems are NP-hard. Many decision problems on graphs become efficiently solvable in the RAM model if the input graph has bounded treewidth. By now, there is the linear time algorithm to compute a tree decomposition [1], working in the classical RAM model. This algorithm has been adapted to the PRAM model [2] and to serial models that take the caches of the memory hierarchy into account [3]. Here, we present an adaption of the algorithm that is simultaneously parallel and cache efficient, as modeled by the PEM model, which is defined in Section 1.1.

Another common technique for solving NP-hard problems is pruning easy parts of an instance in a preprocessing step. In the field of parameterized complexity this is formalized as *kernelization*. It is known that every decidable fixed-parameter tractable problem Π for a parameter κ admits a *kernelization*, which is an algorithm that in polynomial time reduces any instance of Π of size n to

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 78–89, 2014.

an equivalent instance (the *kernel*) of size $g(\kappa)$ for some computable function g. Here, equivalent means that the original instance is a "yes"-instance if and only if the kernel is. For example for the VERTEX COVER problem, Nemhauser and Trotter have shown that any graph G on n vertices can be kernelized efficiently into a graph G' on at most 2κ vertices, where κ denotes the size of a minimum vertex cover of G [4]. Recent meta-results show that large classes of combinatorial optimization problems admit kernels of linear size on planar graphs [5], and, more generally, classes of graphs excluding a fixed minor [6,7]. On the other hand, not every fixed-parameter tractable problem admits a polynomial sized kernel, unless the polynomial hierarchy collapses to the third level [5,8].

The classical view on kernelization in the RAM model is to solve an instance I of a hard problem Π in two phases: the first phase, kernelization, transforms in polynomial time the instance I of size n into a kernel I' whose size $g(\kappa)$ depends solely on the structural parameter κ. The second phase solves the problem Π on the kernel I' in time $f(g(\kappa))$ for some function f, which is often at least exponential due to a brute force algorithm. Thus, this leads in total to a running time of $\mathcal{O}(p(n) + f(g(\kappa)))$ to decide I in the RAM model.

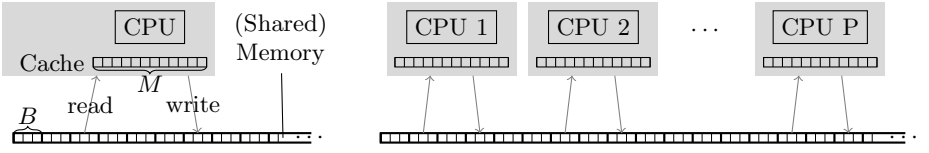
Given the abundance of practically relevant NP-hard problems where input instances are large, it would be nice to have this approach efficiently implemented. Recently, efficient kernelization algorithms have been proposed, for example by Hagerup [9] and van Bevern et al. [10], where linear time RAM algorithms are presented. Further, meta-results by Fomin et al. provide randomized linear-time kernelizations for many combinatorial optimization problems on graph classes excluding a fixed minor [6]. There are some reservations against the practicability of the algorithms implied by these meta-results, namely very large constant factors hidden in the $\mathcal{O}$-notation, and some difficulties in actually creating certain constant sized structures.

To obtain efficient algorithms on modern computers, parallelism and hierarchical memories have to be exploited. Such a parallel and memory efficient implementation is frequently possible for the second phase of kernelization. On the other hand, due to the exponential work that is done, this only moderately extends the size of I', which can be handled. We show that the randomized kernelization algorithms of Fomin et al. can be efficiently implemented in the PEM model, i.e., in parallel and taking the memory hierarchy into account. While this algorithm does not improve the large constant factors in the runtime, it shows that there are no fundamental obstacles when implementing this kind of algorithm in the PEM model. We expected that concrete algorithms for the problems covered by the meta-results can also be implemented efficiently in the PEM model.

1.1 Model of Computation: The Parallel External Memory Model

For using modern computers optimally, two architectural features require special consideration: parallelism and the memory hierarchy. To capture these two fundamental features, the *parallel external memory* (PEM) model was introduced [11]. It is based on the *external memory* (EM) model which captures

cache efficiency of algorithms [12]. The EM model extends the RAM model by a *cache*, located between CPU and memory (see Figure 1a). The cache can store up to M elements, on which the CPU can perform any computations for free. The conceptually infinite (external) memory is partitioned in blocks, each holding up to B elements. A block can be transferred as a unit by an *input/output operation* (I/O) between the cache and the memory. The memory initially holds the input of size n in the first $\lceil n/B \rceil$ blocks.



(a) The EM Model (b) The PEM Model

Fig. 1. The Sequential and the Parallel EM Model for $M = 12$ and $B = 3$

The PEM model (see Figure 1b) is a parallel extension of the EM model. It has P CPUs which all have their own (private) cache of size M. In one *parallel I/O* each processor can perform one I/O operation for moving data between its cache and the shared memory. Similar to the PRAM model, this requires a policy for simultaneous access to the same block. Here, we work with the CREW policy that allows concurrent read but disallows concurrent write. The *complexity* of an algorithm in the PEM model is the number of parallel I/Os it performs. An algorithm is called *PEM-efficient* if its complexity is matched by a lower bound for the problem in the PEM model.

The easiest non-trivial task in the (P)EM model is that of permuting: given a permutation π, create from the input $(x_1, \ldots, x_n)$ the output $(x_{\pi(1)}, \ldots, x_{\pi(n)})$. While in the (P)RAM model the complexity of permuting is the same as for scanning the input, $\Theta(n/P)$, in the (P)EM model it is not. For many settings of the parameters, the optimal solution in the PEM model for permuting is to sort in $\mathsf{sort}_P(n) = \Theta(\frac{n}{PB} \overline{\log}_{\frac{M}{B}} \frac{n}{B})$ parallel I/Os, where $\overline{\log} x$ equals $\max\{1, \log x\}$.

In the PRAM model one important goal is to obtain work optimal algorithms for a maximal number of processors. For the PEM model, one strives similarly to use as many processors as possible, while still using the optimal number of parallel I/Os. We define a *PEM-efficient kernelization* for a parameterized problem Π as an algorithm $\mathcal{A}$ that computes for any instance x of size n with parameter κ a kernel (x', κ'), and $\mathcal{A}$ is PEM-efficient. In our case this means $\mathcal{A}$ executes with $\mathcal{O}(\mathsf{sort}_P(n))$ parallel I/Os.

1.2 Related Work

A *tree decomposition* of a graph G is a tree T and for each node i of T there is a *bag* B_i of vertices of G such that (i) for each vertex v of G there is at least one

bag containing v; (ii) for each edge e of G there is at least one bag containing both its endpoints; and (iii) for each v of G, the set $\{i \mid v \in \mathcal{B}_i\}$ induces a subtree of T. The *width* of a tree decomposition is defined as the maximum bag size minus one. The *treewidth* of G, denoted by $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G.

There are several fixed-parameter algorithms to compute the treewidth of a graph in different computational models. The most important to this paper are the following: Bodlaender and Kloks introduced in 1993 the first linear time algorithm to efficiently compute a tree decomposition of fixed width k of a graph G in the RAM model [13]. Later Bodlaender and Hagerup introduced a work optimal algorithm for the PRAM model for up to $p \leq \frac{n}{\log^2 n}$ processors [2]. An I/O efficient algorithm for computing the treewidth in the EM model has been presented by Maheshwari and Zeh [3].

Recently numerous problems have been shown to admit linear kernels on generalizations of planar graphs, including graphs of bounded genus, and graphs of bounded maximum degree [6,7]. The results are briefly summarized:

Throughout the paper, let H denote a fixed graph and let $\mathcal{G}_H$ be the graph class that excludes H as a minor. For any class $\mathcal{G}_H$, where H is an apex graph (so $H \setminus \{v\}$ is planar for some vertex v), we deal with essentially all problems Π that are "contraction-bidimensional". Roughly speaking, a parameterized problem Π is contraction-bidimensional if contracting an edge cannot increase the objective value κ, and on square grids the objective value grows proportionally with the size of the vertex set of the grid.

Lemma 1 ([14]). *On any class of graphs excluding a fixed apex graph as a minor, all separable contraction-bidimensional problems with finite integer index, parameterized by solution size κ admit kernels of size $\mathcal{O}(\kappa)$.*

Graph classes excluding a fixed graph H as a minor can be generalized to classes excluding H as a topological minor. A graph problem Π, parameterized by κ is called *treewidth-bounding* if all "yes"-instances (G, κ) of Π admit an $\mathcal{O}(\kappa)$-sized set $S \subseteq V(G)$ such that the graph $G \setminus S$ has constant treewidth.

Lemma 2 ([7]). *On any class of graphs excluding a fixed graph as a topological minor, all treewidth-bounding graph problems with finite integer index, parameterized by solution size κ have a kernels of size $\mathcal{O}(\kappa)$.*

Recently, Hagerup [9] and van Bevern et al. [10] argued that not only the size $g(\kappa)$ of the produced kernel, but also lowering the running time of the kernelization algorithm is an important research direction. Another strong case for designing kernels that are as efficient as possible is made by Komusiewicz and Niedermeier [15].

Neither of the approaches considers kernelization in a context of parallel I/O algorithms, but only in the classical RAM model. Thus the approach of designing PEM-efficient algorithms for kernelization is completely new.

Most proofs are omitted due to space limitations and will be published in a full version of this paper.

1.3 Our Contributions

Our first contribution is for every fixed $k \in \mathbb{N}$ a randomized PEM-efficient algorithm which for a given graph G computes a tree decomposition of width at most k, or decides that the treewidth of G is larger than k. Our algorithm is based on the treewidth algorithm for the PRAM model [2] and the treewidth algorithm for the EM model [3]. While the first algorithm yields a framework for load balancing in treewidth computations, the latter yields an EM efficient implementation of the dynamic programming approach of Bodlaender and Kloks [1]. The combination yields a rather technical implementation of the load balancing part of Bodlaender and Hagerup, and a parallelized version of the algorithm of Maheshwari and Zeh. The fundamental building block for our algorithm, which is given in Section 2, is list ranking. Furthermore, the fundamental data structure flippable DAG for the EM model, introduced by Maheshwari and Zeh [3], is replaced by a simpler construction as it appears to be hardly parallelizable.

Theorem 3. *For every $k, t \in \mathbb{N}$, the expected number of parallel I/Os needed to compute a tree decomposition of width at most k, if such exists, for a graph G of size n in the CREW PEM model with $P \leq n/(B^2 \log B \log n \log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$ is $\mathcal{O}(\mathsf{sort}_P(n))$.*

Throughout this paper, t is a fixed constant which can be chosen arbitrary, influencing only the constant in the $\mathsf{sort}_P(n)$ term. Furthermore, it is defined $\log^{(1)} x = \log x$ and $\log^{(k)} x = \log \log^{(k-1)} x$.

Observe that the bound on the I/O complexity is tight for a wide range of the number of processors, which is shown in Section 5.

Our second contribution is a PEM-efficient implementation of the randomized fast protrusion replacer of Fomin et al. [6]. Their fast protrusion replacer works by replacing large protrusions, which are subgraphs of small treewidth and small attachment to the rest of the input graph, by smaller, constant-sized protrusions. It can be used to provide efficient randomized kernelization algorithms for the (linear) kernels mentioned in Lemma 1 and Lemma 2. Using our PEM-efficient algorithm for computing the treewidth of a graph we argue in Section 3 that this protrusion replacer can be implemented efficiently in the PEM model. Using the randomized fast protrusion replacer, in Section 4 the following theorem is shown.

Theorem 4. *The expected number of parallel I/Os to compute a linear kernel for each of the problems of Lemma 1 and Lemma 2 is $\mathcal{O}(\mathsf{sort}_P(n))$ in the CREW PEM model with $P \leq n/(B^2 \log B \log^2 n \log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$.*

By Lemma 1, PEM-efficient kernelizations to linear kernels exist, among others, for Dominating Set, Connected Dominating Set, and Induced Matching on all classes of apex-minor free graphs. Furthermore, by Lemma 2 there exist PEM-efficient parallel randomized kernelizations on H-topological-minor free graphs for, among others, Chordal Vertex Deletion, Interval Vertex Deletion, Treewidth-w Vertex Deletion, and Edge Dominating Set. Moreover, the Width-b Vertex Deletion problem admits a linear kernel in the PEM model on H-topological minor-free graphs, where the width measure is either treewidth, branchwidth, rankwidth, or cliquewidth.

2 Computing Tree Decompositions in the PEM Model

In this section we present for every k a randomized PEM-efficient algorithm to decide the Treewidth-k problem. More precisely, for every k the algorithm computes a tree decomposition of a graph G of width k if there exists one.

We briefly state the I/O-complexities of the most important algorithmic problems in the PEM model. Sorting n records can be done by a multi-way merge sort with $\mathsf{sort}_P(n) = \Theta(\frac{n}{PB} \log_d \frac{n}{B})$ parallel I/Os for $d = \max\{2, \min\{\frac{n}{PB}, \frac{M}{B}\}\}$, and $P \leq \frac{n}{B}$ [16]. The problem of ranking a list of size n is fundamental in the (P)EM model and implies an efficient algorithm for the prefix sums problem [17,18,19]. Both can be solved with $\mathsf{listRank}_P(n) = \mathsf{sort}_P(n)$ parallel I/Os if $P \leq \frac{n}{B^2 \log B \log^{(t)} n}$, and $M = B^{\mathcal{O}(1)}$ [19].

Note that we assume in this paper that all graphs are given by *edge lists* and each undirected edge $\{u, v\}$ is represented by two directed edges (u, v) and (v, u).

2.1 A Framework for Parallel Treewidth Computation

Most tree decomposition algorithms follow a recursive approach [1,2,3]: by recursive application of a reduction rule, the input graph G is reduced to constant size for which a tree decomposition of width k can be given easily. By revoking a round of the application of the reduction rules, a tree decomposition is obtained which is only slightly too large. By using dynamic programming the width of this tree decomposition can be reduced to k, again. Our algorithm follows the approach presented by Bodlaender and Hagerup [2] for dealing with the load balancing in the PRAM model. Its pseudo code, which also indicates a load balancing step, is presented in Algorithm 1.

In the following, for the three methods, `reduce`, `treeDecompostionOfAtMost`, and `balance`, the most important properties and implementation details are presented. They are presented in detail in the full version of this paper. Note that the correctness for all these algorithms follows directly from the original works [2,3].

Algorithm 1. Computing a Tree Decomposition in a parallel Model

```
1  G_0 ← G
2  r ← d · log n
3  for 1 ≤ i ≤ r do
4      G_i ← reduce(G_{i−1})
5  T_r ← treeDecompostionOfAtMost(G_r, k)
6  for r ≥ j ≥ 1 do
7      T_{j−1} ← T_j ∪ (G_{j−1} \ G_j)        // revoking changes of reduce round j
8      T_{j−1} ← balance(T_{j−1})
9      T_{j−1} ← treeDecompostionOfAtMost(T_{j−1}, k)
10 return T_0
```

The `reduce` *Method:* It was first presented by Bodlaender and Hagerup [2]. One call (in round i) to the reduce method decreases the size of the input graph by a constant fraction $\frac{1}{d} \geq \frac{1}{8^k}$ by identifying disjoint pairs of vertices, so called acquainted twins. Therefore, after $d \cdot \log n$ rounds, the resulting graph has constant size and thus a tree decomposition of constant width can be computed by brute force. Furthermore, revoking the vertex identifications of round i increases the width of the tree decomposition T_i for G_{i+1} by at most $k+1$.

The implementation is mostly a straight forward, but technical and careful, implementation of the PRAM algorithm in the PEM model. It uses rounds of local operations, and information exchange, implemented by scanning and sorting. For finding vertices which can be identified in parallel a conflict graph of bounded degree, which represents if acquainted twins can be reduced in parallel, is computed. A fractional independent set (FIS) in the conflict graph then yields the vertices which are reduced in parallel. For computing a FIS of the conflict graph we observe that the PRAM algorithm of Dadoun and Kirkpatrick [20] can be implemented efficiently in the PEM model.

Balancing a Tree Decomposition: The method `balance` obtains from an arbitrary tree decomposition $\mathcal{T} = (T, B)$ of a graph G of width k' a tree decomposition $\mathcal{T}_b = (T', B')$ of width at most $\ell = 7k'$. The most important property of $\mathcal{T}_b$ is that T' is *balanced*, meaning it is suited for processing computations on its nodes in a bottom up manner with at most $\mathcal{O}(\mathsf{listRank}_P(n))$ parallel I/Os. To this end, a contraction tree T' of T is computed. Based on the randomized algorithm for computing an independent set of Vishkin [21] an algorithm to compute a T' can be obtained with at most $\mathcal{O}(\mathsf{listRank}_P(n))$ parallel I/Os. Finally, the bags for $\mathcal{T}_b$ can be computed in a bottom-up manner on the vertices of T'.

The implementation of the `balance` method is a simplified version of the algorithm of Bodlaender and Hagerup [2], yielding $\ell = 7k'$, instead of $\ell = 3k'$. However this affects only constants hidden in the asymptotic notation.

Computing a Tree Decomposition of Width k: Bodlaender and Hagerup [2] use in their PRAM algorithm the algorithm of Bodlaender and Kloks [1] to compute from a tree decomposition of width ℓ for G, a tree decomposition of width k for a graph G. This is possible since the dynamic programming approach [1] is straight forward parallelizable.

Similar to their approach, in our algorithm, a modification of the algorithm for the EM model [3] is used to implement `treeDecompostionOfAtMost`. For efficient parallelization, the algorithm obtains a balanced tree decomposition of width ℓ for a graph of size n.

Large parts of the algorithm of Maheshwari and Zeh can be reused, since it heavily uses scanning, sorting and prefix sum computations. The most challenging part is to return an actual tree decomposition, and not a corresponding implicit (intermediate) representation. Their data structure "Flippable DAG", which represents a DAG such that its entire edge set can be flipped (implicitly) with one I/O, is used in this process fundamentally. Precisely, their procedure to extract from a flippable DAG an explicit DAG uses time forward processing

(TFP) [17]. Since it is not clear how TFP can be implemented efficiently in the PEM model we observe the following: in the case of computing a tree decomposition it is possible to store instead of one flippable DAG data structure a DAG and its "flip" explicitly. Therefore `treeDecompostionOfAtMost` does not rely on TFP anymore and can be implemented PEM-efficiently.

Analysis: The I/O-complexity of the three methods sketched in this section, `reduce`, `balance`, and `treeDecompostionOfAtMost`, are dominated asymptotically by sorting or list ranking. The methods are applied in the loops of Algorithm 1 repeatedly to different sizes of inputs, which geometrically decrease (respectively increase) from the initial input size n to constant size. Thus, the number of available processors increases relatively to the input size and the runtimes $\mathsf{sort}_P(n)$ and $\mathsf{listRank}_P(n)$ are not applicable anymore due to their processor bounds. The I/O complexity of list ranking equals the I/O complexity of sorting in a large parameter range but when applied with many processors (in relation to the input size) it dominates sorting. A detailed analysis yields the following term to capture the complexities of all loops:

$$\mathcal{O}\left(\sum_{j=0}^{\log n-(\log(PB^2\log^{(t)} n)-1)} \mathsf{sort}_P\left(\frac{n}{d^j}\right) + \sum_{\ell=0}^{\log B} B\log B\log^{(t)} n\log_{\frac{M}{B}}\frac{n}{B} + \sum_{m=0}^{\log(PB\log^{(t)} n)}\left(B\log^{(t)} n + \log B\log_{\frac{M}{B}}\frac{n}{B}\right)\right)$$

The first sum can be bound by $\mathcal{O}(\mathsf{sort}_P(n))$ since the input sizes are geometrically decreasing. The last sums can be bound by $\mathcal{O}(\mathsf{sort}_P(n))$ due to the restrictions on $P \leq n/(B^2\log B\log n\log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$, as stated by Theorem 3. This yields in total $\mathcal{O}(\mathsf{sort}_P(n))$ parallel I/Os for computing a tree decomposition. Note that improvements of the parameter range would directly imply improvements on the parameter range of this algorithm.

Furthermore, observe that the bound on P is compared to the list ranking bound decreased by another $\log n$ factor. This seems natural, since the PRAM algorithm [2] requires $P \leq \frac{n}{\log^2 n}$, opposed to $P \leq \frac{n}{\log n}$ for sorting or list ranking in the PRAM model.

3 Replacing Protrusions

Let G be a graph, and $X \subseteq V(G)$, then $\partial_G(X)$ is defined as the set of vertices in X having a neighbor in $V \setminus X$. Furthermore X is a *r-protrusion* of G if $|\partial_G(X)| \leq r$ and $\mathsf{tw}(G[X]) \leq r$.

In this section we present PEM-efficient implementations of the (randomized) large protrusion finder, and the protrusion replacer of Fomin et al. [6]. Both, protrusion finder and protrusion replacer, are be applied to n-vertex graphs G from classes $\mathcal{G}_H^{(\text{top})}$, for some fixed graph H. Thus, all handled graphs G

have $E(G) = \mathcal{O}(|V(G)|)$ edges, and our implementations run with $\mathcal{O}(\mathsf{sort}_P(|E(G)|)) = \mathcal{O}(\mathsf{sort}_P(n))$ parallel I/Os.

There is only one reduction rule which is applied repeatedly on the input graph G to reduce its size. This rule was introduced by Fomin et al. [14]:

If G has a τ-protrusion X, then X is replaced by a constant sized graph $Y \in \mathcal{R}_{\Pi,\tau}$, which is equivalent to X with respect to a relation $\equiv_{\Pi,\tau}$. ($\star$)

An instance (G, κ) is called *reduced* if the reduction rule cannot be applied anymore to it. By [7, Lemma 1] for any parameterized problem Π which has finite integer index (FII) in a graph class $\mathcal{G}$ there exists for every fixed $\tau \in \mathbb{N}$ such a finite set $\mathcal{R}_{\Pi,\tau}$ of representatives for the problem Π restricted to the graph class $\mathcal{G}$. The safety of the reduction rule ($\star$) is proven in [7, p. 620].

Since $\mathcal{R}_{\Pi,\tau}$ depends on τ, our kernelization algorithm (as well as the known polynomial-time kernelizations of [7,6]) is non-uniform in κ. On the other hand, in the following we may assume that $\mathcal{R}_{\Pi,\tau}$ is known explicitly to the algorithms.

By definition, $\mathcal{R}_{\Pi,\tau}$ is finite for every fixed τ and for every Π which has FII. Thus, the *protrusion limit* of Π restricted to a graph class $\mathcal{G}$ is well-defined as $\rho_{\Pi,\mathcal{G}}(\tau) = \max_{G \in \mathcal{R}_{\Pi,\tau}} |V(G)|$ [7].

The key result to be proven in this section is a PEM-efficient implementation of the reduction rule ($\star$). To this end, a randomized PEM-efficient protrusion finder yielding a set $\mathcal{P}$ of protrusions, and a PEM-efficient protrusion replacer, replacing constant sized protrusions, is presented. Since not all protrusions of $\mathcal{P}$ are of constant size, in Lemma 7, a PEM-efficient algorithm for replacing protrusions of unbound size, by constant sized protrusions, is presented.

3.1 PEM-Efficient Protrusion Finder

The protrusion finder is an algorithm that finds τ-protrusions such that every τ-protrusion X has size at least $\rho_{\Pi,\mathcal{G}}(2\tau)$. The complexity of our PEM-efficient protrusion finder is the following:

Lemma 5. *The randomized fast protrusion finder of Fomin et al. [6], for a input graph G of size n, can be implemented in the CREW PEM model with $P \leq n/(B^2 \log B \log n \log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$ taking an expected number of $\mathcal{O}(\mathsf{sort}_P(n))$ parallel I/Os. It yields a set $\mathcal{P}$ of τ-protrusions such that for every $X \in \mathcal{P}$ holds $|X| > \rho_{\Pi,\mathcal{G}}(2\tau)$.*

The proof of Lemma 5 is given in the full version of this paper. The algorithm is a straight forward implementation of the algorithm of Fomin et al. with special considerations on load balancing, using the connected components algorithm for graphs which are sparse under contraction of Arge et al. [19] and our new algorithm for tree decompositions (Section 2).

3.2 PEM-Efficient Protrusion Replacer

In this section, based on the ideas of Bodlaender et al. [22], a PEM-efficient algorithm is given for replacing all τ-protrusions in parallel.

We first deduct an algorithm to replace constant sized τ-protrusions. For this task, there is a constant time algorithm ([23, Lemma 5]) in the RAM model. A trivial simulation of this algorithm requires not more than $\mathcal{O}(1)$ I/Os, which yields the following lemma:

Corollary 6. *Let H be a graph and let Π be a parameterized graph problem with finite integer index in $\mathcal{G}_H^{(\text{top})}$. If for $\tau \in \mathbb{N}$ the set $\mathcal{R}_{\Pi,\tau}$ of representatives of $\equiv_{\Pi,\tau}$ is given, then for any τ-protrusion Y of size at most c one can decide with $\mathcal{O}(1)$ I/Os which representative $G' \in \mathcal{R}_{\Pi,\tau}$ satisfies $G' \equiv_{\Pi,\tau} G[Y]$, where the hidden constants depend only on τ and c.*

Recall that the protrusion finder of Fomin et al. finds a collection $\mathcal{P}$ of τ-protrusions X of size $|X| > \rho_{\Pi,\mathcal{G}}(2\tau)$. Since Corollary 6 can handle only protrusions of constant size, a PEM-efficient implementation of an algorithm [6] is presented, which replaces protrusions which are larger than $2\rho_{\Pi,\mathcal{G}}(2\tau)$ by smaller $(2\tau+1)$-protrusions.

Since $\mathcal{R}_{\Pi,\tau}$ exists for a problem Π for all τ, the resulting protrusions can then be replaced by Corollary 6.

Lemma 7. *Let $\tau \in \mathbb{N}$ and let Π be a graph problem with finite integer index on a graph class $\mathcal{G}$. Given for a graph $G \in \mathcal{G}$, of size n, a set of τ-protrusions $\mathcal{P}$, the expected number of parallel I/Os to find for all $X \in \mathcal{P}$ with $|X| > \rho_{\Pi,\mathcal{G}}(2\tau)$, a $(2\tau+1)$-protrusion $Y \subseteq X$ satisfying $\rho_{\Pi,\mathcal{G}}(2\tau) < |Y| < 2\rho_{\Pi,\mathcal{G}}(2\tau)$ is $\mathsf{sort}_P(n)$ in the CREW PEM model with $P \leq n/(B^2 \log B \log n \log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$.*

The algorithm of Fomin et al. [24] used for Lemma 7 can be implemented by evaluating a tree expression evaluation on the nodes of a tree decomposition.

4 Applying the Protrusion Replacer

The randomized fast protrusion replacer of Fomin et al. [6], respectively its PEM-efficient implementation, does not yield a kernelization yet. By Fomin et al. [24, Theorem 10] one application, transforming G into G', reduces, with high probability, the size of G by at least a constant fraction $r > 0$. Thus, for obtaining a linear kernel it has to be applied $\mathcal{O}(\log n)$ times. We restate Theorem 4 and prove it.

Theorem 4. *The expected number of parallel I/Os to compute a linear kernel for each of the problems of Lemma 1 and Lemma 2 is $\mathcal{O}(\mathsf{sort}_P(n))$ in the CREW PEM model with $P \leq n/(B^2 \log B \log^2 n \log^{(t)} n)$ and $M = B^{\mathcal{O}(1)}$.*

Proof. The value τ of the randomized fast protrusion replacer is chosen depending on κ as described by Fomin et al. [6] and Kim et al. [7], respectively.

Using the I/O complexities given by Lemma 5, Lemma 7, and Corollary 6 the complexity of the algorithm is split up in a term when the number of processors is bigger than the processor bound of Lemma 5 and a term capturing the part

in which the lemma does not provide optimal bounds for the remaining graphs of size at most $x = PB^2 \log B \log n \log^{(t)} n$:

$$\sum_{i=0}^{\log \frac{n}{x}-1} \mathsf{sort}_P\left(\frac{n}{r^i}\right) + \sum_{j=\log \frac{n}{x}}^{\log n} \mathsf{sort}_P(x) \leq \mathcal{O}\left(\mathsf{sort}_P(n)\right) + \mathsf{sort}_P(x) \log n$$

The second term is in $\mathcal{O}\left(\mathsf{sort}_P(n)\right)$, since $\frac{x}{PB} \log n$ is in $\mathcal{O}\left(\frac{n}{PB}\right)$ by the processor bound $P \leq n/(B^2 \log B \log^2 n \log^{(t)} n)$. Since the sorting terms are geometrically decreasing, the first term is in $\mathcal{O}\left(\mathsf{sort}_P(n)\right)$, yielding the theorem. □

5 A Permuting Lower Bound

To show a lower bound in the PEM model, we need a standard indivisibility assumption [12]. More precisely, we assume that the node-identifiers that are used to describe the edges of the graph are atomic [25].

A well known problem to which many computational problems can be reduced to is the Proximate Neighbors problem [17]. The input of the Proximate Neighbors problem are $2n$ atoms representing $x_1, \ldots, x_n, y_{\pi(1)}, \ldots, y_{\pi(n)}$. For solving the problem, a program which may be perfectly adapted to the permutation π, has to move the pair x_i, y_i into the same main memory at some time (and compare them on some CPU). Chiang et al. [17] have shown a permuting lower bound on the number of I/Os needed to solve this task in the EM model. Despite the absence of a speed up theorem [16] the lower bound can be extended to the PEM model [25].

Hence, for reasonably big B, this justifies sorting for solving the proximate neighbors problem. Based on this result the following lower bound can be obtained easily:

Theorem 5. *Every randomized algorithm that can decide if a connected planar input graph is a tree must use* $\Omega\left(\min\{n/P, \mathsf{sort}_P(n)\}\right)$ *parallel I/Os.*

A connected graph G is a tree if and only if G has a tree-decomposition of width 1. This yields a lower bound of $\Omega\left(\min\{n/P, \mathsf{sort}_P(n)\}\right)$ parallel I/Os for tree decomposition algorithms. Similarly, the problem Treewidth-1 Vertex Deletion on a planar graph, for the special case that there is no vertex deletion necessary has the same bound. This problem is covered by Lemma 1 and Lemma 2. Thus, we have matching upper and lower bounds for a large parameter range of Theorem 3 and Theorem 4.

References

1. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms 21(2), 358–402 (1996)
2. Bodlaender, H.L., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. SIAM J. Comput. 27(6), 1725–1746 (1998)
3. Maheshwari, A., Zeh, N.: I/O-efficient algorithms for graphs and bounded treewidth. Algorithmica 54(3), 413–469 (2009)

4. Flum, J., Grohe, M.: Parameterized complexity theory. Springer (2006)
5. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. System Sci. 75(8), 423–434 (2009)
6. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In: Proc. FOCS 2012, pp. 470–479 (2012)
7. Kim, E.J., Langer, A., Paul, C., Reidl, F., Rossmanith, P., Sau, I., Sikdar, S.: Linear kernels and single-exponential algorithms via protrusion decompositions. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 613–624. Springer, Heidelberg (2013)
8. Drucker, A.: New limits to classical and quantum instance compression. In: Proc. FOCS 2012, pp. 609–618 (2012)
9. Hagerup, T.: Simpler linear-time kernelization for planar dominating set. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 181–193. Springer, Heidelberg (2012)
10. van Bevern, R., Hartung, S., Kammer, F., Niedermeier, R., Weller, M.: Linear-time computation of a linear problem kernel for dominating set on planar graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 194–206. Springer, Heidelberg (2012)
11. Arge, L., Goodrich, M.T., Nelson, M., Sitchinava, N.: Fundamental parallel algorithms for private-cache chip multiprocessors. In: Proc. SPAA 2008, pp. 197–206 (2008)
12. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM 31(9), 1116–1127 (1988)
13. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
14. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: Proc. SODA 2010, pp. 503–510 (2010)
15. Komusiewicz, C., Niedermeier, R.: New races in parameterized algorithmics. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 19–30. Springer, Heidelberg (2012)
16. Greiner, G.: Sparse Matrix Computations and their I/O Complexity. Dissertation, Technische Universität München (2012)
17. Chiang, Y.J., Goodrich, M.T., Grove, E.F., Tamassia, R., Vengroff, D.E., Vitter, J.S.: External-memory graph algorithms. In: Proc. SODA, pp. 139–149 (1995)
18. Katriel, I., Meyer, U.: Elementary graph algorithms in external memory. In: Meyer, U., Sanders, P., Sibeyn, J.F. (eds.) Algorithms for Memory Hierarchies. LNCS, vol. 2625, pp. 62–84. Springer, Heidelberg (2003)
19. Arge, L., Goodrich, M.T., Sitchinava, N.: Parallel external memory graph algorithms. In: IPDPS, pp. 1–11 (2010)
20. Dadoun, N., Kirkpatrick, D.G.: Parallel construction of subdivision hierarchies. J. Comput. Syst. Sci. 39(2), 153–165 (1989)
21. Vishkin, U.: Randomized speed-ups in parallel computation. In: Proc. STOC, pp. 230–239 (1984)
22. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M. (Meta) kernelization. In: Proc. FOCS, pp. 629–638 (2009)
23. Kim, E.J., Langer, A., Paul, C., Reidl, F., Rossmanith, P., Sau, I., Sikdar, S.: Linear kernels and single-exponential algorithms via protrusion decompositions. CoRR abs/1207.0835 (2012)
24. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar $\mathcal{F}$-deletion: Approximation and optimal FPT algorithms. CoRR abs/1204.4230 (2012)
25. Jacob, R., Lieber, T., Sitchinava, N.: On the complexity of list ranking in the parallel external memory model. In: Proc. MFCS (to appear, 2014)

Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic*

Martin Lange and Etienne Lozes

School of Electrical Engineering and Computer Science
University of Kassel, Germany

Abstract. Polyadic Higher-Order Fixpoint Logic (PHFL) is a modal fixpoint logic obtained as the merger of Higher-Order Fixpoint Logic (HFL) and the Polyadic μ-Calculus. Polyadicity enables formulas to make assertions about tuples of states rather than states only. Like HFL, PHFL has the ability to formalise properties using higher-order functions. We consider PHFL in the setting of descriptive complexity theory: its fragment using no functions of higher-order is exactly the Polyadic μ-Calculus, and it is known from Otto's Theorem that it captures the bisimulation-invariant fragment of PTIME. We extend this and give capturing results for the bisimulation-invariant fragments of EXPTIME, PSPACE, and NLOGSPACE.

1 Introduction

Higher-Order Fixpoint Logic. Higher-Order Fixpoint Logic (HFL) [1] is a modal logic obtained by combining the modal μ-calculus [2] and the simply typed λ-calculus. The modal μ-calculus is found in HFL as formulas only using the base type, and consequently they denote predicates over the states of a transition system. HFL formulas of higher types which are formed using λ-abstraction for example denote predicate transformers, predicate transformer transformers and so on which can be defined recursively by means of least and greatest fixpoints.

It is known that model-checking formulas with recursive predicate transformers of order at most k is k-EXPTIME complete [3]. On the other hand, its expressiveness is poorly understood and natural questions like a capturing automaton model, a capturing game semantics, the existence of an alternation hierarchy, or the role of fixpoints in a guarded fragment have not been addressed sufficiently yet. This work provides a first step towards the understanding of the expressiveness of higher-order recursive definitions in terms of descriptive complexity.

Descriptive (Bisimulation-Invariant) Complexity. Descriptive complexity studies characterisations of classes of decision problems through means of formal

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 90–103, 2014.

descriptions of such problems, for instance through logical formulas. A logic defines a class of decision problems, namely the membership problem for the class of models of each of the logic's formulas.

One of the main aims of descriptive complexity theory is to provide characterisations of complexity classes in terms of logics, for instance Fagin's Theorem [4] stating that NP consists of *exactly* those problems which can be described by a formula of existential Second-Order Logic ($\exists\mathsf{SO}$). Thus, $\exists\mathsf{SO}$ *captures* NP.

The benefit of such capturing results are the characterisations of complexity classes without reference to a particular machine model or typical ressource bounds in terms of time and space consumption.

Many characterisations of known complexity classes in terms of logics have been found since: Second-Order Logic (SO) captures the polynomial time hierarchy PH [5], PSPACE is captured by SO enriched with a transitive closure operator [6] or, equivalently, First-Order Logic with an operator to define partial fixpoints [7], and so on. P has yet to be captured by a logic; it is known, though, that First-Order Logic with a least fixpoint operator captures P over the class of totally ordered structures [8, 9]. For a more detailed picture of results known in the area of descriptive complexity theory we refer to the respective literature [10, 11].

Another interesting result in a similar style is Otto's Theorem [12] about the polyadic μ-calculus [12,13]. The polyadic μ-calculus is a variant of the (monadic) μ-calculus where formulas denote predicates of any arity as opposed to just monadic ones. The polyadic μ-calculus, like the monadic one, cannot distinguish between bisimilar structures; thus, it can only define bisimulation-invariant graph problems [14, 15]. Moreover, model-checking algorithms for the polyadic μ-calculus are slightly similar to the ones for the monadic μ-calculus, and in particular all problems expressed in the modal μ-calculus can be decided in P. Otto's Theorem states the converse of these: if a problem p is in the class $\mathsf{P}/_{\sim}$ of problems that are both bisimulation-invariant and decidable in P, then p can be expressed by a formula of the polyadic modal μ-calculus. In other words, the polyadic modal μ-calculus captures $\mathsf{P}/_{\sim}$.

Contributions. Here we address the question of the expressiveness of higher-order fixpoints in HFL by extending Otto's Theorem to higher orders. We define PHFL, the polyadic version of HFL and we turn our attention to the first-order fragment $\mathsf{PHFL}(1)$ of PHFL. Here, the term *order* refers to the typing order of functions used in the formulas. Thus, the fragment $\mathsf{PHFL}(0)$ of order 0 contains no proper functions, and it is equal to the polyadic μ-calculus. Note that there is a difference with the term *order* used in predicate logics: the polyadic μ-calculus is in fact a fragment of second-*order* predicate logic. The two interpretations of the term *order* are closely related: the fragment of formulas with functions of typing order at most k can be seen as a fragment of order $(k+2)$ predicate logic. We simply prefer to use the typing order for the indexing of fragments because then the lowest fragment is $\mathsf{PHFL}(0)$ instead of $\mathsf{PHFL}(2)$.

Our first contribution is to show that $\mathsf{PHFL}(1)$ captures the complexity class $\mathsf{EXPTIME}/\sim$. We then turn our attention to tail-recursive functions. It is well-known that such functions are usually more space efficient than arbitrary recursive functions. Our second contribution is to give a formal account of this fact: we show that the fragment $\mathsf{PHFL}(1, \mathsf{tail})$ of order-1 tail-recursive functions captures $\mathsf{PSPACE}/\sim$. We also develop the idea of tail-recursiveness for the polyadic μ-calculus, i.e. the fragment without proper functions, and obtain a fragment $\mathsf{PHFL}(0, \mathsf{tail})$ that captures $\mathsf{NLOGSPACE}/\sim$ on structures equipped with a pre-order which induces a total order on the equivalence classes w.r.t. bisimilarity. This pre-order is to $\mathsf{NLOGSPACE}/\sim$ and $\mathsf{PHFL}(0, \mathsf{tail})$ what a total order is to P and $\mathsf{FO}[\mathsf{LFP}]$: it enables the definition of iterations via fixpoint operators. Interestingly, the cut-off point marking the apparent need for such an order in the bisimulation-invariant complexity hierarchy lies below that in the non-bismulation-invariant world, namely between $\mathsf{NLOGSPACE}/\sim$ and $\mathsf{P}/\sim$ rather than between P and NP.

Related Work. While this paper has taken the approach of characterising complexity classes classes by typing and syntactic restrictions, descriptive complexity theory predominantly has characterised complexity classes in terms of fixpoint combinators (like TC, LFP, or PFP), with the notable exception of characterisations based on Horn and Krom clauses [16].

In a different setting, higher-order grammars have been a topic intensively studied in the 80s that has recently revived in the context of verification of higher-order programs. A problem still open is whether languages defined by such grammars are context sensitive, or in other words if they belong to the complexity class $\mathsf{NLINSPACE}$ (non-deterministic linear space). Recent progresses on this problem have been achieved by Kobayashi *et al* [17], who showed that this is at least the case up to order 2 for tree languages and order 3 for word languages. Beside the fact that this line of research does not target a capturing result, the most significant difference with our work is that we consider a polyadic μ-calculus, or in different words, an automaton model that uses multiple tapes, whereas collapsible pushdown automata (the automaton model for higher-order grammars) only work with one tape.

Implicit complexity is another line of research that aims at ensuring the complexity of the execution of higher-order programs through typing. Our work here is not concerned with the time complexity of performing β-reductions or the space needed to represent the reduced terms but in the complexity of the queries defined by the formulas we consider (which are invariant under β-reduction).

Outline. In section 2 we recall Otto's Theorem that states that the polyadic μ-calculus captures $\mathsf{P}/\sim$. In section 3, we introduce the higher-order polyadic μ-calculus. Section 4 establishes that order 1 captures $\mathsf{EXPTIME}/\sim$. Section 5 studies the tail-recursive fragment and establishes that there, order 1 captures $\mathsf{PSPACE}/\sim$ and order 0 $\mathsf{NLOGSPACE}/\sim$. Due to space constraints, some details are missing and can be found in a longer version [18].

2 Background

Labeled Transition Systems, Bisimulation, and Queries. A *labeled transition system* (LTS) is a tuple $\mathfrak{M} = (Q, \Sigma, P, \Delta, v)$, where $Q = \{q, r, \dots\}$ is a set of states, $\Sigma = \{a, b, \dots\}$ is a finite set of actions, $P = \{p, \dots\}$ is a finite set of propositions, $\Delta \subseteq Q \times \Sigma \times Q$ is the set of labeled transitions, and $v : P \to 2^Q$ is a valuation that associates to every proposition a set of states. We write $q_1 \xrightarrow{a} q_2$ for $(q_1, a, q_2) \in \Delta$ and $q \models p$ for $q \in v(p)$.

A binary relation $R \subseteq Q^2$ is a *bisimulation* if it is a symmetric relation, and for every pair of states $(q_1, q_2) \in R$, it holds that (1) for all $a \in \Sigma$, for all $q_1' \in Q$, if $q_1 \xrightarrow{a} q_1'$, then there is $q_2' \in Q$ such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \, R \, q_2'$, and (2) for all $p \in P$, if $q_1 \models p$, then $q_2 \models p$. Two states q_1, q_2 are *bisimilar*, written $q_1 \sim q_2$, if there is a bisimulation that contains the pair (q_1, q_2).

We assume a fixed encoding of a finite LTS $\mathfrak{M} = (Q, \Sigma, P, \Delta, v)$ as a word $w_{\mathfrak{M}}$ such that $|w_{\mathfrak{M}}|$ is linear in $|Q| \cdot |P| + |\Delta|$ (for instance, using a sparse matrix representation). An r-adic *query* $\mathcal{Q}$ is a set of tuples $(\mathfrak{M}, q_1, \dots, q_r)$ where $\mathfrak{M}$ is an LTS and $q_1, \dots, q_r$ are states of $\mathfrak{M}$. A query $\mathcal{Q}$ is said to belong to a complexity class $\mathcal{C}$ if the language of encodings of $\mathcal{Q}$ is in $\mathcal{C}$. A query $\mathcal{Q}$ is said to be *bisimulation-invariant* if for every two tuples $(\mathfrak{M}, \boldsymbol{q})$ and $(\mathfrak{M}', \boldsymbol{q}')$ such that $q_i \sim q_i'$ for all i, $(\mathfrak{M}, \boldsymbol{q}) \in \mathcal{Q}$ if and only if $(\mathfrak{M}', \boldsymbol{q}') \in \mathcal{Q}$.

Example 1. Let $\mathcal{Q}$ be the binary query consisting of tuples $(\mathfrak{M}, q, q')$ such that $q \sim q'$. Since bisimilarity can be decided in P, this query is in P. Moreover, since bisimilarity is a transitive relation, this query is bisimulation-invariant.

The Polyadic μ-calculus. A formula of the (monadic) modal μ-calculus is often interpreted as a game played by two players (sometimes called Prover and Refuter) that alternatively move a single pebble along the transitions of an LTS. The polyadic μ-calculus is basically a multi-pebble version of this game: $d \geqslant 1$ pebbles are disposed on the states of a LTS, and for every modality $\langle a \rangle_i$ (respectively $[a]_i$), Prover (respectively Refuter) has to move the i-th pebble along an a-transition. Moreover, formulas can use the modality $\{i{\leftarrow}j\}$, that corresponds, in the game interpretation, to moving the i-th pebble to the same place as the j-th pebble.

Let $\mathsf{Var} = \{X, Y, Z, \dots\}$ be some fixed set of variables. Formulas of the polyadic μ-calculus $\mathcal{L}_\mu^\omega$ are given by the following grammar

$$\Phi, \Psi \ ::= \top \mid p_i \mid \Phi \vee \Psi \mid \neg\Phi \mid \langle a \rangle_i \Phi \mid \{\boldsymbol{i}{\leftarrow}\boldsymbol{j}\}\Phi \mid X \mid \mu X.\Phi$$

where $\boldsymbol{i} = (i_1, \dots, i_n)$ and $\boldsymbol{j} = (j_1, \dots, j_n)$ are equal-length tuples of natural numbers. As usual, we only consider formulas in which every bound variable occurs underneath an even number of negations counting from its μ-binder. We also use standard notations for derived logical connectives, namely $\wedge$, $\Rightarrow$, $\Leftrightarrow$, and $[a]_i$ for conjunction, implication, equivalence, and necessity respectively. A formula is d-adic if in each subformula p_i, $\langle a \rangle_j \Phi$, and $\{\boldsymbol{i}{\leftarrow}\boldsymbol{j}\}\Phi$ the indices $i, j, i_1, \dots, i_n, j_1, \dots, j_n$ are in $\{1, \dots, d\}$.

The semantics of a d-adic formula Φ is a set $[\![\Phi]\!]^d_{\mathfrak{M}}$ of d-tuples of states (see [12], and also Section 3). The r-adic query $\mathcal{Q}^r_\Phi$ associated to a closed d-adic formula Φ is the set of tuples $(\mathfrak{M}, q_1, \ldots, q_r)$ such that there is $\boldsymbol{s} \in [\![\Phi]\!]^d_{\mathfrak{M}}$ with $q_i = s_i$ for all $i = 1, \ldots, \min(r, d)$.

Example 2. A standard example of a 2-adic formula is $\Phi_\sim :=$

$$\nu X. \bigwedge_{a\in\Sigma} [a]_1\langle a\rangle_2 X \wedge [a]_2\langle a\rangle_1 X \wedge \bigwedge_{p\in P} p_1 \Leftrightarrow p_2$$

which denotes the set of pairs (q_1, q_2) such that $q_1 \sim q_2$. Thus, $\Phi_\sim$ defines bisimilarity [12, 13], and $\mathcal{Q}^2_\Phi$ is the same query as in Example 1.

Theorem 1 (Otto [12]). *Let $\mathcal{Q}$ be an r-adic query. The following two are equivalent. (1) $\mathcal{Q}$ is bisimulation-invariant and in* P*; (2) $\mathcal{Q} = \mathcal{Q}^r_\Phi$ for some $\Phi \in \mathcal{L}^\omega_\mu$.*

As a consequence of Otto's Theorem, we get for example that trace equivalence is not expressible in the polyadic modal μ-calculus (unless $\mathsf{P} = \mathsf{PSPACE}$), because of the PSPACE-completeness of trace equivalence [19].

3 A Polyadic Higher-Order Fixpoint Logic

In this section, we introduce the polyadic higher-order fixpoint logic, a logic that extends the polyadic modal μ-calculus with higher-order fixpoints *à la* Viswanathan and Viswanathan [1]. In Viswanathans' logic, order-0 formulas denote predicates, order-1 formulas denote *predicate transformers*, i.e. functions mapping predicates to predicates, and so on for higher orders. For instance, $(\lambda F.\ \lambda X.\ F\ (F\ X))\ \ (\lambda Y.\ \langle a\rangle Y)\ \ \top$ is equivalent to the formula $\langle a\rangle\langle a\rangle\top$. Moreover, the least fixpoint combinator can be applied to monotone predicate transformers of any order. For instance, the formula

$$\Big(\mu G.\ \lambda F.\ \lambda X.\ (F\ X) \vee (G\ (\lambda Z.\ F\ (F\ Z))\ X)\Big)\ \ (\lambda Y.\ \langle a\rangle Y)\ \ \top$$

is equivalent to the infinitary disjunction $\bigvee_{n\geqslant 0}\langle a\rangle^{2^n}\top$.

Formally, *formulas* $\Phi, \Psi, \ldots$, *types* $\tau, \sigma, \ldots$ and *variances* v of the polyadic higher-order fixpoint logic ($\mathsf{PHFL}(\omega)$) are defined by the grammar

$$v ::= + \mid - \mid 0 \qquad \sigma, \tau ::= \bullet \mid \sigma^v \to \tau$$
$$\Phi, \Psi ::= \top \mid p_i \mid \Phi \vee \Psi \mid \neg\Phi \mid \langle a\rangle_i\Phi \mid \{\boldsymbol{i}\leftarrow\boldsymbol{j}\}\Phi \mid X \mid \lambda X^{v,\tau}.\Phi \mid \Phi\ \Psi \mid \mu X^\tau.\Phi$$

where $X, Y, \ldots$ range over a finite set of variables, and i, j range over the set $\mathbb{N}$ of natural numbers. We use standard notations like $\Phi \wedge \Psi$, $[a]_i\Phi$, $\nu X^\tau.\Phi$, or $\Phi \Leftrightarrow \Psi$ for dual and derived connectives. The maximal arity $\mathsf{ma}(\tau)$ of a type τ is defined by induction on τ: $\mathsf{ma}(\bullet) = 1$, and $\mathsf{ma}(\tau_1 \to \tau_2 \to \cdots \to \tau_n \to \bullet) = \max(\{n\} \cup \{\mathsf{ma}(\tau_i) \mid i = 1, \ldots, n\})$. The order $\mathsf{ord}(\tau)$ of a type τ is defined by induction on τ: $\mathsf{ord}(\bullet) = 0$, and $\mathsf{ord}(\sigma \to \tau) = \max(1 + \mathsf{ord}(\sigma), \mathsf{ord}(\tau))$. The order

$$\Gamma \vdash \top : \bullet \qquad \Gamma \vdash p_i : \bullet \qquad \frac{\Gamma \vdash \Phi : \bullet}{\Gamma \vdash \langle a \rangle_i \Phi : \bullet} \qquad \frac{\Gamma \vdash \Phi : \bullet}{\Gamma \vdash \{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Phi : \bullet} \qquad \frac{\neg\Gamma \vdash \Phi : \tau}{\Gamma \vdash \neg\Phi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \tau \qquad \Gamma \vdash \Psi : \tau}{\Gamma \vdash \Phi \vee \Psi : \tau} \qquad \frac{v \in \{+, 0\}}{\Gamma,\ X^v : \tau \vdash X : \tau} \qquad \frac{\Gamma, X^v : \sigma \vdash \Phi : \tau}{\Gamma \vdash \lambda X^{v,\sigma}.\ \Phi : \sigma^v \to \tau}$$

$$\frac{\Gamma, X^+ : \tau \vdash \Phi : \tau}{\Gamma \vdash \mu X^\tau.\ \Phi : \tau} \qquad \frac{\Gamma \vdash \Phi : \sigma^+ \to \tau \qquad \Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \sigma^- \to \tau \qquad \neg\Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau} \qquad \frac{\Gamma \vdash \Phi : \sigma^0 \to \tau \qquad \Gamma \vdash \Psi : \sigma \qquad \neg\Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

Fig. 1. The type system of $\mathsf{PHFL}(\omega)$. The type environment $\neg\Gamma$ is the one in which every assumption $X^v : \tau$ is replaced with $X^{-v} : \tau$

of a formula Φ is $\max\{\mathsf{ord}(\tau) \mid \mu X^\tau.\Psi \text{ is a subformula of } \Phi\}$. We write $\mathsf{PHFL}(k)$ for the set of formulas where recursive predicates are annotated with types of order at most k. In particular, $\mathsf{PHFL}(0)$ is the polyadic modal μ-calculus $\mathcal{L}_\mu^\omega$. On the other hand, the 1-adic fragment of $\mathsf{PHFL}(\omega)$ is exactly Viswanathans' Higher-Order Modal Fixpoint Logic.

Given a set A and a bolean algebra B, the set of functions $f : A \to B$ is again a boolean algebra (for instance, $(\neg_{A\to B} f)(x) = \neg_B(f(x))$). A function $f : A \to B$ has variance $+$ if it is monotone, $-$ if $\neg_{A\to B} f$ is monotone, and 0 in any case. We associate to every type τ the boolean algebra D_τ as follows: (1) $D_\bullet$ is the set $\mathcal{P}(Q^d)$ of all d-adic predicates, and (2) if $\tau = \sigma^v \to \sigma'$, then D_τ is the set of functions $f : D_\sigma \to D_{\sigma'}$ that have variance v.

A term $\lambda X^{v,\tau}.\Phi$ denotes a function that expects an argument of type τ and has variance v in this argument. A type judgement is a judgement of the form $X_1^{v_1,\tau_1}, \ldots, X_n^{v_n,\tau_n} \vdash \Phi : \tau$. We say that a type judgement is derivable if it admits a derivation tree according to the rules of Figure 1.

A formula Φ is well-typed if $\vdash \Phi : \tau$ is derivable for some τ. In the remainder, we always implicitly assume that we are working with well-typed formulas and sometimes omit the type annotations.

The semantics of a well-typed formula Φ of type τ is a predicate or function living in D_τ that we are now about to define. The interpretation $[\![\Gamma]\!]$ of a type environment is the set of maps ρ that send each variable $X^v : \tau \in \Gamma$ to $\rho(X) \in D_\tau$. We write $\rho[X \mapsto \mathcal{X}]$ for the map ρ' that is equal to ρ except for $\rho'(X) = \mathcal{X}$. The interpretation $[\![\Gamma \vdash \Phi : \tau]\!]$ is a map from $[\![\Gamma]\!]$ to D_τ defined by induction on Φ as explained of Figure 2 (remember that $D_\bullet = \mathcal{P}(Q^d)$ is the set of all d-adic predicates).

Proposition 1. *For every formula Φ of ground type, for all $d \geqslant 1$, the query $\mathcal{Q}_\Phi^d$ is bisimulation-invariant.*

Proposition 1 can be proved for instance by extending Viswanathans' proof of the bisimulation-invariance of Higher-Order Fixpoint Logic [1] to polyadic

$$
\begin{aligned}
[\![\Gamma \vdash \top : \bullet]\!](\rho) &= Q^d \\
[\![\Gamma \vdash \langle a\rangle_i \Phi : \bullet]\!](\rho) &= \{\boldsymbol{q} \in Q^d \mid \exists \boldsymbol{q}' \in [\![\Gamma \vdash \Phi : \bullet]\!].\ \boldsymbol{q} \xrightarrow{a,i} \boldsymbol{q}'\} \\
[\![\Gamma \vdash \Phi \vee \Psi : \tau]\!](\rho) &= [\![\Gamma \vdash \Phi : \tau]\!](\rho) \sqcup_\tau [\![\Gamma \vdash \Psi : \tau]\!](\rho) \\
[\![\Gamma \vdash \neg\Phi : \tau]\!](\rho) &= \neg_\tau [\![\neg(\Gamma) \vdash \Phi : \tau]\!](\rho) \\
[\![\Gamma \vdash \{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Phi : \bullet]\!](\rho) &= \{\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}(\boldsymbol{q}) \mid \boldsymbol{q} \in [\![\Gamma \vdash \Phi : \bullet]\!](\rho)\} \\
[\![\Gamma, X : \tau \vdash X : \tau]\!](\rho) &= \rho(X) \\
[\![\Gamma \vdash \mu X^\tau.\Phi]\!](\rho) &= \mathsf{LFP}\, [\![\Gamma \vdash \lambda X^{+,\tau}.\ \Phi]\!](\rho) \\
[\![\Gamma \vdash \lambda X^{v,\sigma}.\ \Phi : \sigma^v \to \tau]\!](\rho) &= \mathcal{X} \mapsto [\![\Gamma, X^v : \sigma \vdash \Phi : \tau]\!](\rho[X \mapsto \mathcal{X}]) \\
[\![\Gamma \vdash \Phi\ \Psi : \tau]\!](\rho) &= [\![\Gamma \vdash \Phi : \sigma^v \to \tau]\!](\rho)([\![\Gamma \vdash \Psi : \sigma]\!](\rho))
\end{aligned}
$$

where $\boldsymbol{q} \xrightarrow{a,i} \boldsymbol{q}'$ stands for $q_i \xrightarrow{a} q_i'$ and $q_j = q_j'$ for all $j \neq i$.

Fig. 2. Semantics of $\mathsf{PHFL}(\omega)$

formulas. Furthermore, using fixpoint unfolding and β-reduction it is also possible to see that over every *set* of structures, $\mathsf{PHFL}(\omega)$ is equivalent to infinitary polyadic modal logic, i.e. one with arbitrary disjuncts and conjuncts and no fixpoint quantifiers, no λ-abstraction and no formula application.

Example 3. Let $\Phi := \left(\nu F.\lambda X, Y.X \Leftrightarrow Y \wedge \bigwedge_{a\in\Sigma} F\,\langle a\rangle_1 X\,\langle a\rangle_2 Y\right) \top\ \top$. Then Φ can be unfolded to

$$
\begin{aligned}
F\ \top\ \top &= \top \Leftrightarrow \top \ \wedge\ \textstyle\bigwedge_{a\in\Sigma} F\,\langle a\rangle_1 \top\,\langle a\rangle_2 \top \\
&= \textstyle\bigwedge_{a\in\Sigma} \langle a\rangle_1 \top \Leftrightarrow \langle a\rangle_2 \top \ \wedge\ \bigwedge_{b\in\Sigma} F\,\langle ba\rangle_1 \top\,\langle ba\rangle_2 \top \\
&= \textstyle\bigwedge_{w\in\Sigma^*} \langle w\rangle_1 \top \Leftrightarrow \langle w\rangle_2 \top,
\end{aligned}
$$

where $\langle w\rangle_i$ stands for $\langle w_1\rangle_i\langle w_2\rangle_i \ldots \langle w_n\rangle_i$. Thus, Φ denotes those pairs (q_1, q_2) for which q_1 and q_2 have exactly the same traces, i.e Φ defines trace equivalence.

4 Capturing EXPTIME/$\sim$

The aim of this section is to show that the first-order fragment of PHFL, i.e. $\mathsf{PHFL}(1)$ captures the class of bisimulation-invariant queries which can be evaluated in deterministic exponential time. For one part of this result we show the stronger statement that queries of order k can be evaluated in k-fold exponential time.

Theorem 2. *Let $r, k \geqslant 1$, and $\Phi \in \mathsf{PHFL}(k)$. Then the query $\mathcal{Q}^r_\Phi$ is in k-*EXPTIME.

The proof essentially follows the same ideas as in the case of the (1-adic) Higher-Order Fixpoint Logic [3] (see [18] for details).

Theorem 2 shows in particular that all queries expressible in $\mathsf{PHFL}(1)$ are in EXPTIME/$\sim$. We now consider the converse implication and aim at a proof of the following: if a query is in EXPTIME/$\sim$, then it can be expressed by a $\mathsf{PHFL}(1)$ formula. A direct, but tedious proof would encode the run of an EXPTIME Turing

machine by a query. A more elegant proof can be obtained by making use of Immerman's characterisation of EXPTIME queries over structures [6] as those that are expressible in second-order logic with least fixed points (SO[LFP]). The proof then proceeds in two steps: first, we transfer Immerman's result to the logic SO[LFP]/$\sim$ defined over labeled transition systems, and second we show how to encode SO[LFP]/$\sim$ into PHFL(1).

The first step is relatively easy with the main problem simply being a correct definition of the semantics of SO[LFP]/$\sim$ over LTS. We define the formulas of SO[LFP] (resp. SO[LFP]/$\sim$) as the ones derivable from the grammar

$$\Phi, \Psi \ ::= p(x) \mid a(x,y) \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists x.\Psi \mid \\ \exists X.\Psi \mid \mathsf{LFP}(F, \boldsymbol{X}, \Phi)(\boldsymbol{Y})(\boldsymbol{x}) \mid X(\boldsymbol{x}) \mid F(\boldsymbol{X})(\boldsymbol{x})$$

The semantics of SO[LFP] on labeled graphs is as expected (see for instance [6]). Now, in order to define the semantics of SO[LFP]/$\sim$ over LTS, we just need to map every LTS to a labeled graph and interpret the formula over this graph with the SO[LFP] semantics.

We call a tuple $(\mathfrak{M}, q_1, \ldots, q_r)$ *reduced* if all states of $\mathfrak{M}$ are reachable from at least one q_i, and if $\sim$ coincides with equality. To every tuple $(\mathfrak{M}, \boldsymbol{q})$, we associate the reduced tuple $\mathsf{RED}(\mathfrak{M}, \boldsymbol{q})$ obtained by quotienting with respect to $\sim$ and pruning all states that cannot be reached from at least one q_i. We say that a tuple $(\mathfrak{M}, \boldsymbol{q})$ satisfies a formula Φ of SO[LFP]/$\sim$ if the graph $\mathsf{RED}(\mathfrak{M}, \boldsymbol{q})$ satisfies Φ in the graph semantics.

Lemma 1. *A query $\mathcal{Q}$ is in* EXPTIME/$\sim$ *iff if it is definable in* SO[LFP]/$\sim$.

We refer to [18] for a detailed proof of this result, and now move to the more challenging part, namely the encoding of SO[LFP]/$\sim$ into PHFL(1). For every SO[LFP]/$\sim$ formula Φ with free second-order variables $X_1, \ldots, X_n$, we define a PHFL(1) formula Ψ with the same free (order 0) variables, so that the least fixpoint in SO[LFP]/$\sim$ is naturally represented by a least fixpoint in PHFL(1). First-order variables of SO[LFP]/$\sim$ are encoded differently. Without loss of generality, we may assume an enumeration $x_1, \ldots, x_r, x_{r+1}, \ldots x_d$ of all variables of the SO[LFP] formula, such that $x_1, \ldots, x_r$ are the free variables and $x_{r+1}, \ldots, x_d$ are the quantified ones. We thus code $p(x_i)$ and $a(x_i, x_j)$ as p_i and $\langle a \rangle_i \{1, 2 \leftarrow i, j\} \Phi_\sim$ respectively, where $\Phi_\sim$ is the formula that defines $\sim$. For $i > r$, we define the macro $\exists_i \Phi := \bigvee_{j=1}^{r} \{i \leftarrow j\} \mu X.\Phi \vee \bigvee_{a \in \Sigma} \langle a \rangle_i X$, where Φ is an arbitrary PHFL(1) formula in which X does not occur. Then $\exists_i \Phi$ defines the set of all tuples for which Φ holds once the i-th component has been replaced by some state reachable from one of the states denoted by $x_1, \ldots, x_r$. Due to the bisimulation-invariant semantics of SO[LFP]/$\sim$, this is enough to encode a first-order quantification.

We are thus left with the encoding of second-order quantifiers. There is no obvious way of adapting the same idea we used for first-order quantifiers, and our encoding of second-order quantifiers is significantly trickier.

Let us first recall that it is possible to define a 2-adic formula $\Phi_\prec$ that defines a transitive relation $\prec$ such that $\prec \cap \succ = \varnothing$ and $\prec \cup \succ = \not\sim$; we refer to Otto's

work [12] where this formula is the crux of the proof that the polyadic μ-calculus captures $\mathsf{P}/_\sim$. Let $\mathfrak{M}$ be a reduced LTS, so that $\prec$ defines a total order on states, and let $\prec_{\mathsf{lex}}$ denote the lexicographic extension of $\prec$ over Q^d.

Lemma 2. *There is a predicate transformer $\langle\mathsf{dec}\rangle$ such that for every formula Φ, $\langle\mathsf{dec}\rangle\ \Phi$ denotes the upward closure of Φ with respect to $\prec_{\mathsf{lex}}$.*

The construction of $\langle\mathsf{dec}\rangle$ is rather straightforward (see [18]).

Lemma 3. *Consider the predicate transformer*

$$\mathsf{next} \ := \ \lambda X.\ (\neg X \wedge \neg\langle\mathsf{dec}\rangle\neg X) \vee (X \wedge \langle\mathsf{dec}\rangle\neg X).$$

Then, for any predicate $\mathcal{X} \in \mathcal{P}(Q^d)$, there is $i \geqslant 0$ such that $\mathsf{next}^i(\varnothing) = \mathcal{X}$.

Proof. Consider the bijection $f : \mathcal{P}(Q^d) \to \{0, \ldots, 2^{|Q|^d} - 1\}$ defined by associating to every predicate $\mathcal{X} \in \mathcal{P}(Q^d)$ the integer $f(\mathcal{X})$ whose binary representation $b_1 b_2 \ldots b_{|Q|^d}$ is such that the ith bit b_i is equal to 1 if and only if the i-th element $\boldsymbol{q}$ of Q^d with respect to $\prec_{\mathsf{lex}}$ is in $\mathcal{X}$.

Our claim is that next maps every predicate $\mathcal{X}$ to the predicate $\mathcal{Y}$ such that $f(\mathcal{Y}) = 1 + f(\mathcal{X})$ modulo $2^{|Q|^d}$. Indeed, the ith bit in $\mathcal{Y}$ is 1 if either it is also 1 in $\mathcal{X}$ and a lower bit is 0 in $\mathcal{X}$, or it is 0 in $\mathcal{X}$ but all lower bits are 1 in $\mathcal{X}$. □

Let Φ be a $\mathsf{PHFL}(1)$ formula not containing the variable H, and let formula $\exists X.\Phi$ be defined as $\big(\mu H.\lambda X.\Phi \vee H\ (\mathsf{next}\ X)\big)\ \bot$. Then, thanks to Lemma 3, $\exists X.\Phi$ encodes a second-order existential quantification.

Lemma 4. *Let $r \geqslant 1$. For every formula Φ of $\mathsf{SO}[\mathsf{LFP}]/_\sim$, there is a formula Ψ of $\mathsf{PHFL}(1)$ such that $\mathcal{Q}^r_\Phi = \mathcal{Q}^r_\Psi$.*

Theorem 3. $\mathsf{PHFL}(1)$ *captures* $\mathsf{EXPTIME}/_\sim$ *over labeled transition systems.*

Proof. Lemmas 1 and 4 prove that every $\mathsf{EXPTIME}/_\sim$ query is expressible in $\mathsf{PHFL}(1)$. By Theorem 2 and Proposition 1, we know that $\mathsf{PHFL}(1)$ cannot express more than that. □

5 Tail Recursion and PSPACE/$_\sim$

Tail-recursive functions are functions that are never called recursively in intermediate steps of their body, either for evaluating a condition on branching, or for evaluating an argument of a function call. By analogy, we define tail-recursive formulas as the ones that can be seen as non-deterministic tail-recursive functions.

We assume from now on that the logical connective $\wedge$ is primitive in the syntax (and not just the dual of $\vee$). Without loss of generality, we restrict our attention to formulas in which every variable is bound at most once. We say that a formula $\mu X.\Phi$ is a tail-recursive definition if no subformula of Φ is of the form either $\Phi'\ \Psi$, or $\neg\Psi$, or $\Psi \wedge \Phi'$, with X occuring free in Ψ (see Figure 3 for

$$\bar{Y} \vdash \mathsf{tail}(p_i, \bar{X}) \qquad \frac{X \in \bar{X} \cup \bar{Y}}{\bar{Y} \vdash \mathsf{tail}(X, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}(\neg\Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Phi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X}) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\Phi \vee \Psi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\langle a \rangle_i \Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\Phi \wedge \Psi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}([a]_i \Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X}) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}(\Phi\ \Psi, \bar{X})} \qquad \frac{\bar{Y} \cup \{Z\} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\lambda Z^{v,\tau}.\Phi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X} \cup \{Z\})}{\bar{Y} \vdash \mathsf{tail}(\mu Z^{\tau}.\Phi, \bar{X})}$$

Fig. 3. A closed formula Φ is tail-recursive if $\varnothing \vdash \mathsf{tail}(\Phi, \varnothing)$ is derivable

an inductive definition). Observe that we therefore do not treat both sides of a conjunction symmetrically. A formula is tail recursive if every recursive definition is tail-recursive. We write $\mathsf{PHFL}(i, \mathsf{tail})$ for the set of tail-recursive formulas of $\mathsf{PHFL}(i)$.

Example 4. The formula $(\mu F.\lambda X.(F\ \langle a \rangle_1 X) \vee (X \wedge \langle a \rangle_2 (F\ X))\ (\mu Y.Y)$ is tail-recursive. On the other hand, the formula $\mu X.[a]_1 X$ is not tail-recursive because X occurs underneath $[a]_1$ (see Figure 3). The formula $\mu F.\lambda X.(F\ X) \wedge (F\ (F\ X))$ is not tail-recursive either, for two different reasons: on the one hand F occurs on the left side of $\wedge$, and on the other hand F occurs in the argument $F\ X$ of F.

Theorem 4. *Let $r > 0$ and $\Phi \in \mathsf{PHFL}(1, \mathsf{tail})$. Then $\mathcal{Q}^r_\Phi$ is in* PSPACE.

Proof. The proof of this result is slightly more complicated than for Theorem 2. To achieve EXPTIME, a global model-checking algorithm that closely follows the semantics of $\mathsf{PHFL}(1)$ is enough. However, such an algorithm needs to represent functions denoted by predicate transformer in extension, which requires exponential space. If we want a model-checking algorithm running in PSPACE, we need to avoid representing functions in extension.

For a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ, let the *recursion depth* $\mathsf{rd}(\Phi)$ of Φ be inductively defined as follows: $\mathsf{rd}(p_i) = \mathsf{rd}(X) = 0$, $\mathsf{rd}(\Phi_1 \vee \Phi_2) = \max(\mathsf{rd}(\Phi_1), \mathsf{rd}(\Phi_2))$, $\mathsf{rd}(\Phi_1 \wedge \Phi_2) = \max(\mathsf{rd}(\Phi_2), 1 + \mathsf{rd}(\Phi_1))$, $\mathsf{rd}(\Phi_1\ \Phi_2) = \max(\mathsf{rd}(\Phi_1, 1 + \mathsf{rd}(\Phi_2))$, $\mathsf{rd}(\langle a \rangle_i \Phi) = \mathsf{rd}(\lambda X.\Phi) = \mathsf{rd}(\mu X.\Phi) = \mathsf{rd}(\Phi)$, and $\mathsf{rd}(\neg\Phi) = 1 + \mathsf{rd}(\Phi)$.

If π is a list $\mathcal{X}_1, \ldots, \mathcal{X}_n$ of elements of $\mathcal{P}(Q^d)$, we write $\mathsf{hd}(\pi)$ for $\mathcal{X}_1$ and $\mathsf{tl}(\pi)$ for $\mathcal{X}_2, \ldots, \mathcal{X}_n$, and $\mathcal{X} :: \pi$ for the list π' with $\mathsf{hd}(\pi') = \mathcal{X}$ and $\mathsf{tl}(\pi') = \pi$. For simplicity, we assume without loss of generality that we work with formulas such that every variable is bound at most once. We call a variable *recursive* if it is bound by a μ. For a function $c : \mathsf{Var} \to \mathbb{N}$ we write $c[X{+}{+}]$ for the function defined by $c[X{+}{+}](X) = 1 + c(X)$ and $c[X{+}{+}](Y) = c(Y)$ for all $Y \neq X$.

For a tuple $\boldsymbol{q} \in Q^d$, a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ, a list $\pi \in \mathcal{P}(Q^d)^*$, a function $\rho : \mathsf{Var} \to \mathcal{P}(Q^d)$ and a function $c : \mathsf{Var} \to \mathbb{N}$, let $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ be the non-deterministic recursive procedure defined as follows:

- if Φ is an atomic formula, return true if $\boldsymbol{q} \in [\![\Phi]\!](\rho)$, false otherwise;
- if $\Phi = \{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Psi$, return $\mathbf{check}(\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}(\boldsymbol{q}), \Psi, \pi, \rho, c)$;
- if $\Phi = X$ and X is not a recursive variable, return true if $\boldsymbol{q} \in \rho(X)$, false otherwise;
- if $\Phi = X$ for a recursive variable X with $\mathsf{fp}_X = \mu X^\tau.\Psi$, let $N := |Q|^d$ if X is order 0, otherwise X is a l-ary predicate transformer for some $l > 0$, and we set $N := |Q|^{ld} \cdot 2^{l|Q|^d}$; if $c(X) = N$, return false, otherwise return $\mathbf{check}(\boldsymbol{q}, \Psi, \pi, \rho, c[X{+}{+}])$;
- if $\Phi = \neg\Psi$, return $\neg\mathbf{check}(\boldsymbol{q}, \Psi, \pi, \rho, c)$;
- if $\Phi = \Phi_1 \vee \Phi_2$, guess $i \in \{1, 2\}$ and return $\mathbf{check}(\boldsymbol{q}, \Phi_i, \mathcal{X}, \pi, \rho, c)$;
- if $\Phi = \langle a \rangle_i \Psi$, guess $\boldsymbol{s}$ such that $\boldsymbol{q} \xrightarrow{a,i} \boldsymbol{s}$ and return $\mathbf{check}(\boldsymbol{s}, \Psi, \pi, \rho, c)$;
- if $\Phi = \Psi_1\, \Psi_2$, compute first the set of tuples $\mathcal{X} := \{\boldsymbol{r} \mid \mathbf{check}(\boldsymbol{r}, \Psi_2, \pi, \rho, c) = \mathsf{true}\}$, then return $\mathbf{check}(\Psi_1, \mathcal{X} :: \pi, \rho, c)$;
- if $\Phi = \Phi_1 \wedge \Phi_2$, return false if $\mathbf{check}(\boldsymbol{q}, \Phi_1, \pi, \rho, c) = \mathsf{false}$, otherwise return $\mathbf{check}(\boldsymbol{q}, \Phi_2, \pi, \rho, c)$;
- if $\Phi = \lambda X.\Psi$, return $\mathbf{check}(\Psi, \mathsf{tl}(\pi), \rho[X \mapsto \mathsf{hd}(\pi)], c)$;
- if $\Phi = \mu X^\tau.\Phi'$, return $\mathbf{check}(\boldsymbol{q}, \Phi', \pi, \rho, c[X \mapsto 0])$.

Consider a fixed d-adic formula Φ with l variables and recursion depth $\mathsf{rd}(\Phi) = k$, and an LTS $\mathfrak{M}$ with n states. Encoding sets as bit vectors and integers in binary, ρ, π and c require $\mathcal{O}(n^{dl})$ space, whereas encoding $\boldsymbol{q}$ requires $\mathcal{O}(d \cdot \log n)$. Moreover, if we avoid to stack the calling context at every tail recursive call (recursive calls of the form $\mathsf{return}\ \mathbf{check}(\dots)$), then the height of the stack of calling contexts is bounded by the recursion depth k of the formula. So $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ requires overall space $\mathcal{O}(k \cdot n^{dl})$. As a consequence, for a fixed parameter formula Φ, the procedure works in $\mathsf{NPSPACE}$, and by Savitch's theorem we get that $\mathcal{Q}^r_\Phi$ is in PSPACE. □

We are now interested in the proof of the converse of Theorem 4. In order to establish that $\mathsf{PHFL}(1, \mathsf{tail})$ captures $\mathsf{PSPACE}/_\sim$, we again pick a logic that captures PSPACE over graphs, transfer this result to transition systems, and encode this logic in $\mathsf{PHFL}(1, \mathsf{tail})$. The logic we consider is first-order logic with partial fixpoints ($\mathsf{FO}[\mathsf{PFP}]$) introduced by Abiteboul and Vianu [7]. The partial fixpoint $\mathsf{PFP}(f)$ of a predicate transformer $f : \mathcal{P}(Q^d) \to \mathcal{P}(Q^d)$ is the predicate defined as follows: if there is some $i \geqslant 0$ for which $f^i(\varnothing) = f^{i+1}(\varnothing)$, then $\mathsf{PFP}(f) := f^i(\varnothing)$; otherwise $\mathsf{PFP}(f) := \varnothing$. We only detail the encoding of this partial fixpoint combinator in $\mathsf{PHFL}(1, \mathsf{tail})$, since the rest of the proof does not significantly differ from the proofs of Lemmas 1 and 4 (see [18] for details).

Let $\forall_i\Phi$ denote the formula $\neg\exists_i\neg\Phi$, where $\exists_i\Phi$ is the formula we introduced for encoding first-order quantifiers. For a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ with a free variable X, let $\Psi := \big(\mu F.\lambda X.(X \wedge \forall_1 \dots \forall_d(X \Leftrightarrow \Phi)) \ \vee \ F\ \Phi\big)\ \bot$. It can be checked that Ψ is tail-recursive and order 1, and that it defines the partial fixpoint of $\lambda X.\Phi$.

Theorem 5. $\mathsf{PHFL}(1, \mathsf{tail})$ *captures* $\mathsf{PSPACE}/_{\sim}$ *over labeled transition systems.*

Next we consider tail-recursive formulas of order 0. The same algorithm that we used in the proof of Theorem 4 has a better space complexity for formulas of $\mathsf{PHFL}(0)$.

Theorem 6. *For all* $r \geqslant 0$*, for every formula* Φ *of* $\mathsf{PHFL}(0, \mathsf{tail})$*,* $\mathcal{Q}^r_\Phi$ *is in* $\mathsf{NLOGSPACE}$*.*

Proof. Consider again the procedure $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ for local model-checking introduced in the proof of Theorem 4. When Φ is in $\mathsf{PHFL}(0, \mathsf{tail})$, all variables are recursive, so the parameters ρ and π are useless, and for every recursive variable X, the counter $c(X)$ remains smaller than $|Q|^d$, so c can be represented in logarithmic space. Since $\boldsymbol{q}$ can also be represented in logarithmic space, and the height of the stack of recursive calls is bounded by the constant $\mathsf{rd}(\Phi)$, **check** is a non-deterministic logarithmic space procedure. □

In order to capture $\mathsf{NLOGSPACE}/_{\sim}$, we consider the logic $\mathsf{FO}[\mathsf{TC}]$ whose syntax is defined as follows

$$\Phi ::= p(x_i) \mid a(x_i, x_j) \mid x_i \prec x_j \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists x_i.\Phi \mid X(\bar{x}) \mid [\mathsf{TC}\ \Phi](\boldsymbol{x}, \boldsymbol{y}).$$

A formula Φ of $\mathsf{FO}[\mathsf{TC}]$ is d-adic if for every first-order variable x_i occurring in Φ the index i is smaller than $2d$. The semantics of a d-adic formula is then a binary relation $R \subseteq Q^{2d}$, with $[\![\mathsf{TC}\ \Phi]\!]$ being the reflexive transitive closure of $[\![\Phi]\!]$. As before, let $\mathsf{FO}[\mathsf{TC}]/_{\sim}$ be the syntactically same logic which is interpreted over reduced LTS only. Bisimulation is still definable in $\mathsf{FO}[\mathsf{TC}]$ because of the preorder $\prec$. Since bisimulation $\sim$ is P-complete [21], $\mathsf{FO}[\mathsf{TC}]/_{\sim}$ is very unlikely to capture $\mathsf{NLOGSPACE}/_{\sim}$ over all transition systems. The way to go around this problem is to assume that the preorder $\prec$ is given as part of the model.

We call *totally ordered LTS* a tuple $\mathfrak{M} = (Q, \prec, \Sigma, P, \Delta, v)$ where $\prec$ is a preoder over Q such that $\prec \cup \succ = \not\sim$. Observe that for a totally ordered LTS $\mathfrak{M}$, $(\mathfrak{M}, \boldsymbol{q})$ is reduced if and only if $\prec$ is a total order, and all states are reachable from one of the $\boldsymbol{q}$ root states, so the query containing all reduced $(\mathfrak{M}, \boldsymbol{q})$ is in $\mathsf{NLOGSPACE}$ over totally ordered LTS. Now, using the same arguments as in Lemma 1, we get that $\mathsf{FO}[\mathsf{TC}]/_{\sim}$ captures $\mathsf{NLOGSPACE}/_{\sim}$ over totally ordered LTS.

We call $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ the set of formulas $\Phi[\Phi_\prec/X]$ such that Φ is in $\mathsf{PHFL}(0, \mathsf{tail})$. It follows from Theorem 6 that all queries over totally ordered LTS that are definable in $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ are in $\mathsf{NLOGSPACE}/_{\sim}$.

Theorem 7. $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ *captures* $\mathsf{NLOGSPACE}/_{\sim}$ *over totally ordered LTS.*

Proof. We need to show that for every $r > 0$ and every formula Φ of $\mathsf{FO}[\mathsf{TC}]$, there is a $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ formula Ψ such that $\mathcal{Q}^r_\Phi = \mathcal{Q}^r_\Psi$ over totally ordered LTS. For any FO formula Φ and its already defined translation $\mathsf{tr}(\Phi)$ into a $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ formula, consider the formulas

$$\Psi_0 := \bigwedge_{i=1}^{d} \{(1,2) \leftarrow (i, i+d)\}\neg\Phi_\prec \wedge \{(2,1) \leftarrow (i, i+d)\}\Phi_\prec$$
$$\Psi := \{s\}\big(\mu X.\Psi_0 \vee \exists_{2d+1} \ldots \exists_{3d}\big((\{s_1\}\mathsf{tr}(\Phi)) \wedge \{s_2\}X\big)\big)$$

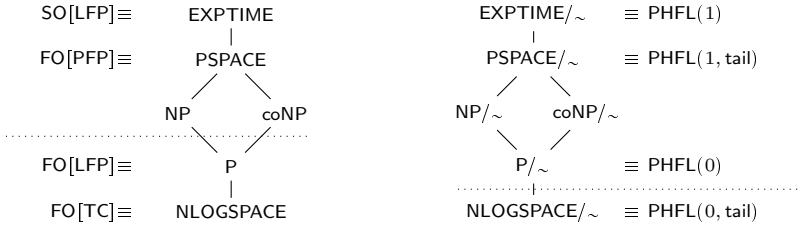


Fig. 4. Capturing bisimulation-invariant complexity classes, compared to their non-bisimulation-invariant counterparts. Below dotted lines, structures are assumed to be equipped with a total order (resp. a bisimulation-invariant preorder).

where $s := \{1, \ldots, 2d{\leftarrow}i_1, \ldots, i_d, j_1, \ldots, j_d\}$, $\{s_1\} := \{1, \ldots, d{\leftarrow}2d+1, 3d\}$ and $\{s_2\} := \{d+1, \ldots, 2d{\leftarrow}2d+1, 3d\}$. Then Ψ is equivalent to $[\mathsf{TC}\ \Phi](\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{x} = x_{i_1}, \ldots, x_{i_d}$ and $\boldsymbol{y} = x_{j_1}, \ldots, x_{j_d}$. Moreover, as $\mathsf{tr}(\Phi)$ is closed and tail-recursive, so is Ψ. So any formula of $\mathsf{FO}[\mathsf{TC}]$ with a single application of the transitive closure has an equivalent in $\mathsf{PHFL}(0, \prec, \mathsf{tail})$; since $\mathsf{FO}[\mathsf{TC}]$ formulas have a normal form with a single application of the transitive closure [10], this proves the result. □

6 Conclusion and Further Work

The results obtained here are presented in Fig. 4 and compared to those results in the descriptive complexity of the non-bisimulation-invariant world which are being used to obtain the results of this paper.

Besides the obvious question of characterisations for $\mathsf{NP}/_\sim$ and $\mathsf{coNP}/_\sim$, bisimulation-invariant complexity classes beyond $\mathsf{EXPTIME}$ also remain to be captured. We believe that $k{-}\mathsf{EXPTIME}/_\sim$ is captured by $\mathsf{PHFL}(k)$ for $k > 1$ and $k{-}\mathsf{EXPSPACE}/_\sim$ is captured by $\mathsf{PHFL}(k-1, \mathsf{tail})$ for $k > 2$. In order to establish such results, we may want to look for other logical characterisations of these complexity classes and encode them in $\mathsf{PHFL}(\omega)$ as we did here. Although we can think about natural candidates for such logical characterisations we could not find suitable references in the literature and the generalisation of our results to higher orders and higher complexity classes is therefore deferred to future work.

References

1. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 512–528. Springer, Heidelberg (2004)
2. Kozen, D.: Results on the propositional μ-calculus. TCS 27, 333–354 (1983)
3. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. Logical Methods in Computer Science 3, 1–33 (2007)

4. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. Complexity and Computation 7, 43–73 (1974)
5. Stockmeyer, L.J.: The polynomial-time hierarchy. TCS 3(1), 1–22 (1976)
6. Immerman, N.: Languages that capture complexity classes. SIAM Journal of Computing 16(4), 760–778 (1987)
7. Abiteboul, S., Vianu, V.: Computing with first-order logic. Journal of Computer and System Sciences 50(2), 309–335 (1995)
8. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: STOC, pp. 137–146. ACM (1982)
9. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68(1-3), 86–104 (1986)
10. Immerman, N.: Descriptive Complexity. Springer, New York (1999)
11. Grädel, E., Kolaitis, P.G., Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Weinstein, S.: Finite Model Theory and its Applications. Springer (2007)
12. Otto, M.: Bisimulation-invariant PTIME and higher-dimensional μ-calculus. Theor. Comput. Sci. 224(1-2), 237–265 (1999)
13. Andersen, H.R.: A polyadic modal μ-calculus. Technical Report 145, IT Universitetet i København (1994)
14. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
15. Stirling, C.: Modal and Temporal Properties of Processes. Texts in Computer Science. Springer (2001)
16. Grädel, E.: Capturing complexity classes by fragments of second-order logic. Theor. Comput. Sci. 101(1), 35–57 (1992)
17. Kobayashi, N., Inaba, K., Tsukada, T.: Unsafe order-2 tree languages are context-sensitive. In: Muscholl, A. (ed.) FOSSACS 2014 (ETAPS). LNCS, vol. 8412, pp. 149–163. Springer, Heidelberg (2014)
18. Lange, M., Lozes, E.: Capturing bisimulation invariant complexity classes with higher-order modal fixpoint logic, `http://carrick.fmv.informatik.uni-kassel.de/~lozes/tcs14-long.pdf`
19. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time. In: STOC, pp. 1–9. ACM (1973)
20. Tarski, A.: A lattice-theoretical fixpoint theorem and its application. Pacific Journal of Mathematics 5, 285–309 (1955)
21. Àlvarez, C., Balcázar, J.L., Gabarró, J., Santha, M.: Parallel complexity in the design and analysis on concurrent systems. In: Aarts, E.H.L., Rem, M., van Leeuwen, J. (eds.) PARLE 1991. LNCS, vol. 505, pp. 288–303. Springer, Heidelberg (1991)

Sensitivity, Block Sensitivity, and Certificate Complexity of Unate Functions and Read-Once Functions

Hiroki Morizumi

Interdisciplinary Graduate School of Science and Engineering, Shimane University, Shimane 690-8504, Japan
morizumi@cis.shimane-u.ac.jp

Abstract. Sensitivity, block sensitivity, and certificate complexity are complexity measures for Boolean functions. In this paper, we prove that these three complexity measures are equal to each other if a Boolean function is a unate function or a read-once function. We also prove $\sqrt{n}$ tight lower bounds for the three complexity measures of read-once functions. As an application of our results, the decision tree complexity of unate functions and read-once functions is upper bounded by the square of the sensitivity of the function.

1 Introduction

Sensitivity, block sensitivity, and certificate complexity of a Boolean function f, denoted by $s(f)$, $bs(f)$ and $C(f)$, respectively, are complexity measures for Boolean functions, and related to other complexity measures including the time complexity of CREW PRAMs and decision tree complexity. A long-standing open problem for these measures is whether or not block sensitivity can be polynomially upper bounded by sensitivity:

$$bs(f) \leq \text{poly}(s(f))?$$

Although many efforts have been devoted to the open problem as we see later, it is still open. On the other hand, if a function f is a monotone function, it is known that $s(f) = bs(f) = C(f)$ [8]. Our main motivation of this paper is to seek other Boolean function classes such that $s(f) = bs(f) = C(f)$.

In this paper, we prove that $s(f) = bs(f) = C(f)$ for unate functions, which are generalized functions of monotone functions, and for read-once functions over the Boolean operators $\wedge$, $\vee$ and $\oplus$. We also prove that $\sqrt{n} \leq s(f)$ $(= bs(f) = C(f))$ for read-once functions which have n input variables, and the lower bound is tight.

Related Works

Rubinstein [9] exhibited a Boolean function f which has $bs(f) = \frac{1}{2}s(f)^2$. The result has been improved [10,2], although the best known gap is still quadratic. Kenyon and Kutin [7] have proved that $bs(f) \leq \frac{e}{\sqrt{2\pi}} e^{s(f)} \sqrt{s(f)}$. The upper

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 104–110, 2014.

bound has been improved to $bs(f) \leq 2^{s(f)-1}s(f)$ by Ambainis et al. [1]. Survey papers [4,5] include more background for this topic. On the average version of the sensitivity, Impagliazzo and Kabanets [6] have given the tight bound on the average sensitivity of read-once de Morgan formulas.

2 Preliminaries

2.1 Sensitivity, Block Sensitivity, and Certificate Complexity

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For an input $x = (x_1, x_2, \ldots, x_n)$ of f and $S \subseteq [n] = \{1, 2, \ldots, n\}$, let x^S denotes the input obtained from x by flipping all the bits x_i such that $i \in S$. We abbreviate $x^{\{i\}}$ to x^i. Sensitivity, block sensitivity, and certificate complexity are defined as follows, respectively.

Definition 1. *The* sensitivity *of f on x, denoted by $s(f,x)$, is the number of indices i such that $f(x) \neq f(x^i)$. The* sensitivity *of f, denoted by $s(f)$, is* $\max_x s(f,x)$. *For $z \in \{0,1\}$, the* z-sensitivity *of f, denoted by $s_z(f)$, is* $\max_{x \in f^{-1}(z)} s(f,x)$.

Definition 2. *The* block sensitivity *of f on x, denoted by $bs(f,x)$, is the maximum number of disjoint subsets $B_1, B_2, \ldots, B_b$ of $[n]$ such that $f(x) \neq f(x^{B_i})$ for all i. The* block sensitivity *of f, denoted by $bs(f)$, is* $\max_x bs(f,x)$. *For $z \in \{0,1\}$, the* z-block sensitivity *of f, denoted by $bs_z(f)$, is* $\max_{x \in f^{-1}(z)} bs(f,x)$.

Definition 3. *A* certificate *of f on x is a subset $S \subseteq [n]$ such that $f(y) = f(x)$ whenever $y_i = x_i$ for all $i \in S$. The* size *of a certificate is $|S|$.*

The certificate complexity *of f on x, denoted by $C(f,x)$, is the size of a smallest certificate of f on x. The* certificate complexity *of f, denoted by $C(f)$, is* $\max_x C(f,x)$. *For $z \in \{0,1\}$, the* z-certificate complexity *of f, denoted by $C_z(f)$, is* $\max_{x \in f^{-1}(z)} C(f,x)$.

We can easily show the following relation between $s(f)$, $bs(f)$ and $C(f)$.

Proposition 1. *For any Boolean function f,*

$$s(f) \leq bs(f) \leq C(f).$$

Proof. By the definitions of $s(f)$ and $bs(f)$, $s(f) \leq bs(f)$. For all x, since a certificate on x have to contain indices of at least one variable of each sensitive block, $bs(f,x) \leq C(f,x)$. Thus, $bs(f) \leq C(f)$. □

Let $x_i, y_i \in \{0,1\}$ for $1 \leq i \leq n$. A Boolean function is called *monotone* if $f(x_1, x_2, \ldots, x_n) \leq f(y_1, y_2, \ldots, y_n)$ whenever $x_i \leq y_i$ for all $1 \leq i \leq n$. Nisan [8] showed the following proposition for monotone functions.

Proposition 2 ([8]). *If f is a monotone function, then*

$$s(f) = bs(f) = C(f).$$

2.2 Unate Functions and Read-Once Functions

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *positive unate* in x_i, $1 \leq i \leq n$, if

$$f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \leq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

for all x_j, $j \neq i$, and is *negative unate* in x_i if

$$f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \geq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

for all x_j, $j \neq i$. A function f is called *unate* if f is positive or negative unate in all x_i for $1 \leq i \leq n$. Monotone functions are a special case of unate functions such that a function is positive unate in all input variables.

A *Boolean formula* is a rooted binary tree in which each internal node is labeled by the Boolean operators $\wedge$, $\vee$, or $\oplus$ and each leaf is labeled by a Boolean variable or its negation. A Boolean formula computes a Boolean function in a natural way. A Boolean formula is called *read-once* if every variable appears exactly once. A *read-once* Boolean function is a Boolean function that can be represented by a read-once Boolean formula. Notice that we define read-once Boolean functions based on Boolean formulas which have the Boolean operator $\oplus$.

3 Unate Functions

In this section, we prove the following theorem.

Theorem 1. *If f is a unate function, then*

$$s(f) = bs(f) = C(f).$$

$s(f)$, $bs(f)$ and $C(f)$ of a Boolean function f are not changed even if some input variables of f are flipped. More precisely, the following lemma holds.

Lemma 1. *Let $f(x)$ be a Boolean function, and let $S \subseteq [n]$. For any S, if $g(y)$ is defined as $f(y^S)$, then,*

$$s(f) = s(g),\ bs(f) = bs(g),\ C(f) = C(g).$$

Proof. It is obvious by the definitions of $s(f)$, $bs(f)$ and $C(f)$. □

Proof (of Theorem 1). Let $S = \{i | f \text{ is negative unate in } x_i, 1 \leq i \leq n\}$. We define $g(y)$ as $f(y^S)$, then $g(y)$ is monotone. By Lemma 1,

$$s(f) = s(g),\ bs(f) = bs(g),\ C(f) = C(g).$$

By Proposition 2,

$$s(g) = bs(g) = C(g).$$

Hence,

$$s(f) = bs(f) = C(f).$$

□

4 Read-Once Functions

In this section, we prove that $s(f) = bs(f) = C(f)$ for any read-once Boolean function (Theorem 2), and prove that $\sqrt{n} \leq s(f)$ (Corollary 1) and the $\sqrt{n}$ lower bound is tight.

4.1 Lemma

Consider a read-once Boolean formula F representing a read-once Boolean function. In F, two subformulas which are connected to a same node have no common input variables, since every variable appears exactly once in a read-once Boolean formula. This fact enables us to analyze the sensitivity and certificate complexity of functions computed at each node in F.

Lemma 2. *Let f_1 and f_2 be Boolean functions such that f_1 and f_2 have no common input variables, and f_1 and f_2 are not constant functions.*

If $f = f_1 \wedge f_2$, then

$$\begin{aligned} s_0(f) &= \max\{s_0(f_1), s_0(f_2)\}, \\ C_0(f) &= \max\{C_0(f_1), C_0(f_2)\}, \\ s_1(f) &= s_1(f_1) + s_1(f_2), \\ C_1(f) &= C_1(f_1) + C_1(f_2). \end{aligned}$$

If $f = f_1 \vee f_2$, then

$$\begin{aligned} s_0(f) &= s_0(f_1) + s_0(f_2), \\ C_0(f) &= C_0(f_1) + C_0(f_2), \\ s_1(f) &= \max\{s_1(f_1), s_1(f_2)\}, \\ C_1(f) &= \max\{C_1(f_1), C_1(f_2)\}. \end{aligned}$$

If $f = f_1 \oplus f_2$, then

$$\begin{aligned} s_0(f) &= \max\{s_0(f_1) + s_0(f_2), s_1(f_1) + s_1(f_2)\}, \\ C_0(f) &= \max\{C_0(f_1) + C_0(f_2), C_1(f_1) + C_1(f_2)\}, \\ s_1(f) &= \max\{s_0(f_1) + s_1(f_2), s_1(f_1) + s_0(f_2)\}, \\ C_1(f) &= \max\{C_0(f_1) + C_1(f_2), C_1(f_1) + C_0(f_2)\}. \end{aligned}$$

Proof. Assume that $f = f_1 \wedge f_2$. We consider that $s_0(f) = \max\{s_0(f_1), s_0(f_2)\}$. If $s_0(f_1) \geq s_0(f_2)$, we can assign input variables of f_2 so that $f_2 = 1$, and independently we can assign input variables of f_1. Thus, we can confirm that $s_0(f) = \max\{s_0(f_1), s_0(f_2)\}$.

Similarly, we can confirm all equations by the definitions of sensitivity and certificate complexity. □

4.2 Equality

Lemma 2 immediately gives the following lemma.

Lemma 3. *Let f_1 and f_2 be Boolean functions such that f_1 and f_2 have no common input variables, and f_1 and f_2 are not constant functions. If*

$$f = f_1 \wedge f_2,\ f = f_1 \vee f_2,\ \text{or } f = f_1 \oplus f_2,$$

and

$$s_0(f_1) = C_0(f_1),\ s_1(f_1) = C_1(f_1),$$
$$s_0(f_2) = C_0(f_2),\ s_1(f_2) = C_1(f_2),$$

then

$$s_0(f) = C_0(f),\ s_1(f) = C_1(f).$$

Now, we prove the following theorem.

Theorem 2. *If f is a read-once Boolean function, then*

$$s(f) = bs(f) = C(f).$$

Proof. Since $s(f) \leq bs(f) \leq C(f)$ for any Boolean function f by Proposition 1, we only need to prove $s(f) = C(f)$.

Let n be the number of input variables of f. We use induction on n and prove $s_0(f) = C_0(f)$ and $s_1(f) = C_1(f)$.

Base: $n = 1$. Then, $f = x_1$ or $f = \neg x_1$, and $s_0(f) = s_1(f) = 1$ and $C_0(f) = C_1(f) = 1$. Thus, $s_0(f) = C_0(f)$ and $s_1(f) = C_1(f)$.

Induction Step: Suppose $s_0(f') = C_0(f')$ and $s_1(f') = C_1(f')$ for every Boolean function f' such that the number of input variables of f' is less than n.

Let F be a read-once Boolean formula which computes f. Recall that we define Boolean formulas as rooted binary trees. Let f_1 and f_2 are Boolean functions computed by subformulas which are connected to the root node of F. Then, $f = f_1 \wedge f_2$, $f = f_1 \vee f_2$, or $f = f_1 \oplus f_2$, and the number of input variables of f_1 and f_2 is less than n, respectively. By the supposition, $s_0(f_1) = C_0(f_1)$, $s_1(f_1) = C_1(f_1)$, $s_0(f_2) = C_0(f_2)$ and $s_1(f_2) = C_1(f_2)$. Thus, by Lemma 3, $s_0(f) = C_0(f)$ and $s_1(f) = C_1(f)$, which mean $s(f) = C(f)$. □

4.3 Lower Bound

Lemma 2 also gives a lower bound for the sensitivity of read-once functions.

Theorem 3. *If f is a read-once Boolean function of n input variables, then*

$$n \leq s_0(f)s_1(f).$$

Proof. We use induction on n.

Base: $n = 1$. Then, $f = x_1$ or $f = \neg x_1$, and $s_0(f)s_1(f) = 1$. Thus, $n \leq s_0(f)s_1(f)$.

Induction Step: Suppose $n' \leq s_0(f')s_1(f')$ for every Boolean function f' such that the number of input variables of f', denoted by n', is less than n.

Let F be a read-once Boolean formula which computes f. Recall that we define Boolean formulas as rooted binary trees. Let f_1 and f_2 are Boolean functions computed by subformulas which are connected to the root node of F, and let n_1 and n_2 are the number of input variables of f_1 and f_2, respectively. Then, $f = f_1 \wedge f_2$, $f = f_1 \vee f_2$, or $f = f_1 \oplus f_2$, and $n_1 < n$, $n_2 < n$, and $n_1 + n_2 = n$. By the supposition, $n_1 \leq s_0(f_1)s_1(f_1)$ and $n_2 \leq s_0(f_2)s_1(f_2)$.

If $f = f_1 \wedge f_2$, then, by Lemma 2,

$$\begin{aligned} s_0(f)s_1(f) &= \max\{s_0(f_1), s_0(f_2)\}s_1(f_1) + \max\{s_0(f_1), s_0(f_2)\}s_1(f_2) \\ &\geq s_0(f_1)s_1(f_1) + s_0(f_2)s_1(f_2) \\ &\geq n_1 + n_2 = n. \end{aligned}$$

Similarly, we can prove that $n \leq s_0(f)s_1(f)$ also for the cases that $f = f_1 \vee f_2$ and $f = f_1 \oplus f_2$. □

Recall that $s(f) = \max\{s_0(f), s_1(f)\}$.

Corollary 1. *If f is a read-once Boolean function of n input variables, then*

$$\sqrt{n} \leq s(f).$$

The lower bounds in Theorem 3 and Corollary 1 are tight, since we can easily confirm that the following read-once Boolean function f has $s_0(f) = n/m$ and $s_1(f) = m$. (We assume that m is a positive integer such that n/m becomes an integer.)

$$f = \bigvee_{i=1}^{n/m} \bigwedge_{j=1}^{m} x_{m(i-1)+j}.$$

5 Concluding Remarks

In this paper, we investigated the sensitivity, block sensitivity, and certificate complexity of unate functions and read-once functions. As the conclusion of this paper, we show an application of our results to decision tree complexity.

Let $D(f)$ denote the decision tree complexity of f, i.e., the depth of an optimal decision tree that computes f. Beals et al. [3] prove

Theorem 4 ([3]). *For any Boolean function f,*

$$D(f) \leq C_1(f)bs(f).$$

Recall that we proved that $s(f) = bs(f) = C(f)$ for any unate function f (Theorem 1) and for any read-once function f (Theorem 2), and $C_1(f) \leq C(f)$ by the definition. Thus, we obtain the following corollary.

Corollary 2. *If f is a unate function or a read-once function, then*

$$D(f) \leq s(f)^2.$$

Although Corollary 2 is meaningful for unate functions, we have to be attentive for read-once functions, since we can easily see that $D(f) = n$ for every read-once function. Thus, Corollary 2 is an alternating proof of Corollary 1 rather than an upper bound of $D(f)$. Notice that the alternating proof depends on Theorem 4 and cannot prove Theorem 3.

References

1. Ambainis, A., Gao, Y., Mao, J., Sun, X., Zuo, S.: New upper bound on block sensitivity and certificate complexity in terms of sensitivity. CoRR abs/1306.4466 (2013)
2. Ambainis, A., Sun, X.: New separation between $s(f)$ and $bs(f)$. Electronic Colloquium on Computational Complexity (ECCC) 18, 116 (2011)
3. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. ACM 48(4), 778–797 (2001)
4. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theor. Comput. Sci. 288(1), 21–43 (2002)
5. Hatami, P., Kulkarni, R., Pankratov, D.: Variations on the sensitivity conjecture. Theory of Computing, Graduate Surveys 2, 1–27 (2011)
6. Impagliazzo, R., Kabanets, V.: Fourier concentration from shrinkage. Electronic Colloquium on Computational Complexity (ECCC) 20, 163 (2013)
7. Kenyon, C., Kutin, S.: Sensitivity, block sensitivity, and l-block sensitivity of boolean functions. Inf. Comput. 189(1), 43–53 (2004)
8. Nisan, N.: CREW PRAMs and decision trees. SIAM J. Comput. 20(6), 999–1007 (1991)
9. Rubinstein, D.: Sensitivity vs. block sensitivity of boolean functions. Combinatorica 15(2), 297–299 (1995)
10. Virza, M.: Sensitivity versus block sensitivity of boolean functions. Inf. Process. Lett. 111(9), 433–435 (2011)

Subshifts, MSO Logic, and Collapsing Hierarchies

Ilkka Törmä

TUCS – Turku Center for Computer Science
University of Turku, Finland
iatorm@utu.fi

Abstract. We use monadic second-order logic to define two-dimensional subshifts, or sets of colorings of the infinite plane. We present a natural family of quantifier alternation hierarchies, and show that they all collapse to the third level. In particular, this solves an open problem of [Jeandel & Theyssier 2013]. The results are in stark contrast with picture languages, where such hierarchies are usually infinite.

Keywords: subshift, MSO logic, quantifier alternation.

1 Introduction

A two-dimensional subshift is a set of colorings of the infinite plane with finitely many colors. Concrete examples are given by sets of *Wang tiles*, or squares with colored edges, introduced by Wang in [13]. The associated *tiling system* consists of all tilings of the plane where overlapping edges have the same color. The initial motivation for Wang tiles was to use a possible algorithm for the infinite tiling problem to recognize tautologies in first-order logic. The tiling problem was proved undecidable by Berger [2], and more undecidability results for tiling systems followed. More recently, strong connections between multidimensional subshifts and computability theory have been found. For example, it was shown in [3], [1] that every vertically constant co-RE subshift can be implemented as a letter-to-letter projection of a tiling system. The topological entropies of tiling systems were characterized in [4] as the right recursively enumerable nonnegative reals. It seems that every conceivable behavior occurs in the class of (projections of) tiling systems, if there is no obvious geometric or computational obstruction.

In this article, we follow the approach of [5,6] and define two-dimensional subshifts by monadic second-order (MSO) logical formulas. We show that certain hierarchies obtained by counting quantifier alternations are finite, solving an open problem posed in [6]. Classes of finite structures defined by MSO formulas have been studied extensively. Examples include finite words, trees, grids and graphs; see [8] and references therein. For words and trees, MSO formulas define exactly the regular languages, and the quantifier alternation hierarchy collapses to the second level. On the other hand, the analogous hierarchy of picture languages was shown to be infinite in [9] and strict in [11]. Although subshifts

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 111–122, 2014.

behave more like sets of words or trees than picture languages in this sense, the reasons are different: MSO-definable languages are regular because the geometry is so simple, while the subshift hierarchy collapses since we can simulate arbitrary computation already on the third level. The concept of constructing subshifts by quantifying over infinite configurations has also been studied in [7] under the name of *multi-choice shift spaces*, and in [12] under the more general framework of *quantifier extensions*. Both formalisms are subsumed by MSO logic.

2 Preliminary Definitions

2.1 Patterns and Subshifts

Fix a finite *alphabet* A. A *pattern* is a map $P : D \to A$ from an arbitrary *domain* $D = D(P) \subset \mathbb{Z}^2$ to A. A pattern with domain $\mathbb{Z}^2$ is a *configuration*, and the set $A^{\mathbb{Z}^2}$ of all configurations is the *full shift over* A. The set of finite patterns over A is denoted by A^{**}, and those with domain $D \subset \mathbb{Z}^2$ by A^D. The restriction of a pattern P to a smaller domain $E \subset D(P)$ is denoted $P|_E$. A pattern P *occurs at* $\boldsymbol{v} \in \mathbb{Z}^2$ in another pattern Q, if we have $\boldsymbol{v} + \boldsymbol{w} \in D(Q)$ and $Q_{\boldsymbol{v}+\boldsymbol{w}} = P_{\boldsymbol{w}}$ for all $\boldsymbol{w} \in D(P)$. We denote $P \sqsubset Q$ if P occurs in Q at some coordinate. For a set of patterns X, we denote $P \sqsubset \mathsf{X}$ if P occurs in some element of X.

A set of finite patterns $F \subset A^{**}$ defines a *subshift* as the set of configurations $\mathsf{X}_F = \{x \in A^{\mathbb{Z}^2} \mid \forall P \in F : P \not\sqsubset x\}$ where no pattern of F occurs. If F is finite, then X_F is *of finite type*, or SFT. The *language* of a subshift $\mathsf{X} \subset A^{\mathbb{Z}^2}$ is $\mathcal{B}(\mathsf{X}) = \{P \in A^{**} \mid P \sqsubset \mathsf{X}\}$. For a finite $D \subset \mathbb{Z}^2$, we denote $\mathcal{B}_D(\mathsf{X}) = \mathcal{B}(\mathsf{X}) \cap A^D$. For $\boldsymbol{v} \in \mathbb{Z}^2$, we denote by $\sigma^{\boldsymbol{v}} : A^{\mathbb{Z}^2} \to A^{\mathbb{Z}^2}$ the *shift by* $\boldsymbol{v}$, defined by $\sigma^{\boldsymbol{v}}(x)_{\boldsymbol{w}} = x_{\boldsymbol{w}+\boldsymbol{v}}$ for all $x \in A^{\mathbb{Z}^2}$ and $\boldsymbol{w} \in \mathbb{Z}^2$. Subshift are invariant under the shift maps.

A *block map* is a function $f : \mathsf{X} \to \mathsf{Y}$ between two subshifts $\mathsf{X} \subset A^{\mathbb{Z}^2}$ and $\mathsf{Y} \subset B^{\mathbb{Z}^2}$ defined by a finite *neighborhood* $D \subset \mathbb{Z}^2$ and a *local function* $F : \mathcal{B}_D(\mathsf{X}) \to B$ which is applied to every coordinate synchronously: $f(x)_{\boldsymbol{v}} = F(x|_{D+\boldsymbol{v}})$ for all $x \in \mathsf{X}$ and $\boldsymbol{v} \in \mathbb{Z}^2$. The image of an SFT under a block map is a *sofic shift*.

Example 1. Let $A = \{0, 1\}$, and let $F \subset A^{**}$ be the set of patterns where 1 occurs twice. Then $\mathsf{X}_F \subset A^{\mathbb{Z}^2}$ is the set of configurations containing at most one letter 1. This subshift is sometimes called the *sunny side up shift*, and it is sofic.

A famous example of an SFT is the *two-dimensional golden mean shift* on the same alphabet, defined by the forbidden patterns $1\ 1$ and $\begin{smallmatrix}1\\1\end{smallmatrix}$. In its configurations, no two letters 1 can be adjacent, but there are no other restrictions.

2.2 Logical Formulas

We continue the line of research of [5,6], and define subshifts by monadic second-order (MSO) formulas. We now introduce the terminology used in these articles, and then expand upon it. A *structure* is a tuple $\mathfrak{M} = (U, \tau)$, where U is an *underlying set*, and τ a *signature* consisting of functions $f : U^n \to U$ and relations $r \subset U^n$ of different *arities* $n \in \mathbb{N}$. A configuration $x \in A^{\mathbb{Z}^2}$ defines a structure $\mathfrak{M}_x = (\mathbb{Z}^2, \tau_A)$, whose signature τ_A contains the following objects:

- Four unary functions, named North, South, East and West, and called *adjacency functions* in this article. They are interpreted in the structure $\mathfrak{M}_x$ as $\mathsf{North}^{\mathfrak{M}_x}((a,b)) = (a,b+1)$, $\mathsf{East}^{\mathfrak{M}_x}((a,b)) = (a+1,b)$ and so on for $a, b \in \mathbb{Z}$.
- For each symbol $a \in A$, a unary *symbol predicate* P_a. It is interpreted as $P_a^{\mathfrak{M}_x}(\boldsymbol{v})$ for $\boldsymbol{v} \in \mathbb{Z}^2$ being true if and only if $x_{\boldsymbol{v}} = a$.

The MSO formulas that we use are defined with the signature τ_A as follows.

- A *term (of depth $k \in \mathbb{N}$)* is a chain of k nested applications of the adjacency functions to a first-order variable.
- An *atomic formula* is either $t = t'$ or $P(t)$, where t and t' are terms and P is either a symbol predicate or a second-order variable.
- A *formula* is either an atomic formula, or an application of a logical connective ($\wedge, \vee, \neg, \ldots$) or first- or second-order quantification to other formulas.

The *radius* of a formula is the maximal depth of a term in it. First-order variables (usually denoted $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_\ell$) hold elements of $\mathbb{Z}^2$, and second-order variables hold subsets of $\mathbb{Z}^2$. Formulas without second-order variables are *first-order*.

Let ϕ be a closed MSO formula, and let $D \subset \mathbb{Z}^2$. A configuration $x \in A^{\mathbb{Z}^2}$ is a *D-model* for ϕ, denoted $x \models_D \phi$, if ϕ is true in the structure $\mathfrak{M}_x$ when the quantification of the first-order variables in ϕ is restricted to D. If $D = \mathbb{Z}^2$, then we denote $x \models \phi$ and say that x *models* ϕ. We define a set of configurations $\mathsf{X}_\phi = \{x \in A^{\mathbb{Z}^2} \mid x \models \phi\}$, which is always shift-invariant, but may not be a subshift. A subshift is *MSO-definable* if it equals X_ϕ for some MSO formula ϕ.

As we find it more intuitive to quantify over configurations than subsets of $\mathbb{Z}^2$, and we later wish to quantify over the configurations of specific subshifts, we introduce the following definitions.

- The notations $\forall X[\mathsf{X}]$ and $\exists X[\mathsf{X}]$ (read *for all (or exists) X in X*) define a new *configuration variable* X, which represents a configuration of a subshift $\mathsf{X} \subset B^{\mathbb{Z}^2}$ over a new alphabet B.
- For $X[\mathsf{X}]$ quantified as above, $b \in B$ and a term t, the notation $X_t = b$ defines an atomic formula that is true if and only if the configuration represented by X has the letter b at the coordinate represented by t.

MSO formulas with configuration variables instead of ordinary second-order variables are called *extended MSO formulas*, and the relation $\models$ is extended to them. We state without proof that if the subshifts occurring in an extended MSO formula ϕ are MSO-definable, then so is X_ϕ. Conversely, we can convert an MSO formula to an extended MSO formula by replacing every second-order variable with a configuration variable over the binary full shift. Unless stated otherwise, by second-order variables (usually denoted $X_1, \ldots, X_n$) we mean configuration variables, and by MSO formulas we mean extended MSO formulas.

Example 2. The two-dimensional golden mean shift is defined by the formula

$$\forall \boldsymbol{n}\big(P_1(\boldsymbol{n}) \implies \big(P_0(\mathsf{North}(\boldsymbol{n})) \wedge P_0(\mathsf{East}(\boldsymbol{n}))\big)\big).$$

Also, the sunny side up shift is defined by the formula

$$\forall \boldsymbol{m} \forall \boldsymbol{n} \big(P_1(\boldsymbol{n}) \implies (P_0(\boldsymbol{m}) \vee \boldsymbol{m} = \boldsymbol{n}) \big).$$

Another way to define the sunny side up shift is to use a second-order quantifier:

$$\begin{aligned} \exists U \forall \boldsymbol{n} \big(& U(\boldsymbol{n}) \iff \big(U(\mathsf{North}(\boldsymbol{n})) \wedge U(\mathsf{West}(\boldsymbol{n})) \big) \big) \\ & \wedge \big(P_1(\boldsymbol{n}) \implies \big(U(\boldsymbol{n}) \wedge \neg U(\mathsf{South}(\boldsymbol{n})) \wedge \neg U(\mathsf{East}(\boldsymbol{n})) \big) \big). \end{aligned}$$

We can produce an equivalent extended MSO formula, as per the above remark:

$$\begin{aligned} \exists X[\{0,1\}^{\mathbb{Z}^2}] \forall \boldsymbol{n} \big(& X_{\boldsymbol{n}} = 1 \iff (X_{\mathsf{North}(\boldsymbol{n})} = 1 \wedge X_{\mathsf{West}(\boldsymbol{n})} = 1) \big) \\ & \wedge \big(P_1(\boldsymbol{n}) \implies (X_{\boldsymbol{n}} = 1 \wedge X_{\mathsf{South}(\boldsymbol{n})} = 0 \wedge X_{\mathsf{East}(\boldsymbol{n})} = 0) \big). \end{aligned}$$

2.3 Computability Theory

We recall the *arithmetical hierarchy*, a classical reference for which is [10]. A first-order arithmetical formula over $\mathbb{N}$ is Π_0^0 (equivalently, Σ_0^0), if it only contains bounded quantifiers (of the form $\forall n \leq k$ or $\exists n \leq k$). The formula is Π_{k+1}^0 (Σ_{k+1}^0) if it is of the form $\forall n_1 \cdots \forall n_\ell \phi$ ($\exists n_1 \cdots \exists n_\ell \phi$) where ϕ is Σ_k^0 (Π_k^0, respectively). Every such formula is equivalent to a Π_k^0 or Σ_k^0 one, and if it defines a subset of $\mathbb{N}$, that set is given the same classification. Completeness and hardness in the classes are defined using Turing reductions. For all $k \in \mathbb{N}$, the class $\Delta_{k+1}^0 = \Pi_{k+1}^0 \cap \Sigma_{k+1}^0$ contains exactly the languages decidable by Turing machines with Π_k^0 oracles. Also, Σ_1^0 is the class of recursively enumerable subsets of $\mathbb{N}$.

When classifying subsets of countable sets other than $\mathbb{N}$, we assume they are in some natural and computable bijection with $\mathbb{N}$. For example, a co-recursively enumerable set of finite patterns is Π_1^0. A subshift X is given the same classification as its language $\mathcal{B}(\mathsf{X})$. If X is Π_k^0 for some $k \in \mathbb{N}$, then it can be defined by a Σ_k^0 set of forbidden patterns (the complement of $\mathcal{B}(\mathsf{X})$), and a subshift defined by such a set is always Π_{k+1}^0. In particular, SFTs and sofic shifts are Π_1^0.

Remark 1. We use several hierarchies of subshifts obtained by counting quantifier alternations in different kinds of formulas, and the notation for them can be confusing. In general, classes defined by computability conditions (the arithmetical hierarchy) are denoted by Π and Σ, while classes defined by MSO formulas via the modeling relation are denoted by $\bar{\Pi}$ and $\bar{\Sigma}$.

3 Hierarchies of MSO-Definable Subshifts

In this section, we recall the definition of a hierarchy of subshift classes defined in [5,6], and then generalize it. We also state some general lemmas.

Definition 1. *Let C be a class of subshifts. An MSO formula ψ is* over C with universal first-order quantifiers*, or C-u-MSO for short, if it is of the form*

$$\psi = Q_1 X_1[\mathsf{X}_1] Q_2 X_2[\mathsf{X}_2] \cdots Q_n X_n[\mathsf{X}_n] \forall \boldsymbol{n}_1 \cdots \forall \boldsymbol{n}_\ell \phi,$$

where each Q_i is a quantifier, $\mathsf{X}_i \in C$, and ϕ is quantifier-free. If there are k quantifier alternations and Q_1 is the existential quantifier $\exists$, then ψ is called $\bar{\Sigma}_k[C]$, and if Q_1 is $\forall$, then ψ is $\bar{\Pi}_k[C]$. The set X_ψ is given the same classification. If C is the singleton class containing only the binary full shift $\{0,1\}^{\mathbb{Z}^2}$, then ψ is called u-MSO, and we denote $\bar{\Sigma}_k[C] = \bar{\Sigma}_k$ and $\bar{\Pi}_k[C] = \bar{\Pi}_k$. The classes $\bar{\Sigma}_k$ and $\bar{\Pi}_k$ for $k \in \mathbb{N}$ form the u-MSO hierarchy.

In [6], the u-MSO hierarchy was denoted by the letter $\mathcal{C}$, but we use the longer name for clarity. In the rest of this article, C denotes an arbitrary class of subshifts, unless otherwise noted. We proceed with the following result, stated for u-MSO formulas in [6]. We omit the proof, as it is essentially the same.

Theorem 1 (Generalization of Theorem 13 of [6]). *Let ϕ be a C-u-MSO formula over an alphabet A. Then for all $x \in A^{\mathbb{Z}^2}$, we have $x \models \phi$ if and only if $x \models_D \phi$ for every finite domain $D \subset \mathbb{Z}^2$.*

Corollary 1. *Every C-u-MSO formula ϕ over an alphabet A defines a subshift.*

Proof. Let $r \in \mathbb{N}$ be the radius of ϕ. By Theorem 1, we have $\mathsf{X}_\phi = \mathsf{X}_F$, where $F = \{x|_{D+[-r,r]^2} \mid D \subset \mathbb{Z}^2 \text{ finite}, x \in A^{\mathbb{Z}^2}, x \not\models_D \phi\}$. □

Corollary 2. *For all $k, n \in \mathbb{N}$, we have $\bar{\Pi}_n[\Pi^0_k] \subset \Pi^0_{k+1}$. In particular, the u-MSO hierarchy only contains Π^0_1 subshifts.*

Proof. Let $\phi = \forall X_1[\mathsf{X}_1]\exists X_2[\mathsf{X}_2] \dots Q_n X_n[\mathsf{X}_n]\psi$ be a $\bar{\Pi}_n[\Pi^0_k]$ formula, where each $\mathsf{X}_i \subset A_i^{\mathbb{Z}^2}$ is a Π^0_k subshift and ψ is first-order. Then the product subshift $\prod_{i=1}^n \mathsf{X}_i$ is also Π^0_k. Let $P \in A^{**}$ be a finite pattern. Theorem 1, together with a basic compactness argument, implies that $P \in \mathcal{B}(\mathsf{X}_\phi)$ holds if and only if for all finite domains $D(P) \subset D \subset \mathbb{Z}^2$, there exists a configuration $x \in A^{\mathbb{Z}^2}$ such that $x|_{D(P)} = P$ and $x \models_D \phi$. For a fixed D, denote this condition by $C_P(D)$.

We show that deciding $C_P(D)$ for given pattern P and domain D is Δ^0_{k+1}. Denote $E = D+[-r,r]^2$, where $r \in \mathbb{N}$ is the radius of ϕ, and let $L = \mathcal{B}_E(\prod_{i=1}^n \mathsf{X}_i)$. For a configuration $x \in A^{\mathbb{Z}^2}$, the condition $x \models_D \phi$ only depends on the finite pattern $x|_E \in A^E$, and is computable from it and the set L. Thus $C_P(D)$ is equivalent to the existence of a pattern $Q \in A^E$ such that $x|_E = Q$ implies $x \models_D \phi$ for all $x \in A^{\mathbb{Z}^2}$. Moreover, this can be decided by the oracle Turing machine that computes L using a Π^0_k oracle, and then goes through the finite set A^E, searching for such a Q. Thus the condition $C_P(D)$ is Δ^0_{k+1}, which implies that deciding $P \in \mathcal{B}(\mathsf{X}_\phi)$ is Π^0_{k+1}. □

Finally, if the final second-order quantifier of a u-MSO formula is universal, it can be dropped. This does not hold for C-u-MSO formulas in general. We omit the proof, as it is essentially the same as that of [6, Lemma 7].

Lemma 1. *If $k \geq 1$ is odd, then $\bar{\Pi}_k = \bar{\Pi}_{k-1}$, and if it is even, then $\bar{\Sigma}_k = \bar{\Sigma}_{k-1}$.*

Example 3. Define the *mirror shift* $\mathsf{M} \subset \{0,1,\#\}^{\mathbb{Z}^2}$ by the forbidden patterns $\begin{smallmatrix} a \\ \# \end{smallmatrix}$ and $\begin{smallmatrix} \# \\ a \end{smallmatrix}$ for $a \neq \#$, every pattern $\{\mathbf{0} \mapsto \#, (n,0) \mapsto \#\}$, and every pattern

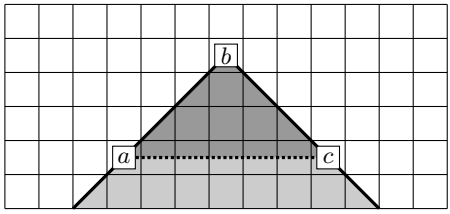


Fig. 1. A pattern of X in Example 3, containing its entire alphabet

$\{(-n, 0) \mapsto a, \mathbf{0} \mapsto \#, (n, 0) \mapsto b\}$ for $n \in \mathbb{N}$ and $a \neq b$. A 'typical' configuration of M contains one infinite column of #-symbols, whose left and right sides are mirror images of each other. It is well-known that M is not sofic. We show that it can be implemented by an SFT-u-MSO formula $\psi = \forall X[\mathsf{X}]\forall \boldsymbol{n}_1 \forall \boldsymbol{n}_2 \forall \boldsymbol{n}_3 \phi$ in the class $\bar{\Pi}_1$[SFT]. This also shows that Lemma 1 fails outside the u-MSO hierarchy.

Let X be the SFT whose alphabet is seen in Figure 1, defined by the obvious 2×2 forbidden patterns. Define the formula ϕ as $\phi_1 \wedge (\phi_2 \implies \phi_3)$, where

$$\begin{aligned}
\phi_1 &= P_{\#}(\boldsymbol{n}_2) \iff P_{\#}(\mathsf{North}(\boldsymbol{n}_2)) \\
\phi_2 &= X_{\boldsymbol{n}_1} = a \wedge X_{\boldsymbol{n}_2} = b \wedge X_{\boldsymbol{n}_3} = c \wedge P_{\#}(\boldsymbol{n}_2) \\
\phi_3 &= \neg P_{\#}(\boldsymbol{n}_1) \wedge \neg P_{\#}(\boldsymbol{n}_3) \wedge (P_0(\boldsymbol{n}_1) \iff P_0(\boldsymbol{n}_3))
\end{aligned}$$

It is easy to see that the subshift X_ψ is exactly M, with ψ defined as above.

4 The u-MSO Hierarchy

The u-MSO hierarchy is a quite natural hierarchy of MSO-definable subshifts. Namely, the lack of existential first-order quantification makes it easy to prove that every u-MSO formula defines a subshift, and quantifier alternations give rise to interesting hierarchies in many contexts. The following is already known.

Theorem 2 ([6]). *The class of subshifts defined by formulas of the form $\forall \boldsymbol{n} \phi$, where ϕ is first-order, is exactly the class of SFTs. The class $\bar{\Pi}_0 = \bar{\Sigma}_0$ consists of the* threshold counting shifts*, which are obtained from subshifts of the form $\{x \in A^{\mathbb{Z}^2} \mid P$ occurs in x at most n times$\}$ for $P \in A^{**}$ and $n \in \mathbb{N}$ using finite unions and intersections. Finally, the class $\bar{\Sigma}_1$ consists of exactly the sofic shifts.*

We show that the hierarchy collapses to the third level, which consists of exactly the Π_1^0 subshifts. This gives negative answers to the questions posed in [6] of whether the hierarchy is infinite, and whether it only contains sofic shifts.

Theorem 3. *For all $n \geq 2$ we have $\Pi_1^0 = \bar{\Pi}_n$.*

Proof. As we have $\bar{\Pi}_n \subset \Pi_1^0$ by Corollary 2, and clearly $\bar{\Pi}_n \subset \bar{\Pi}_{n+1}$ also holds, it suffices to prove $\Pi_1^0 \subset \bar{\Pi}_2$. Let thus $\mathsf{X} \subset A^{\mathbb{Z}^2}$ be a Π_1^0 subshift. We construct an MSO formula of the form $\phi = \forall Y[B^{\mathbb{Z}^2}]\exists Z[C^{\mathbb{Z}^2}]\forall \boldsymbol{n} \psi(\boldsymbol{n}, Y, Z)$ such that $\mathsf{X}_\phi = \mathsf{X}$.

The main idea is the following. We use the universally quantified configuration Y to specify a finite square $R \subset \mathbb{Z}^2$ and a word $w \in A^*$, which may or may not encode the pattern x_R of a configuration $x \in A^{\mathbb{Z}^2}$. The existentially quantified Z enforces that either w does not correctly encode x_R, of that it encodes *some* pattern of $\mathcal{B}(\mathsf{X})$. As R and w are arbitrary and universally quantified, this guarantees $x \in \mathsf{X}$. The main difficulty is that Y comes from a full shift, so we have no control over it; there may be infinitely many squares, or none at all.

First, we define an auxiliary SFT $\mathsf{Y} \subset B^{\mathbb{Z}^2}$, whose configurations contain the aforementioned squares. The alphabet B consists of the tiles seen in Figure 2, where every w_i ranges over A, and it is defined by the set F_{Y} of 2×2 forbidden patterns where some colors or lines of neighboring tiles do not match. A configuration of Y contains at most one maximal pattern colored with the lightest gray in Figure 2, and if it is finite, its domain is a square. We call this domain the *input square*, and the word $w \in A^*$ that lies above it is called the *input word*.

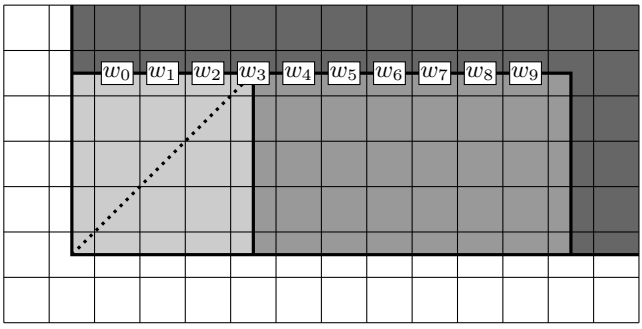


Fig. 2. A pattern of Y. In this example, the input word $w \in A^*$ is of length 10.

We now define another SFT S, this time on the alphabet $A \times B \times C$. The alphabet C is more complex than B, and we specify it in the course of the construction. The idea is to simulate a computation in the third layer to ensure that if the second layer contains a valid configuration of Y and the input word encodes the contents of the input square in the first layer, then that square pattern is in $\mathcal{B}(\mathsf{X})$. We also need to ensure that a valid configuration exists even if the encoding is incorrect, or if second layer is not in Y. For this, every locally valid square pattern of Y containing an input square will be covered by another square pattern in the third layer, inside which we perform the computations. We will force this pattern to be infinite if the second layer is a configuration of Y.

Now, we describe a configuration $(x, y, z) \in \mathsf{S}$. The coordinates of every 2×2 rectangle $R \subset \mathbb{Z}^2$ with $y|_R \in F_{\mathsf{Y}}$ are called *defects*. A non-defect coordinate $\boldsymbol{v} \in \mathbb{Z}^2$ such that $y_{\boldsymbol{v}} =$  is called a *seed*. Denote $C = C_1 \cup C_2$, where C_1 is the set of tiles depicted in Figure 3 (a). Their adjacency rules in S are analogous to those of Y. The rules of S also force the set of seeds to coincide with the coordinates $\boldsymbol{v} \in \mathbb{Z}^2$ such that $z_{\boldsymbol{v}} =$  . These coordinates are the southwest

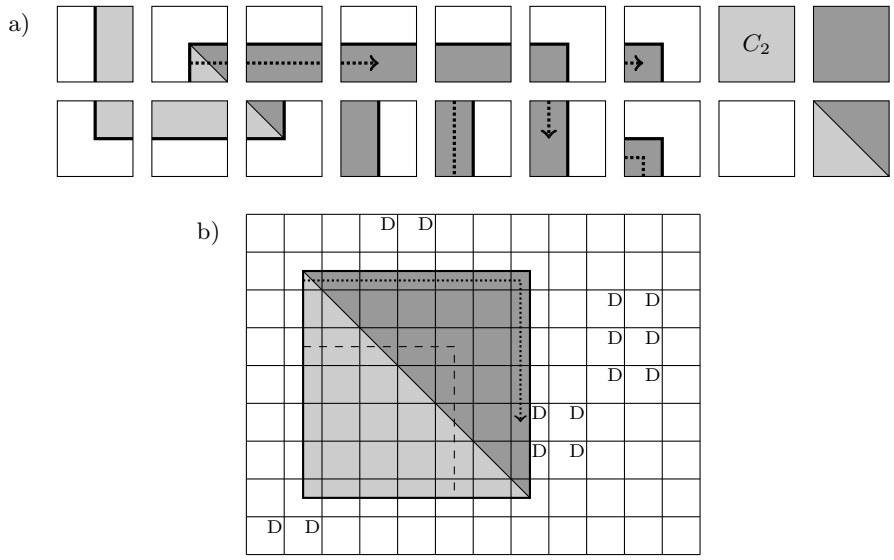


Fig. 3. The alphabet C (a) and a pattern of the third layer of S (b), with the elements of C_2 represented by the featureless light gray tiles. The dashed line represents the border of an input square on the second layer. Defects are marked with a small D.

corners of *computation squares* in z, whose square shape is again enforced by a diagonal signal. The southwest half of a computation square is colored with letters of C_2. See Figure 3 (b) for an example of a computation square.

A computation square may not contain defects or coordinates $\boldsymbol{v} \in \mathbb{Z}^2$ such that $y_{\boldsymbol{v}} = \square$ except on its north or east border, and conversely, one of the borders will contain a defect. This is enforced by a signal emitted from the northwest corner of the square (the dotted line in Figure 3 (b)), which travels along the north and east borders, and disappears when it encounters a defect.

We now describe the set C_2, and for that, let M be a Turing machine with input alphabet $\Sigma = A \times (A \cup \{0, 1, \#\})$ and two initial states q_1 and q_2. This machine is simulated on the southwest halves of the computation squares in a standard way, and we will fix its functionality later. The alphabet C_2 is shown in Figure 4. Note that the colors and lines in C_2 are disjoint from those in C_1, even though the figures suggest otherwise. The idea is to initialize the machine M with either the input word (if it correctly encodes the input square), or a proof that the encoding is incorrect, in the form of one incorrectly encoded symbol.

The white squares and circles of C_2 must be placed on the letters of the input word $w \in A^*$ of the computation square, the square on the leftmost letter and circles on the rest. The A-letters of these tiles must match the letters of w, and the second component is 1 if the tile lies on the corner of the input square, 0 if not, $b \in A$ in the presence of a vertical signal, and # in the presence of a diagonal signal. Such signals are sent by a white diamond tile (called a *candidate error*), which can only be placed on the interior tiles of the input square, and whose

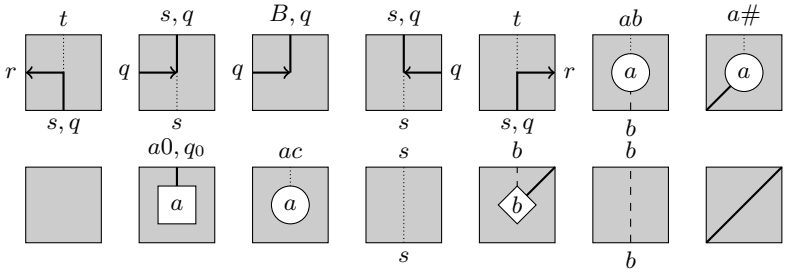


Fig. 4. The sub-alphabet C_2. The letters a and b range over A, c can be 0 or 1, the letter s over the tape alphabet of M, the letter q_0 can be either of the initial states q_1 and q_2, and in the first (fourth) tile on the top row we require that the machine M writes $t \in \Sigma$ on the tape, switches to state r and steps to the left (right, respectively) when reading the letter $s \in \Sigma$ in state q.

letter must match the letter on the first layer x. Other tiles of C_2 simulate the machine M, which can never halt in a valid configuration. See Figure 5 for a visualization. We also require that for a pattern $\begin{smallmatrix} c_2 \\ c_1 \end{smallmatrix}$ to be valid, where $c_i \in C_i$ for $i \in \{1, 2\}$, the tile c_2 should have a gray south border with no lines. Other adjacency rules between tiles of C_1 and C_2 are explained by Figure 3 (a).

We now describe the machine M. Note first that from an input $u \in \Sigma^*$ one can deduce the input word $w \in A^*$, the height $h \in \mathbb{N}$ of the input square, and the positions and contents of all candidate errors. Now, when started in the state q_1, the machine checks that there are no candidate errors at all, that $|w| = h^2$, and that the square pattern $P \in A^{h \times h}$, defined by $P_{(i,j)} = w_{ih+j}$ for all $i, j \in [0, h-1]$, is in $\mathcal{B}(\mathsf{X})$. If all this holds, M runs forever (the check for $P \in \mathcal{B}(\mathsf{X})$ can indeed take infinitely many steps). When started in q_2, the machine checks that there is exactly one candidate error at some position $(i, j) \in [0, h-1]^2$ of the input square containing some letter $b \in A$, and that one of $|w| \neq h^2$ or $w_{ih+j} \neq b$ holds. If this is the case, M enters an infinite loop, and halts otherwise.

The definition of S is now complete, and it can be realized using a set F of forbidden patterns of size 3×3. We define the quantifier-free formula $\psi(\boldsymbol{n}, Y, Z)$ as $\neg \bigvee_{P \in F} \psi_P$, where ψ_P states that the pattern P occurs at the coordinate $\boldsymbol{n}$. This is easily doable using the adjacency functions, color predicates and the variables Y and Z. If we fix some values $y \in B^{\mathbb{Z}^2}$ and $z \in C^{\mathbb{Z}^2}$ for the variables Y and Z, then $x \models \forall \boldsymbol{n} \psi(\boldsymbol{n}, y, z)$ holds for a given $x \in A^{\mathbb{Z}^2}$ if and only if $(x, y, z) \in \mathsf{S}$.

Let $x \in A^{\mathbb{Z}^2}$ be arbitrary. We need to show that $x \models \phi$ holds if and only if $x \in \mathsf{X}$. Suppose first that x models ϕ, and let $\boldsymbol{v} \in \mathbb{Z}^2$ and $h \geq 1$. Let $y \in \mathsf{Y}$ be a configuration whose input square has interior $D = \boldsymbol{v} + [0, h-1]^2$, and whose input word correctly encodes the pattern $x|_D$. By assumption, there exists $z \in C^{\mathbb{Z}^2}$ such that $(x, y, z) \in \mathsf{S}$, so that the southwest neighbor of $\boldsymbol{v}$ is the southwest corner of a computation square in z, which is necessarily infinite, since no defects occur in y. In this square, M runs forever, and it cannot be initialized in the state q_2 as the encoding of the input square is correct. Thus its computation proves that $x|_D \in \mathcal{B}(\mathsf{X})$. Since D was an arbitrary square domain, we have $x \in \mathsf{X}$.

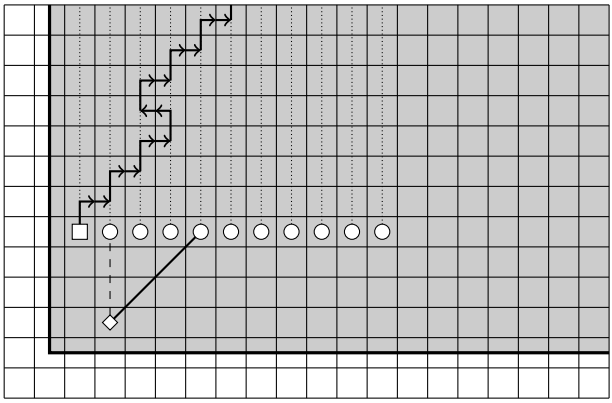


Fig. 5. An infinite computation square with an input word of length 11 and a single candidate error

Suppose then $x \in \mathsf{X}$, and let $y \in B^{\mathbb{Z}^2}$ be arbitrary. We construct a configuration $z \in C^{\mathbb{Z}^2}$ such that $(x, y, z) \in \mathsf{S}$, which proves $x \models \phi$. First, let $S \subset \mathbb{Z}^2$ be the set of seeds in y, and for each $\boldsymbol{s} \in S$, let $\ell(\boldsymbol{s}) \in \mathbb{N} \cup \{\infty\}$ be the height of the maximal square $D(\boldsymbol{s}) = \boldsymbol{s} + [0, \ell(\boldsymbol{s}) - 1]^2$ whose interior contains no defects. We claim that $D(\boldsymbol{s}) \cap D(\boldsymbol{r}) = \emptyset$ holds for all $\boldsymbol{s} \neq \boldsymbol{r} \in S$. Suppose the contrary, and let $\boldsymbol{v} \in D(\boldsymbol{s}) \cap D(\boldsymbol{r})$ be lexicographically minimal. Then $\boldsymbol{v}$ is on the south border of $D(\boldsymbol{s})$ and the west border of $D(\boldsymbol{r})$ (or vice versa). Since these borders contain no defects, $y_{\boldsymbol{v}}$ is a south border tile and a west border tile, a contradiction.

Now, we can define every $D(\boldsymbol{s})$ to be a computation square in z. If it contains an input square and an associated input word which correctly encodes its contents, we initialize the simulated machine M in the state q_1. Then the computation does not halt, since the input square contains a pattern of $\mathcal{B}(\mathsf{X})$. Otherwise, we initialize M in the state q_2, and choose a single candidate error from the input square such that it does not halt, and thus produces no forbidden patterns. Then $(x, y, z) \in \mathsf{S}$, completing the proof. □

We have now characterized every level of the u-MSO hierarchy. The first level $\bar{\Pi}_0 = \bar{\Sigma}_0$ contains the threshold counting shifts and equals $\bar{\Pi}_1$ by Lemma 1, the class $\bar{\Sigma}_1 = \bar{\Sigma}_2$ contains the sofic shifts, and the other levels coincide with Π^0_1.

The quantifier alternation hierarchy of MSO-definable picture languages was shown to be strict in [11]. It is slightly different from the u-MSO hierarchy, as existential first-order quantification is allowed. However, in the case of pictures we know the following. Any MSO formula $\mathcal{Q}_L \exists \boldsymbol{n} \mathcal{Q}_R \phi$, where $\mathcal{Q}_L$ and $\mathcal{Q}_R$ are strings of quantifiers, is equivalent to a formula of the form $\mathcal{Q}_L \exists X \mathcal{Q}_R \forall \boldsymbol{n} \psi$, where ϕ and ψ are quantifier-free. See [8, Section 4.3] for more details. Thus the analogue of the u-MSO hierarchy for picture languages is infinite. The proof of the result of [11] relies on the fact that one can simulate computation within the pictures, and the maximal time complexity depends on the number of alternations. In the case of infinite configurations, this argument naturally falls apart.

Finally, Theorem 3 has the following corollary (which was also proved in [6]).

Corollary 3. *Every Π^0_1 subshift is MSO-definable.*

5 Other C-u-MSO Hierarchies

Next, we generalize Theorem 3 to hierarchies of Π^0_k-u-MSO formulas. The construction is similar to the above but easier, since we can restrict the values of the variable Y to lie in a geometrically well-behaved subshift.

Theorem 4. *For all $k \geq 1$ and $n \geq 2$ we have $\Pi^0_{k+1} = \bar{\Pi}_n[\Pi^0_k]$. Furthermore, $\Pi^0_2 = \bar{\Pi}_n[\mathrm{SFT}]$ for all $n \geq 2$.*

Proof (sketch). As in Theorem 3, it suffices to show that for a given Π^0_{k+1} subshift $\mathsf{X} \subset A^{\mathbb{Z}^2}$, there is a $\bar{\Pi}_2[\Pi^0_k]$ formula $\phi = \forall Y[\mathsf{Y}]\exists Z[\mathsf{Z}]\forall \boldsymbol{n}\psi$ such that $\mathsf{X}_\phi = \mathsf{X}$. In our construction, $\mathsf{Y} \subset B^{\mathbb{Z}^2}$ is a Π^0_k subshift and $\mathsf{Z} = C^{\mathbb{Z}^2}$ is a full shift.

For a square pattern $P \in A^{h \times h}$, define the word $w(P) \in A^{h^2}$ by $w_{ih+j} = P_{(i,j)}$ for all $i, j \in [0, h-1]$. Let $R \subset A^* \times \mathbb{N}$ be a Π^0_k predicate such that the set

$$F = \{P \in A^{h \times h} \mid h \in \mathbb{N}, \exists n \in \mathbb{N} : R(w(P), n)\}$$

satisfies $\mathsf{X}_F = \mathsf{X}$. As in Theorem 3, configurations of Y may contain one input square with an associated input word. This time, the input word is of the form $w\#^n$ for some $w \in A^*$, $n \in \mathbb{N}$ and a new symbol $\#$. As Y is Π^0_k, we can enforce that $R(w, n)$ holds, so that w does *not* encode any square pattern of X. This can be enforced by SFT rules if $k = 1$: a simulated Turing machine checks $R(w, n)$ by running forever if it holds. As before, the existential layer Z enforces that w does *not* correctly encode the contents of the input square in the first layer.

Let $x \in \mathsf{X}$ and $y \in \mathsf{Y}$ be arbitrary. If y has a finite input square $D \in \mathbb{Z}^2$ and input word $w\#^n$, then $w \in A^*$ cannot correctly encode the pattern $x|_D \in \mathcal{B}(\mathsf{X})$, and thus a valid choice for the variable Z exists. Degenerate cases of y (with, say, an infinite input square) are handled as in Theorem 3. Thus we have $x \models \phi$. Next, suppose that $x \notin \mathsf{X}$, so there is a square domain $D \subset \mathbb{Z}^2$ with $x|_D \notin \mathcal{B}(\mathsf{X})$. Construct $y \in \mathsf{Y}$ such that the input square has domain D, the word $w \in A^*$ correctly encodes $x|_D$, and the number $n \in \mathbb{N}$ of $\#$-symbols is such that $R(w, n)$ holds. For this value of Y, no valid choice for Z exists, and thus $x \not\models \phi$. □

Corollary 3, Theorem 4 and a simple induction argument show the following.

Corollary 4. *For every $k \in \mathbb{N}$, every Π^0_k subshift is MSO-definable.*

However, note that the converse does not hold, since one can construct an MSO-formula defining a subshift whose language is not Π^0_k for any $k \in \mathbb{N}$.

Acknowledgments. I am thankful to Emmanuel Jeandel for introducing me to [5,6] and the open problems therein, and to Ville Salo for many fruitful discussions.

References

1. Aubrun, N., Sablik, M.: Simulation of effective subshifts by two-dimensional subshifts of finite type. Acta Appl. Math. 126(1), 35–63 (2013)
2. Berger, R.: The undecidability of the domino problem. Mem. Amer. Math. Soc. No. 66, 72 pages (1966)
3. Durand, B., Romashchenko, A., Shen, A.: Fixed-point tile sets and their applications. J. Comput. System Sci. 78(3), 731–764 (2012)
4. Hochman, M., Meyerovitch, T.: A characterization of the entropies of multidimensional shifts of finite type. Ann. of Math (2) 171(3), 2011–2038 (2010)
5. Jeandel, E., Theyssier, G.: Subshifts, languages and logic. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 288–299. Springer, Heidelberg (2009)
6. Jeandel, E., Theyssier, G.: Subshifts as models for MSO logic. Inform. and Comput. 225, 1–15 (2013)
7. Louidor, E., Marcus, B., Pavlov, R.: Independence entropy of $\mathbb{Z}^d$-shift spaces. Acta Applicandae Mathematicae, 1–21 (2013)
8. Matz, O., Schweikardt, N.: Expressive power of monadic logics on words, trees, pictures, and graphs. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata. Texts in Logic and Games, vol. 2, pp. 531–552. Amsterdam University Press (2008)
9. Matz, O., Thomas, W.: The monadic quantifier alternation hierarchy over graphs is infinite. In: Twelfth Annual IEEE Symposium on Logic in Computer Science, pp. 236–244. IEEE (1997)
10. Sacks, G.E.: Higher recursion theory. Perspectives in mathematical logic. Springer (1990)
11. Schweikardt, N.: The monadic quantifier alternation hierarchy over grids and pictures. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 441–460. Springer, Heidelberg (1998)
12. Törmä, I.: Quantifier Extensions of Multidimensional Sofic Shifts. ArXiv e-prints (January 2014)
13. Wang, H.: Proving theorems by pattern recognition II. Bell System Technical Journal 40, 1–42 (1961)

Fast Nondeterministic Matrix Multiplication via Derandomization of Freivalds' Algorithm*

Jiří Wiedermann

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. We design two nondeterministic algorithms for matrix multiplication. Both algorithms are based on derandomization of Freivalds' algorithm for verification of matrix products. The first algorithm works with real numbers and its time complexity on Real RAMs is $O(n^2 \log n)$. The second one is of the same complexity, works with integer matrices on a unit cost RAM with numbers whose size is proportional to the size of the largest entry in the underlying matrices. Our algorithms bring new ideas into the design of matrix multiplication algorithms and open new avenues for their further development. The results pose exciting questions concerning the relation of the complexity of deterministic versus nondeterministic algorithms for matrix multiplication, and complexity of integer versus real matrices multiplication.

1 Introduction

Matrix multiplication probably is one of the most closely watched areas of algorithm design. A great attention is paid even to minor advancements in this field. Undoubtedly, this is because of the immense practical importance of the problem backed by its deep algorithmic beauty. Matrix multiplication has attracted considerable attention for more than four decades and the challenge is whether or not matrix multiplication can be done in quadratic time. The origin of this conjecture lies back in the late nineteen sixties when Volker Strassen discovered an algorithm for matrix multiplication of complexity $O(n^{2.807})$ [23] (in terms of the number of arithmetical operations). This has been an unexpected and significant improvement over the classical algorithm of complexity $O(n^3)$. Since then the search for matrix multiplication algorithms of complexity $O(n^{2+\omega})$, for $0 \leq \omega < 1$, has started and over the years it has proceeded through several incremental improvements of the ω exponent. The best current upper bound achieves the algorithm by Vasilevska Williams from 2011 with $\omega = 0.3729$.

The increased interest in matrix multiplication algorithms came about in the same time as the increased interest in randomized algorithms. Randomization has become a subject of systematic investigation as a "cheap" tool for solution

* This research was partially supported by RVO 67985807 and the GA ČR grant No. P202/10/1333.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 123–135, 2014.

of problems for which no efficient deterministic algorithms have been known. Could randomization also help in matrix multiplication?

Perhaps this was the motivation of Rusins Freivalds who in 1977 designed a randomized algorithm with a bounded error probability for verifying matrix multiplication in quadratic randomized time [10], [11]. This algorithm has become one of the first algorithms showing the advantage of randomized algorithm over the deterministic ones also in a domain related to matrix multiplication and no wonder that Freivalds' algorithm has become a standard textbook example illustrating the power of randomized computations over the deterministic ones (cf. [7],[17]). At the same time this result has indicated that for a task related to matrix multiplication, viz. its verification, a quadratic time bound can be achieved.

The idea of Freivalds' algorithm is as follows. In order to verify whether $\mathbf{AB} = \mathbf{C}$ for any three matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ of real numbers of size $n \times n$, Freivalds' algorithm chooses a specific (column) vector $\mathbf{x}$ of length n and compares the product $\mathbf{ABx}$ with the product $\mathbf{Cx}$. Both products can be computed using $O(n^2)$ arithmetical operations (the former product thanks to the associativity of the matrix products: $(\mathbf{AB})\mathbf{x} = \mathbf{A}(\mathbf{Bx})$). The entries of vector $\mathbf{x}$ are uniformly chosen from the set $\{0, 1\}$. It can be shown that if $\mathbf{AB} \neq \mathbf{C}$, then the Freivalds algorithm returns a wrong answer with the probability at most $1/2$ (cf. [7]). The probability of error can be reduced to $1/2^k$ by performing k independent iterations of the algorithm.

Similarly as in the case of matrix multiplication algorithms research effort aiming at the performance improvement of Freivalds' algorithm has followed. The goal was to diminish the amount of randomness in this algorithm (cf. [18], [14]). This has culminated by the recent result of Korec and Wiedermann [15] showing that integer matrix product verification can entirely be derandomized. For real matrices only one random real number is needed in order to verify the correctness of matrix product with probability one. Both verification algorithms run in quadratic time w.r.t. the number of arithmetic operations. The deterministic algorithm for integer matrix verification has straightforwardly lead to the design of a nondeterministic algorithm for computing the product of two integer matrices in quadratic time, answering thus positively the conjecture concerning the complexity of matrix product, albeit only for integer matrices and for non-deterministic algorithms.

In order to completely understand the development sketched above one also has to pay attention to the underlying models of computation. The standard computational model for matrix product is by arithmetic circuits (aka straight line programs) over some field $\mathbb{F}$. The inputs to the circuit are the entries of the two matrices and the output from the circuit are the entries of the resulting matrix product. All entries are elements of $\mathbb{F}$. In the circuit's gates arithmetic operations with the field elements are allowed.

For verification of matrix products a stronger computational model is used. It is a probabilistic model allowing random moves and in addition to the standard arithmetic operations also comparisons between the elements of the field. For the

case of integer matrices an appropriate underlying model is probabilistic unit-cost RAM (cf [1]). When one wants to take into account the size of integers manipulated during a computation the log-cost RAM model should be used. For the case of real matrices, the so-called Real RAM (aka BSS machine [4]) model is used. Essentially, Real RAM is a Random Access Machine with registers that can store arbitrary real numbers and that can compute rational functions over reals at unit cost.

Quantum algorithms for matrix product verification and matrix multiplication have been considered, too (cf. [8], [16]).

Finally, in [15] for computing the product of integer matrices in quadratic time a nondeterministic model has been used. Bringing nondeterminism into the matrix multiplication means a further shift along the scale of the computational power of the underlying computational devices. As we shall see later the departure from the standard models of arithmetical circuits towards more powerful models of computations is not a purposeless move serving for circumventing the existing obstacles by using more powerful tools. Rather, such a proceeding allows approaching the problem from an other end. It might help in the development of a theoretical background for the development of more practical methods.

The present paper fits into the line of research efforts sketched above. Namely, it investigates the benefits that can be gained from considering nondeterministic algorithms in the context of matrix multiplication. It is based on the previous paper [15] in which the framework for the present approach has been established. It solves some problems that have remained open in the previous paper. Progress along these lines has been enabled by using a new fast deterministic criterion suitable for verification of matrix products over infinite and finite fields. Specifically, the paper brings an efficient algorithm for the nondeterministic multiplications (or deterministic verification of products) of matrices with real entries. On a Real RAM this task can be done deterministically using $O(n^2 \log n)$ arithmetical operations. When compared to the probabilistic algorithm for the latter task from [15], the factor of $\sim \log n$ seems to be a penalization for derandomization of the Freivalds' algorithm. Next, an algorithm of similar complexity is designed for multiplication of integer matrices using modular arithmetic. Its advantage over the quadratic-time algorithm for the same task designed in [15] is that the new algorithm does not compute with large numbers; the size of all numbers in this algorithm is bounded by the size of the largest entry in the underlying matrices.

The contribution of the present paper does not merely lie in the design of the previous algorithms which establish new upper bounds on the complexity of matrix multiplication on the non-deterministic models of computations. What is perhaps more important is that these algorithms make use of algebraic techniques that so far have not been often considered in the matrix multiplication research. Among other things these techniques include elements of the theory of Vandermonde matrices, properties of polynomial roots, multipoint polynomial evaluation, primality certificates, fast Fourier transform and number theoretic transform. It is hoped that the insight gained by the use of such techniques will

help to understand better the nature of matrix multiplication algorithms and eventually bring a further progress to the underlying field.

Our results raise a number of interesting questions to which we presently do not know the answers. For instance, is multiplication of integer matrices easier than multiplication of real matrices, under the unit cost model? So far we do not know an algorithm for nondeterministic multiplication of real matrices of quadratic complexity, albeit such an algorithm for integer matrices in known. By the way, the existence of the latter algorithm implies that any proof for superquadratic lower bound for integer matrix multiplication must be constructed so as not to hold for nondeterministic computations or must take into account the size of integers stored in the RAM registers. Nevertheless, perhaps the most important question arising from our research is the question, whether nondeterminism helps in matrix multiplication. A positive answer would be a surprise since it would put matrix multiplication among the problems that separate determinism from nondeterminism. In any case, our results seem to strengthen the hope that there exist substantially better deterministic algorithms for matrix multiplication than those known today.

The structure of the paper is as follows. In Section 2 those results from the predecessor paper [15] are briefly reviewed that will be used in the elaboration of the new results presented here. Especially, Lemma 1 is presented here that forms the starting point of all subsequent considerations, inclusively of a nondeterministic Algorithm 1 for multiplication of integer matrices in quadratic time on a unit cost RAM (with no bounds on the size of RAM registers). In Section 3 it is shown that ideas from Algorithm 1 cannot be straightforwardly used for multiplication of real matrices. To that end a corollary of Lemma 1 is proved serving as a new criterion enabling efficient verification of the product of real matrices. This results into the design of a nondeterministic Algorithm 2 for multiplication of real matrices on Real RAM in time $O(n^2 \log n)$. In Section 4 we return to the problem of multiplication of integer matrices using only registers of bounded-size. This is achieved by using modular arithmetic over a properly selected finite field. The resulting Algorithm 3 is of complexity $O(n^2 \log n)$ on a unit-cost RAM and computes with registers whose size is proportional to the size of the largest entry in the underlying integer matrices. In order to achieve their complexity as stated here both Algorithm 2 and Algorithm 3 make use of an appropriate version of the fast Fourier transform tuned to the underlying number field. Section 5 contains an overview of the results presented in this paper, open problems and conclusions.

2 Preliminaries

The key to all results in this paper is the following lemma from [15] which is related to the theory of Vandermonde matrices. Here, we recall this lemma inclusively of its proof because we will refer to it in our further considerations:

Lemma 1. *If* $\mathbf{D} \neq \mathbf{0}$ *is a real matrix of size* $n \times n$ *then there are at most* $n-1$ *real numbers* r *such that* $\mathbf{D}(1, r, r^2, \ldots, r^{n-1})^T = \mathbf{0}$

Proof: Since $\mathbf{D} = \{d_{i,j}\} \neq \mathbf{0}$ at least one row of $\mathbf{D}$ is non-zero. In the resulting matrix-vector product there is one algebraic equation in indeterminate r of degree less than n corresponding to this non-zero row. If j is the index of this row then the respective equation is of form $\sum_{i=1}^{n-1} d_{i,j}x^i = 0$ and for some i, $d_{i,j} \neq 0$. This equation has at most $n-1$ real roots. □

Based on the previous lemma a simple probabilistic algorithm of quadratic time complexity for verifying the product of two matrices has been designed in [15]. We sketch the respective algorithm since it will be the basis of our further improvements.

To verify $\mathbf{AB} = \mathbf{C}$, we pick a "random" real number r, create vector $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$ and compute $\mathbf{Y} = \mathbf{A}(\mathbf{Bx}) - \mathbf{Cx}$.

If $\mathbf{Y} = \mathbf{0}$ then $\mathbf{AB} = \mathbf{C}$ with probability 1 (because among all reals there are at most $n-1$ "bad" numbers r causing $(\mathbf{AB} - \mathbf{C})\mathbf{x} = \mathbf{0}$ even if $\mathbf{AB} \neq \mathbf{C}$).

If $\mathbf{Y} \neq \mathbf{0}$ then $\mathbf{AB} \neq \mathbf{C}$ "for sure".

Obviously, $\mathbf{Y}$ can be computed in $O(n^2)$ operations if matrix-vector products are computed first.

The probabilistic step in this algorithm could be eliminated if we could deterministically choose r in such a way that $(\mathbf{AB} - \mathbf{C})\mathbf{x} = \mathbf{0}$ if and only if $\mathbf{AB} = \mathbf{C}$. That is, if $\mathbf{D} = \mathbf{AB} - \mathbf{C} = \{d_{i,j,}\}$, such an r cannot be a root of any polynomial of form $P_i(r) = \sum_{j=1}^{n} d_{i,j}x^{j-1} = 0$ for $i = 1, 2, \ldots, n$. Such an r can be found using the so-called Cauchy's bound ([9], p. 122, or the textbook [13], p. 82):

Theorem 1. *Let $P(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_1 x + a_0$ be a polynomial with real coefficients. If x is a real root of $P(x) = 0$ then $|x| < 1 + A/|a_k|$, with $A = \max_{i=0}^{k-1}\{|a_i|\}$.*

One can see that in order to upper-bound the magnitude of the roots we have to know coefficient a_k and the maximum of the absolute value of all coefficients in a polynomial. In our case, $k = n$ in the previous theorem, $d_{i,j} = \sum_{j=1}^{n} a_{i,j}b_{j,i} - c_{i,j}$ and $P_i(r) = \sum_{j=1}^{n} d_{i,j}r^{j-1} = 0$ for $i = 1, 2, \ldots, n$.

Let $c_{max} = \max\{|a_{i,j}|, |b_{i,j}|, |c_{i,j}|\}$. Then the maximal coefficient in any polynomial — the value of A — can be upper-bounded by $nc_{max}^2 + c_{max}$. Further, for integer matrices, and only for integer matrices, for any i, the absolute value of the leading coefficient in front of the highest power of r in $P_i(r)$ can be lower-bounded by 1 since the leading coefficient must be an integer.

Then, from Cauchy's bound it follows that for any i the absolute values of the roots of polynomial $P_i(r)$ with integer coefficients are upper-bounded by $\alpha = nc_{max}^2 + c_{max} + 1$.

Thus, choosing $r \geq \alpha$ in the previous proposition will guarantee that $P_i(r) \neq 0$ and hence $\mathbf{Dx} = \mathbf{0}$ can only hold for $\mathbf{D} = \mathbf{0}$. From these considerations a deterministic algorithm of quadratic complexity for integer matrix multiplication verification and nondeterministic algorithm for integer matrix multiplication follow easily (cf. [15]). Here we will only give the algorithm for nondeterministic matrix multiplication that will be the subject of our further improvements (cf. Algorithm 1).

Algorithm 1:
A non-deterministic integer matrix multiplication algorithm:

1. Input non-zero integer matrices $\mathbf{A}$ and $\mathbf{B}$;
2. Guess matrix $\mathbf{C} \neq \mathbf{0}$ with the absolute values of entries bounded by $r = \alpha$;
3. Compute vector $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$;
4. If $\mathbf{A}(\mathbf{Bx}) = \mathbf{Cx}$ then output $\mathbf{C}$.

In [15] it is shown that Algorithm 1 can be generalized to the case of matrices of rational numbers given as numerator and denominator pairs.

Note that the seemingly similar idea of guessing $\mathbf{C}$ and its subsequent probabilistic verification *à la* Freivalds cannot work since through the error margin of the respective algorithm, however small, matrices $\mathbf{C}$ can come through for which $\mathbf{AB} \neq \mathbf{C}$. Thus, getting rid of probabilistic features of a verification algorithm for matrix multiplication turns out to be a crucial ingredient for the success of our nondeterministic algorithm.

3 Non-deterministic Multiplication of Real Matrices

Can we adjust Algorithm 1 to also work with real matrices? Unfortunately, no. What will not go through for reals is the upper bound on the magnitude of the maximal root for a class of polynomials with real coefficients depending on matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Namely, although we can estimate the size of A in Cauchy's bound, we cannot get a lower bound on a_k whose value can be arbitrary close to 0. This will make the value of A/a_k arbitrarily large and subsequently we cannot get a bound on $|x|$. Thus, it appears that we cannot easily find number r such that $\mathbf{AB} = \mathbf{C}$ if and only if $\mathbf{ABx} = \mathbf{Cx}$ for $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$.

Nevertheless, from Lemma 1 it follows that no matter how we pick n distinct real numbers $r_1, r_2, \ldots, r_n$, among them there will be at least one number r_j that is not a root of the respective polynomial. Such an r_j will certify $\mathbf{AB} \neq \mathbf{C}$ if and only if $\mathbf{ABx}_j \neq \mathbf{Cx}_j$ for $\mathbf{x}_j = (1, r_j, r_j^2, \ldots, r_j^{n-1})^T$. This suggests a possible strategy for a verification algorithm. For $j = 1, 2, \ldots, n$ we test whether $\mathbf{ABx}_j = \mathbf{Cx}_j$. If the equality will be confirmed for all j then $\mathbf{AB} = \mathbf{C}$. Otherwise $\mathbf{AB} \neq \mathbf{C}$.

Thanks to a special form of vectors $\mathbf{x}_j$ it is promising to realize that the product $\mathbf{Bx}_j$ for all $j = 1, 2 \ldots, n$ can be computed faster than in cubic time. Namely, we can reorganize the computation of $\mathbf{Bx}_j$ for $j = 1, 2, \ldots, n$ in such a way that we compute the value of each polynomial defined by each row of $\mathbf{B}$, at points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

Let $P(i, x) = \sum_{j=1}^{n} b_{i,j} x^{j-1}$ be the polynomial corresponding to the product of the i-th row of matrix $\mathbf{B}$ with vector $\mathbf{x} = (1, x, x^2, \ldots, x^{n-1})^T$. Then for each i we have to compute $P(i, x)$ in n different points $x = r_1, r_2, \ldots, r_n$. This is a typical task of multipoint polynomial evaluation that for any set of points can be solved by divide-and-conquer method. This leads to complexity of $O(n \log^2 n)$

operations per one row of $\mathbf{B}$ (i.e., per one i) — cf. [1], [5], [6]. When the points are selected to be the roots of unity fast Fourier transform can be used (cf. [1]). This method saves one factor of $\log n$ in the resulting complexity estimate. In a similar way we can compute $\mathbf{Cx}_i$.

Unfortunately, for computing the next product $\mathbf{A}(\mathbf{Bx}_i)$ the advantage of a specific form of vector $\mathbf{Bx}_i$ is lost and this seems to lead again to an algorithm of cubic complexity. Fortunately, a slightly more complicated verification criterion based on the following corollary of Lemma 1 solves the problem:

Corollary 1. *Let* $\mathbf{D}$ *be a real matrix of size* $n \times n$, *let* r_i *for* $i = 1, 2, \ldots, 2n-1$ *be* $2n-1$ *different reals numbers, let* $\mathbf{x}_i = (1, r_i, r_i^2, \ldots, r_i^{n-1})^T$.

Then $\mathbf{D} = \mathbf{0}$ *if and only if for all* $i = 1, 2, \ldots, 2n-1$, $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i = 0$.

Proof: The left-to-right implication is obvious. For the proof of the reverse direction suppose that under the assumptions of the corollary $\mathbf{D} \neq \mathbf{0}$. Then at least one element of $\mathbf{D}$ would be non-zero. In the vector-matrix-vector product $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i$ this element will give rise to a polynomial in indeterminate r_i of degree at most $2n-2$. This polynomial has at most $2n-2$ real roots and therefore it cannot turn to zero in $2n-1$ distinct real numbers. This is a contradiction and therefore $\mathbf{D} = \mathbf{0}$. □

Translating back to our context of matrix multiplication the previous corollary implies a strategy for a verification algorithm. For $i = 1, 2, \ldots, 2n-1$, we test whether $(\mathbf{x}_i^T \mathbf{A})(\mathbf{Bx}_i) = \mathbf{x}_i^T(\mathbf{Cx}_i)$. If this test fails for some i then $\mathbf{AB} \neq \mathbf{C}$. Otherwise $\mathbf{AB} = \mathbf{C}$ (cf. Algorithm 2). The advantage of the above mentioned criterion is that for computing $\mathbf{x}_i^T \mathbf{A}$, $\mathbf{Bx}_i$, and $\mathbf{Cx}_i$, respectively, again fast multipoint polynomial evaluation algorithms can be used. This time, however, we must evaluate the respective polynomials in $2n-1$ different points, but this does not change the asymptotic complexity of the evaluation task.

Algorithm 2:
A non-deterministic real matrix multiplication algorithm:

1. Input real matrices $\mathbf{A} \neq \mathbf{0}$ and $\mathbf{B} \neq \mathbf{0}$;
2. Guess matrix $\mathbf{C} \neq \mathbf{0}$;
3. if (for $i = 1, 2, \ldots, 2n-1$ and $\mathbf{x}_i = (1, i, i^2, \ldots, i^{n-1})^T$, $(\mathbf{x}_i^T \mathbf{A})(\mathbf{Bx}_i) = \mathbf{x}_i^T(\mathbf{Cx}_i)$), then output($\mathbf{C}$)

Therefore, all the necessary vector-matrix or matrix-vector products can be computed in time $O(n^2 \log^2 n)$ arithmetic operations, or even in $O(n^2 \log n)$ using fast Fourier transform. Hence the entire verification can be done within the same arithmetic complexity.

Theorem 2. *For a Real RAM there exists a non-deterministic algorithm for computing matrix product of real matrices using* $O(n^2 \log n)$ *arithmetical operations.*

For bounded coefficient arithmetic circuits over the real or complex numbers Raz [22] proved a lower bound of order $\Omega(n^2 \log n)$. This bound does not apply to our case since we are using a different computational model than Raz — namely the model where nondeterministic computations with arbitrary large reals and equality tests are allowed.

4 Non-deterministic Multiplication of Integer Matrices Using Modular Arithmetic

Algorithm 1 from Section 2 has one serious drawback: it computes with very large integers since the largest entry in vector $\mathbf{x}$ is as high as $(2n-1)^{n-1}$. In matrix $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ this entry can subsequently be multiplied by α, where α, defined in Section 2, denotes the bound on the maximal value of coefficient of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. That is, the bit representation of such an entry requires $O(n \log n + \log \alpha) = O(n \log n)$ bits for a sufficiently large n. Hence, for such n the size of this entry is dominated by the term $O(n \log n)$ rather than by the size of the maximal possible coefficient in matrix $\mathbf{D}$. Fortunately, this situation can be remedied by using modular arithmetic.

However, one must be careful when trying to adjust the basic Algorithm 1 from Section 2 to modular arithmetic. The first problem arises because, as it appears, in finite fields no analogue of Cauchy's bound is known. This is probably due to the fact that finite fields cannot be ordered. A number that is not a root of a polynomial in the ring $\mathbb{Z}$ of integers may become a root of the same polynomial in a finite field $\mathbb{Z}_p$, for some p prime. For instance, in $\mathbb{Z}$ polynomial $x^2 + 1$ has no real roots. The same polynomial in $\mathbb{Z}_5$ has two roots: 2 and 3. Thus, we must use a different criterion than that from Lemma 1 for identifying a zero matrix.

It appears that we can design a criterion similar to that used for the case of real numbers (cf. Corollary 1). This is possible since a polynomial of degree n has at most n roots in any finite field. The criterion from Corollary 1 can be adjusted to the case of finite fields thanks to the following lemma:

Lemma 2. *Let* $\mathbf{D}$ *be an integer matrix of size* $n \times n$ *with* $\max\{|d_{i,j}|\} \leq \delta$. *Let* $\mathbb{Z}$ *be the ring of integers and let* $\mathbb{Z}_p$ *be a finite field of integers* $\mod p$, *for some prime* $p > \max\{\delta, 2n-1\}$. *Let further* r_i *for* $i = 1, 2, \ldots, 2n-1$, $r_i < p$ *be* $2n-1$ *different integers and let* $\mathbf{x}_i = (1, r_i, r_i^2, \ldots, r_i^{n-1})^T$.

Then $\mathbf{D} = \mathbf{0}$ *(in* $\mathbb{Z}$*) if and only if for all* $i = 1, 2, \ldots, 2n-1$, $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i \equiv 0 \pmod{p}$ *(in* $\mathbb{Z}_p$*).*

Proof: The left-to-right implication is obvious. For the proof of the reverse direction suppose that under the assumptions of the lemma $\mathbf{D} \neq \mathbf{0}$. Thanks to this and the choice of p at least one element $d_{i,j}$ of $\mathbf{D}$ must be non-zero $\mod p$: $d_{i,j} \not\equiv 0 \pmod{p}$. This means that $\mathbf{D} \not\equiv \mathbf{0} \pmod{p}$. In the vector-matrix-vector product $(\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i) \mod p$ this element will give rise to a non-zero polynomial in indeterminate r_i of degree at most $2n-2$. This polynomial has at most $2n-2$ roots in $\mathbb{Z}_p$ and therefore it cannot turn to zero in $2n-1$ distinct numbers r_i

mod $p = r_i$. This, however, implies that it cannot hold $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i \equiv 0 \pmod{p}$ for all $i = 1, 2, \ldots, 2n-1$. This is a contradiction. □

This Lemma opens the way for designing an algorithm using modular arithmetic for integer matrix multiplication similar to Algorithm 2 from Section 3.

In order to do so we have to choose a field of a suitable characteristic p. We will want to keep p relatively small, with $p \geq \alpha$. While avoiding computations with numbers above p, thanks to the definition of α such a choice of p will preserve the value of entries of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ when considered in $\mathbb{Z}$. However, one must be careful with this idea since in arithmetic $\pmod{p}$ each element (except zero) has two representations: one positive and one negative: for any a with $-p < a < p$, $a \equiv a - p \pmod{p}$. Therefore, when guessing the values of matrix $\mathbf{C}$ the correct values (with a correct sign) — i.e., those from $\mathbb{Z}$ — must be guessed because these are the values that would eventually be output. This is because, as we shall see, the correctness check is performed in arithmetic $\pmod{p}$ (as suggested by Lemma 2) through which both the positive and the negative representation of the same non-zero number could pass. This could be prevented once we knew what would be the sign of each entry in $\mathbf{C} = \{c_{i,j}\}$. In general, it appears that for arbitrary matrices $\mathbf{A}$ and $\mathbf{B}$ the sign of each $c_{i,j}$ cannot be determined without actually computing it. Nevertheless, for special matrices this can be done. E.g., when all entries of $\mathbf{A}$ and $\mathbf{B}$ are positive, so will be the entries of $\mathbf{C}$. This motivates the following definition.

A matrix $\mathbf{A}$ with all entries non-negative (non-positive) will be called a *non-negative (non-positive) matrix.* A matrix $\mathbf{A}$ that is either non-negative or non-positive will be called an *equisigned* matrix. Observe that thanks to the definition of matrix product in which rows are multiplied by columns the product of two equisigned matrices is again an equisigned matrix.

For any real matrix $\mathbf{A} = \{a_{i,j}\}$, $\mathbf{A}^+$ is a matrix defined as $\mathbf{A}^+ = \{a^+_{i,j}\}$, with $a^+_{i,j} = a_{i,j}$ if $a_{i,j} > 0$ and $a^+_{i,j} = 0$ if $a_{i,j} \leq 0$. Matrix $\mathbf{A}^-$ consisting of non-positive entries of matrix $\mathbf{A}$ is defined similarly. Obviously, $\mathbf{A}^+$ and $\mathbf{A}^-$ are equisigned matrices and any non-equisigned matrix $\mathbf{A}$ can be uniquely written as $\mathbf{A} = \mathbf{A}^+ + \mathbf{A}^-$. The following lemma is nearly obvious:

Lemma 3. *Let* $\mathbf{A}$, $\mathbf{B}$ *and* $\mathbf{C}$ *be three matrices of real numbers of size* $n \times n$. *Then* $\mathbf{AB} = \mathbf{C}$ *if and only if* $\mathbf{A}^+\mathbf{B}^+ + \mathbf{A}^-\mathbf{B}^- = \mathbf{C}^+$ *and* $\mathbf{A}^+\mathbf{B}^- + \mathbf{A}^-\mathbf{B}^+ = \mathbf{C}^-$.

It follows that instead of multiplying two general matrices $\mathbf{A}$ and $\mathbf{B}$ it is enough to add four products of equisigned matrices. Note that by this we have solved the problem of a possible dual representation of entries in matrix $\mathbf{C}$ in modular arithmetic. In an equisigned matrix all elements are either non-negative or non-positive. Therefore, in the sequel, w.l.o.g. we will only consider multiplication of equisigned matrices.

Now, let us return to the problem of computing the product of integer matrices using arithmetic $\pmod{p}$.

The question remains how to find a suitable p efficiently, i.e., in a way that would not deteriorate the (nearly) quadratic algebraic complexity of the verification algorithm. To that end we use the Bertrand's postulate stating that for any

integer $k > 1$, there is a prime number p such that $k < p < 2k$ (cf. [21]). Since we are aiming at a nondeterministic algorithm a suitable prime p will be guessed and subsequently tested for primality. An appropriate test for our purpose is the check devised by Pratt in 1975 [19] that can be nondeterministically realized with the help of $O(\log^2 p)$ modular multiplication on a unit-cost RAM.

The resulting Algorithm 3 mirroring Algorithm 2 is given below. This algorithm is based on Lemma 2 in which we put $\mathbf{D} = \mathbf{AB} - \mathbf{C}$. In order to determine p we need to estimate δ from Lemma 2, i.e., to bound the absolute values of $\mathbf{D}$.

Clearly, the maximal values in $\mathbf{C}$ are bounded by $\beta = n(\max\{|a_{i,j}|, |b_{i,j}|\})^2$ and hence the maximal absolute value in matrix $\mathbf{D}$ is at most $\delta = 2\beta$. Thus, choosing $p > \max\{\delta, 2n-1\}$ will do.

Algorithms 3:
A log-cost efficient non-deterministic multiplication algorithm for equisigned integer matrices:

1. Input two non-zero equisigned matrices $\mathbf{A}$ and $\mathbf{B}$;
2. Guess the equisigned matrix $\mathbf{C} \neq \mathbf{0}$ with the absolute values of its entries bounded by β;
3. Check, whether the signs of elements of $\mathbf{C}$ correspond to the signs of the input equisigned matrices;
4. Compute $\alpha := \max\{2\beta, 2n-1\}$;
5. Guess $p : \alpha < p < 2\alpha$ and using Pratt's algorithm verify that p is a prime;
6. For $i = 1, 2, \ldots, 2n-1$ compute vectors $\mathbf{x}_i = (1, i, i^2 \bmod p, \ldots, i^{n-1} \bmod p)^T$;
7. Using multipoint polynomial evaluation in arithmetic $\bmod p$ deterministically verify whether (for $i = 1, 2, \ldots, 2n-1$, $(\mathbf{x}_i^T \mathbf{A})(\mathbf{B}\mathbf{x}_i) = \mathbf{x}_i^T(\mathbf{C}\mathbf{x}_i)$);
   if YES then Output $\mathbf{C}$.

As in the case of the previous algorithm the number of operation $(\bmod p)$ of this algorithm is $O(n^2 log^2 n)$. Its correctness follows from several facts. After Step 2, there exist many computational paths (one for each guess of $\mathbf{C}$) in the computation, but the verification in Step 7 selects exactly one of them satisfying $(\mathbf{x}_i^T \mathbf{A})(\mathbf{B}\mathbf{x}_i) \equiv \mathbf{x}_i^T(\mathbf{C}\mathbf{x}_i) \pmod p$ for all i. Note that by choosing $r_i = i$ for $i = 1, 2, \ldots, 2n-1$ the values of r_is are all different and therefore by Lemma 2 in which we put $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ the latter verification is equivalent to verifying $\mathbf{AB} = \mathbf{C}$. Because the equisigned matrices were considered in Steps 1 and 2 matrix $\mathbf{C}$ verified and produced in Step 7 has the correct values (and signs). Obviously, since all computation is done $(\bmod p)$ the size of all RAM registers is bounded by $O(\log p) = O(\log \alpha)$.

Theorem 3. *For a unit-cost RAM there exists a non-deterministic integer matrix multiplication algorithm using $O(n^2 \log^2 n)$ modular arithmetical operations with operands proportional to the size of the largest entry in all involved matrices.*

Consider now the complexity of the previous computation under the logarithmic cost. Let $M(q)$ be the complexity of multiplying q-bit numbers on a log-cost

RAM. The best known upper bound $M(q) = \log q \; 2^{\Theta(\log^* q)}$ is due to Fürer [12] and De, Saha, Kurur and Saptharishi [2]. When measured in bit operations, $O(\log^3 p \; 2^{\Theta(\log^* q)})$ steps suffice to check a proof of p's primality in Step 4.

Since $p : \alpha < p < 2\alpha$ is the largest number occurring in the computation, the bit complexity of Step 6 and 7 is $O(n^2 \log^2 nM(\log p))$. For sufficiently large n the latter value dominates the complexity of the entire algorithm.

Similarly as in the case of computing with real numbers we can select the interpolation points used in multipoint polynomial evaluation (in Step 7) carefully in order to exploit a variant of fast Fourier transform also in the case of a finite field. Such transform is called number theoretic transform (NTT). Here an additional problem arises — namely the fact that not every finite field possesses an n-th root of unity. There exists numerous literature devoted to the optimal selection of parameters n (number of points), p and m (the size of the base field which must be of form p^m for p prime and m a natural number) of NTT (cf. [20] or [1]). The basic results here are that a logarithmic factor in the number of arithmetic operations can be saved, indeed, when compared with the general case of multipoint polynomial evaluation over arbitrary set of points. We will not follow this line any further because it is not directly related to the complexity of matrix products.

Theorem 4. *For a log-cost RAM there exists a non-deterministic integer matrix multiplication algorithm using $O(n^2 \log nM(\log p))$ bit operations.*

Obviously, this algorithm can be implemented on the standard nondeterministic model of Turing machines with a similar efficiency, too. This is to be compared to the bit complexity of the best known deterministic algorithms which is of order $O(n^\omega M(\log p)$, for some $\omega : 2 < \omega < 3$. For any $\omega > 2$ the latter expression is asymptotically worse than our estimation from the previous theorem.

5 Conclusion

We have presented two nondeterministic algorithms for matrix multiplication. Both algorithms make use of a derandomized version of Freivalds' algorithm for verification of matrix products. The first algorithm deals with real numbers and on a Real RAM is of time complexity $O(n^2 \log n)$. The second one computes with integers and on a unit-cost RAM is also of complexity $O(n^2 \log n)$. Moreover, the latter algorithm computes with numbers whose size is bounded by the maximal number occurring within the matrices. The respective complexity bounds hold when variants of fast Fourier transform are used for the multipoint polynomial evaluation used by both algorithms. With respect to their complexity both algorithms are the fastest known sequential algorithms for computing the product of integer and real matrices, respectively.

Of course, due to their nondeterministic nature our algorithms are more of a theoretical than of a practical value. Nevertheless, both of them can be used for deterministic matrix multiplication verification.

But there are more lessons to be taken from our results. First, for the case of integer matrices, they show that any lower bound for matrix multiplication which would be greater than our quadratic upper bound must avoid proof techniques that work both for deterministic and nondeterministic computations. For integer matrices such a lower bound must probably be dependent on the size of matrix entries since Algorithm 1 is of quadratic complexity (i.e., it is optimal). Second, our results pose interesting questions that we cannot answer at present: is there an intrinsic difference between the complexity of integer and real matrices multiplication (cf. Algorithm 1 vs. Algorithm 2)? Does matrix multiplication belong among problems that separate determinism from nondeterminism, i.e., does the use of nondeterminism in matrix multiplication help? Last but not least, it is interesting to observe that in order to prove our results we have used, in addition to nondeterminism, more powerful tools from computational algebra than those usually used in the design of fast "classical" matrix multiplication algorithms: comparisons, theory of Vandermonde matrices, properties of polynomial roots, primality certificate, modular arithmetic, and fast Fourier transform. Could it be the case that the use of such techniques is the key to the design of fast deterministic matrix multiplication algorithms?

We believe that our results have brought a further insight into the nature of matrix multiplication. They strengthen the hope that perhaps deterministic algorithms for matrix product of a similar complexity will be found.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms, p. 470. Addison-Wesley (1974)
2. De, A., Kurur, P.P., Saha, C., Saptharishi, R.: Fast Integer Multiplication Using Modular Arithmetic. SIAM J. Comput. 42(2), 685–699 (2013)
3. Ar, S., Blum, M., Codenotti, B., Gemmel, P.: Checking approximate computations over the reals. In: Proc. 25th ACM STOC, pp. 786–795 (1993)
4. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation, p. 452. Spinger (1998)
5. Borodin, A., Moenck, R.: Fast modular transform. Journal of Computer and System Sciences 8, 366–386 (1974)
6. Bostan, A., Schost, E.: Polynomial evaluation and interpolation on special sets of points. Journal of Complexity 21(4), 420–446 (2005)
7. Brassard, G., Bratley, P.: Fundamentals of Algorithmics, 524 pages. Prentice Hall, Englewood Cliffs (1996)
8. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of SODA, pp. 880–889 (2006)
9. Cauchy, A.-L.: Mémoire sur la résolution numérique des équations et sur la théorie de l'élimination. Exercices de mathématiques, 40e et 41e livraisons, dans OC (2), IX, 87–161 (1829)
10. Freivalds, R.: Probabilistic machines can use less running time. In: Information Processing 1977, Proceedings of the IFIP Congress 1977, pp. 839–842 (1977)
11. Freivalds, R.: Fast Probabilistic Algorithms. In: Bečvář, J. (ed.) Mathematical Foundations of Computer Science 1979. LNCS, vol. 79, pp. 57–69. Springer, Heidelberg (1979)

12. Fürer, M.: Faster Integer Multiplication. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13 (2007)
13. Householder, A.: The Numerical Treatment of a Single Nonlinear Equation. McGraw-Hill, New York (1970)
14. Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. Inf. Process. Lett. 45(2), 107–110 (1993)
15. Korec, I., Wiedermann, J.: Deterministic verification of integer matrix multiplication in quadratic time. In: Geffert, V., et al. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 375–382. Springer, Heidelberg (2014)
16. Le Gall, F., Nishimura, H.: Quantum Algorithms for Matrix Products over Semirings. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 331–343. Springer, Heidelberg (2014)
17. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
18. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. 22(4), 838–856 (1993)
19. Pratt, V.: Every prime has a succinct certificate. SIAM Journal on Computing 4, 214–220 (1975)
20. Preparata, F.P., Sarwate, D.V.: Computational complexity of Fourier transform over finite fields. Math. of Comp. 31(139), 740–751 (1977)
21. Ramanujan, S.: A Proof of Bertrand's Postulate. J. Indian Math. Soc. 11, 181–182 (1919)
22. Raz, R.: On the complexity of matrix product. In: Proc. of the 34th Annual ACM Symposium on Theory of Computing. ACM Press (2002)
23. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. 13, 354–356 (1969)
24. Vassilevska Williams, V.: Multiplying matrices faster than Coppersmith-Winograd. In: ACM STOC, pp. 887–898 (2012)

Not All Multi-Valued Partial CFL Functions Are Refined by Single-Valued Functions (Extended Abstract)

Tomoyuki Yamakami

Department of Information Science, University of Fukui
3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract. We give an answer to a fundamental question, raised by Konstantinidis, Santean, and Yu [Acta Inform. 43 (2007) 395–417], of whether all multi-valued partial CFL functions can be refined by single-valued partial CFL functions. We negatively solve this question by presenting a special multi-valued partial CFL function as an example function and by proving that no refinement of this particular function becomes a single-valued partial CFL function. This contrasts an early result of Kobayashi [Inform. Control 15 (1969) 95–109] that multi-valued partial NFA functions are always refined by single-valued NFA functions. Our example function turns out to be unambiguously 2-valued, and thus we obtain a stronger separation result, in which no refinement of unambiguously 2-valued partial CFL functions can be single-valued. Our proof consists of manipulations and close analyses of underlying one-way one-head nondeterministic pushdown automata equipped with write-only output tapes.

Keywords: multi-valued partial function, CFL function, NFA function, refinement, pushdown automaton, context-free language, stack history.

1 Resolving a Fundamental Question

Since early days of automata and formal language theory, multi-valued partial functions,[⋆] which are also known as transductions, computed by various types of automata equipped with supplemental write-only output tapes have been investigated extensively. Among them, we intend to spotlight *CFL functions* (also known as algebraic transductions), which are computed by *one-way one-head nondeterministic pushdown automata* (succinctly abbreviated as npda's) *with write-only output tapes*. These functions naturally inherit certain distinctive traits from context-free languages; however, their behaviors are in essence quite different from the behaviors of the language counterparts. Intriguing properties of those functions have been addressed in the past literature (e.g., [1–3, 6, 14]).

When the number of output values is restricted to at most one, we obtain *single-valued* functions. Concerning a relationship between multi-valued and single-valued partial functions, multi-valued partial functions in general cannot

⋆ We often call those multi-valued partial functions just "functions."

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 136–150, 2014.

be single-valued; therefore, it is more appropriate to ask a question of whether multi-valued partial functions can be *refined* by single-valued partial functions, where "refinement" refers to a certain natural restriction on the outcomes of multi-valued functions. To be more precise, we say that a function g is a *refinement* of another function f [7] (which was also called "uniformization" [6]) if and only if (i) f and g have the same domain and (ii) for every input x in the domain of f, all output values of g on x are also output values of f on the same input x. When g is particularly single-valued, g acts as a "selecting" function that picks exactly one value from a set of output values of f on x. This refinement notion is known to play a significant role in language recognition. In a polynomial-time setting, for instance, if we can effectively find an accepting computation path of any polynomial-time nondeterministic Turing machine, then every multi-valued partial NP function (computed by a certain polynomial-time nondeterministic Turing machine) has a refinement in the form of single-valued NP function. The "no-refinement" claim therefore leads to a negative answer to the long-standing P =?NP question.

We intend to discuss the same refinement question regarding CFL functions. In this line of research, the first important step was taken by Kobayashi [5] in 1969. He gave an affirmative answer to the refinement question for multi-valued partial NFA functions, which are computed by one-way one-head nondeterministic finite automata (or nfa's, in short) with write-only output tapes; namely, multi-valued partial NFA functions can be refined by certain single-valued partial NFA functions. Konstantinidis, Santean, and Yu [6] discussed the same question for CFL functions. They managed to obtain a partial affirmative answer but left the whole question open, probably due to a technical limitation of their algebraic treatments of CFL functions.

This paper is focused on CFL functions whose output values are particularly produced by npda's running in linear time** (that is, all computation paths terminate in time $O(n)$, where n is the size of input) with write-only output tapes. By adopting succinct notations from [12], we express as CFLMV a collection of all such CFL functions and we also write CFLSV for a collection of all single-valued functions in CFLMV. As a concrete example of our CFL function, let us consider f defined by setting $f(1^n\#x)$ to be a set of all substrings of x of length between 1 and n exactly when $1 \leq n \leq |x|$. This function f is a multi-valued partial CFL function and the following function g is an obvious refinement of f: let $g(1^n\#x)$ consist only of the first symbol of x whenever $1 \leq n \leq |x|$. Notice that g belongs to CFLSV.

Given two classes $\mathcal{F}$ and $\mathcal{G}$ of multi-valued partial functions, the notation $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ means that every function in $\mathcal{F}$ can be refined by a certain function in $\mathcal{G}$. Using these notations, the aforementioned refinement question regarding CFL functions can be neatly rephrased as follows.

** This linear time-bound ensures that every CFL function produces only at most an exponential number of output values and it therefore becomes an NP function. This fact naturally extends a well-known containment of CFL $\subseteq$ NP.

Question 1. *Is it true that* CFLMV $\sqsubseteq_{ref}$ CFLSV*?*

Various expansions of CFLMV are possible. Yamakami [14], for instance, introduced a hierarchy $\{\Sigma_k^{\mathrm{CFL}}\mathrm{MV}, \Pi_k^{\mathrm{CFL}}\mathrm{MV} \mid k \geq 1\}$ of multi-valued partial functions built upon CFL functions by applying Turing relativization and a complementation operation. Its single-valued version is customarily denoted by $\{\Sigma_k^{\mathrm{CFL}}\mathrm{SV}, \Pi_k^{\mathrm{CFL}}\mathrm{SV} \mid k \geq 1\}$. Our focal question, Question 1, can be further generalized to the following.

Question 2. *Does* $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ *hold for each index* $k \geq 1$*?*

Yamakami [14] also shed partial light on this general question when $k \geq 3$. He showed that, for every index $k \geq 3$, $\Sigma_{k-1}^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$ implies $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$, where Σ_k^{CFL} is the kth level of the *CFL hierarchy* [13], which is a natural analogue of the well-known polynomial(-time) hierarchy. Since the collapse of the CFL hierarchy is closely related to that of the polynomial hierarchy, an answer to Question 2 (when $k \geq 3$) could be quite difficult to obtain. Nevertheless, the remaining cases of $k = 1, 2$ have been left unsolved.

In this paper, without relying on any unproven assumptions, we solve Question 2 negatively when $k = 1$; therefore, our result completely settles Question 1. Our solution actually gives an essentially stronger statement than what we have discussed so far. To clarify this point, we first introduce a function class CFL2V as a collection of all functions f in CFLMV satisfying the condition that the number of output values of f on each input should be at most 2.

Theorem 3. CFL2V $\not\sqsubseteq_{ref}$ CFLSV.

Since CFLSV $\subseteq$ CFL2V $\subseteq$ CFLMV holds, Theorem 3 clearly leads to a negative answer to Question 1. The proof of the theorem is essentially a manifestation of the following intuition: an npda relying on limited functionality of its memory device cannot simulate two independent computation paths simultaneously.

Instead of providing a detailed proof for Theorem 3, we wish to present a simple and clear argument to demonstrate a slightly stronger result regarding a subclass of CFL2V. To explain such a subclass, we first address that even if a function f is single-valued, its underlying npda may have numerous computation paths producing the same value of f on each input. Let us call an npda N with a write-only output tape *unambiguous* if, for every input x and any output value y, N has exactly one accepting computation path producing y. Let UCFL2V denote a class of all 2-valued partial functions computed in linear time by unambiguous npda's with output tapes. Succinctly, those functions are called *unambiguously 2-valued*. Obviously, UCFL2V $\subseteq$ CFL2V holds.

Throughout this paper, we wish to show the following stronger separation result (than Theorem 3), which is referred to as the "main theorem."

Theorem 4 (Main Theorem). UCFL2V $\not\sqsubseteq_{ref}$ CFLSV.

Following a brief explanation of key notions and notations in Section 2, we give in Section 3 the proof of Theorem 4, completing the proof of Theorem 3 as

well. Our proof starts in Sections 3.1 with a presentation of our example function h_3, a member of UCFL2V. The proof then proceeds, by contradiction, with an assumption that a certain refinement, say, g of h_3 belongs to CFLSV. In Section 3.2, the proof requires an introduction of "colored" automaton—a new type of automaton having no output tape—which simulates an npda equipped with an output tape that computes g. To lead to the desired contradiction, the proof further exploits special properties of such a colored automaton by analyzing the behaviors of its stack history (i.e., time transitions of stack contents) generated by this colored automaton. The detailed analysis is presented in Sections 3.3–3.6. All proofs omitted here will appear in a forthcoming complete paper.

2 Preliminaries

We wish to explain key notions and notations necessary to read through the rest of this paper.

Let $\mathbb{N}$ denote a set of all nonnegative integers and define $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Given a number $n \in \mathbb{N}^+$, the notation $[n]$ expresses an integer set $\{1, 2, 3, \ldots, n\}$. An *alphabet* is a finite nonempty set of "symbols" or "letters." Given alphabet Σ, a *string over* Σ is a finite series of symbols taken from Σ and $|x|$ denotes the *length* (or *size*) of string x. We use λ for the *empty string*. A *language over* Σ is a subset of Σ^*, where Σ^* is a set of all strings over Σ. Given two strings x and y over the same alphabet, $x \sqsubseteq y$ indicates that x is a *substring* of y; namely, for certain two strings u and v, y equals uxv. Moreover, given a string x and an index $i \in [|x|]$, the notation $(x)_i$ expresses a unique substring made up only of the first i symbols of x. Clearly, $(x)_i \sqsubseteq x$ holds. The notation $|C|$ for finite set A refers to its *cardinality*.

Let us consider multi-valued partial functions, each of which maps elements of a given set to subsets of another set. Slightly different from a conventional notation (e.g., [7, 8]), we write $f : A \to \mathcal{P}(B)$ for two sets A and B to refer to a multi-valued partial function that takes an element in A as input and produces a certain number of elements in B, where $\mathcal{P}(A)$ denotes the *power set* of A. In particular, when $f(x) = \emptyset$, we briefly say that $f(x)$ is *undefined*. The *domain* of f, denoted by $\mathrm{dom}(f)$, is the set $\{x \in A \mid f(x) \text{ is not undefined }\}$. Given a constant $k \in \mathbb{N}^+$, f is *k-valued* if $|f(x)| \leq k$ holds for every input x in A. For two multi-valued partial functions $f, g : A \to \mathcal{P}(B)$, we say that g is a *refinement* of f (or f is *refined by* g), denoted $f \sqsubseteq_{ref} g$, if (i) $\mathrm{dom}(f) = \mathrm{dom}(g)$ and (ii) $g(x) \subseteq f(x)$ (set inclusion) holds for every $x \in \mathrm{dom}(f)$. For any function classes $\mathcal{F}$ and $\mathcal{G}$, the succinct notation $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ means that every function in $\mathcal{F}$ has a certain refinement in $\mathcal{G}$.

Our mechanical model of computation is a *one-way one-head nondeterministic pushdown automaton* (or an *npda*, in short) with/without a write-only output tape, allowing λ-moves (or λ-transitions). We use an infinite input tape, which holds two special endmarkers: the left endmarker ¢ and the right endmarker \$. In addition, we use a semi-infinite output tape on which its tape head is initially positioned at the first (i.e., the leftmost) tape cell and moves in one

direction to the right unless it stays still. Formally, an npda M with an output tape is a tuple $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with a finite set Q of inner states, an input alphabet Σ, a stack alphabet Γ, an output alphabet Θ, the initial state $q_0 \in Q$, the bottom marker $Z_0 \in \Gamma$, a set Q_{acc} (resp., Q_{rej}) of accepting (resp., rejecting) states with $Q_{halt} \subseteq Q$, and a transition function $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times (\Theta \cup \{\lambda\}))$, where $\check{\Sigma} = \Sigma \cup \{\mathfrak{c}, \$\}$ and $Q_{halt} = Q_{acc} \cup Q_{rej}$. We demand that M should neither remove Z_0 nor replace it with any other symbol at any step of its computation. Furthermore, the output tape is *write-only*; namely, whenever M writes a non-blank symbol on this tape, its tape head must move to the right. It is important to recognize two types of λ-move. When δ is applied to (q, λ, γ), M modifies the current contents of its stack and its output tape while neither scanning input symbols nor moving its input-tape head. When $(p, w, \lambda) \in \delta(q, \sigma, \gamma)$ holds, M neither moves its output-tape head nor writes any non-blank symbol onto the output tape.

Whenever we need to discuss an npda *having no output tape*, we automatically drop "Θ" as well as "$\Theta \cup \{\lambda\}$" from the above definition of M and δ. As stated in Section 1, we consider only npda's whose computation paths are all terminate within $O(n)$ steps, where n refers to any input size, and this particular condition concerning the termination of computation is conventionally called the *termination condition* [13]. Throughout this paper, all npda's are implicitly assumed to satisfy this termination condition.

In general, an *output* (outcome or output string) of M along a given computation path refers to a string over Θ written down on the output tape when the path terminates. Such an output is classified as being *valid* (or *legitimate*) if the corresponding computation path is an accepting computation path (i.e., M enters an accepting state along this path). We say that an npda M with an output tape *computes* function f if, on every input x, M produces exactly all the strings in $f(x)$ as valid outputs; namely, for every pair x, y, $y \in f(x)$ if and only if y is a valid outcome of M on input x. Notice that an npda can generally produce more than one valid output strings, its computed function inherently becomes multi-valued. Because invalid outputs produced by M are all discarded from our arguments in the subsequent sections, we will refer to valid outputs as just "outputs" unless otherwise stated.

The notation CFLMV (resp., CFLkV for a fixed $k \in \mathbb{N}^+$) stands for a class of multi-valued (resp., k-valued) partial functions that can be computed by npda's with write-only output tapes in linear time. When $k = 1$, we customarily write CFLSV instead of CFL1V. In addition, we define UCFLkV as a collection of all functions f in CFLkV for which a certain npda with an output tape computes f with the extra condition (called the *unambiguous computation condition*) that, for every input x and every value $y \in f(x)$, there exists exactly one accepting computation path producing y on input x. It then follows that UCFLkV $\subseteq$ CFLkV $\subseteq$ CFLMV. Since any function producing exactly $k + 1$ values cannot be in CFLkV by definition, CFLkV $\neq$ CFL$(k + 1)$V holds; thus, in particular,

we obtain CFLSV $\neq$ CFLMV. Notice that this inequality does not directly lead to the desired conclusion CFLMV $\not\sqsubseteq_{ref}$ CFLSV.

To describe behaviors of an npda's stack, we closely follow terminology from [10, 11]. A *stack content* is formally a series $z_m z_{m-1} \cdots z_1 z_0$ of stack symbols sequentially stored into a stack in such a way that z_0 is the bottom marker Z_0 and z_m is a symbol at the top of the stack. We sometimes refer to a stack content obtained just after the tape head scans and moves off the ith cell of the input tape as a *stack content at the ith position*.

3 Proof of the Main Theorem

Our ultimate goal is to solve negatively a question that was posed in [6] and reformulated in [14] as in the form of Question 1. In what follows, we will present an example function, called h_3, which belongs to UCFL2V, and then give an explanation of why no refinement of this function is found in CFLSV, resulting in the main theorem, namely, UCFL2V $\not\sqsubseteq_{ref}$ CFLSV.

3.1 An Example Function

Our example function h_3 is a natural extension of a well-recognized deterministic context-free language $\{x\#x^R \mid x \in \{0,1\}^*\}$ (marked even-length palindromes), where $\#$ is a distinguished symbol not in $\{0,1\}$. Let us define two supporting languages $L = \{x_1\#x_2\#x_3 \mid x_1, x_2, x_3 \in \{0,1\}^*\}$ and $L_3 = \{w \mid w = x_1\#x_2\#x_3 \in L, \exists (i,j) \in I_3\ [x_i^R = x_j]\}$, where $I_3 = \{(i,j) \mid i,j \in \mathbb{N}^+, 1 \leq i < j \leq 3\}$. We then introduce the desired function h_3 by setting $h_3(w) = \{0^i 1^j \mid (i,j) \in I_3, x_i^R = x_j\}$ if $w = x_1\#x_2\#x_3 \in L$, and $h_3(w) = \emptyset$ if w is not in L. It thus follows that $L_3 = \{w \in L \mid h_3(w) \neq \emptyset\}$. Now, let us claim the following assertion.

Proposition 1. *The above function h_3 is in* UCFL2V.

Proof. Obviously, h_3 is 2-valued. Let us consider the following npda M equipped with a write-only output tape. On any input w, M checks whether w is of the form $x_1\#x_2\#x_3$ in L by moving its input-tape head from left to right by counting the number of $\#$ in w. At the same time, M nondeterministically chooses a pair $(i,j) \in I_3$, writes $0^i 1^j$ onto its output tape, stores x_i into a stack, and then checks whether x_i^R matches x_j by retrieving x_i in reverse from the stack. If $x_i^R = x_j$ holds, then M enters an accepting state; otherwise, it enters a rejecting state. It follows by this definition that, for each choice of (i,j) in I_3, there is at most one accepting computation path producing $0^i 1^j$. It is not difficult to show that M computes h_3. Therefore, h_3 belongs to UCFL2V.

To complete the proof of the main theorem, it suffices to verify the following proposition regarding the existence of refinements of the function h_3.

Proposition 2. *The function h_3 has no refinement in* CFLSV.

3.2 Colored Automata

Our proof of Proposition 2 proceeds by contradiction. To lead to the desired contradiction, we first assume that h_3 has a refinement, say, g in CFLSV. Since g is single-valued, we rather write $g(x) = y$ instead of $g(x) = \{y\}$ for $x \in \mathrm{dom}(f)$. Take an npda N computing g with a write-only output tape. Let N have the form $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times (\Theta \cup \{\lambda\}))$, where $\Sigma = \Theta = \{0, 1\}$.

Unfortunately, we find it difficult to directly analyze the moves of an output-tape head. To overcome this difficulty, we then try to modify N into a new variant of *npda having no output tape*, say, M. As seen later, this modification is possible because g's output values are limited to strings of constant lengths. Now, let us introduce this new machine, dubbed as "colored" automaton, which has no output tapes but uses "colored" stack symbols. Using a finite set C of "colors," a *colored automaton* $M = (Q', \Sigma, \{\mathfrak{c}, \$\}, \Gamma', C, \delta', q'_0, Z_0, Q'_{acc}, Q'_{rej})$ partitions its stack alphabet Γ', except for the bottom marker, into sets $\{\Gamma_\xi\}_{\xi \in C}$; namely, $\bigcup_{\xi \in C} \Gamma_\xi = \Gamma - \{Z_0\}$ and $\Gamma_\xi \cap \Gamma_{\xi'} = \emptyset$ for any distinct pair $\xi, \xi' \in C$. We define a *color* of stack symbol γ to be ξ in C if γ is in Γ'_ξ $(= \Gamma_\xi \cup \{Z_0\})$. Notice that Z_0 has all colors. Given a color $\xi \in C$, we call a computation path of M a *ξ-computation path* if all configurations along this computation path use only stack symbols in color ξ. An *output* of M on input x is composed of all colors ξ in C for which there is an accepting ξ-computation path of M on x.

Lemma 1. *There exists a colored automaton M that computes g.*

Proof Sketch. Recalling the set I_3, we introduce a set $\bar{I}_3 = \{0^i 1^j \mid (i, j) \in I_3\}$ and another set $\bar{I}_3^{part}$ composed of all substrings of any strings in $\bar{I}_3$. Recall the given npda N with a write-only output tape. Now, we want to define a new colored automata $M = (Q', \Sigma, \{\mathfrak{c}, \$\}, \Gamma', \bar{I}_3, \delta', q'_0, Z_0, Q'_{acc}, Q'_{rej})$ that simulates N as follows. Roughly speaking, on any input x, M first *guesses* (i.e., nondeterministically chooses) an output string t of $g(x)$. Whenever N pushes u, M pushes its corresponding color-t symbol $u^{(t)}$ into a stack. Further along this computation path, M keeps using only color-t stack symbols. Instead of having an output tape, M remembers the currently produced string on N's output tape. Whenever N enters an accepting state with an output string that matches the firstly guessed string t of M, M also enters an appropriate accepting state. In other cases, M rejects the input. □

To simplify notation in our argument, we describe the colored automaton M guaranteed by Lemma 1 as $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, I_3, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$. It is also useful to restrict the "shape" of M. A colored automaton M is said to be *in an ideal shape* if M satisfies all of the following six conditions.

1. There are only one accepting state q_{acc} and one rejecting state q_{rej}. Moreover, the set Q of inner states equals $\{q_0, q, q_{acc}, q_{rej}\}$. The machine M is in state q during its computation except for the initial and final configurations.
2. The input-tape head of M always moves.

3. The machine M never aborts its computation; that is, δ is a total function (i.e., $\delta(q, \sigma, \gamma) \neq \emptyset$ holds for any $(q, \sigma, \gamma) \in (Q - Q_{halt}) \times \check{\Sigma} \times \Gamma$).
4. Every stack operation either modifies a single top stack symbol or pushes extra one symbol onto the top of the stack after (possibly) altering the then-top symbol; that is, δ's range is $\mathcal{P}(Q \times (\Gamma \cup \Gamma^2) \times (\Theta \cup \{\lambda\}))$.
5. The stack never becomes empty (excluding the bottom marker Z_0) at any step of the computation except for the initial and the final configurations. In addition, at the first step of reading ¢, M must push a stack symbol onto Z_0 and this stack symbol determines the stack color in the rest of its computation path. After reading \$, M's stack becomes empty.
6. The machine never enters any halting state before scanning the endmarker.

It is well-known that, for any context-free language L, there always exists an npda (with no output tape) in an ideal shape that recognizes L (see, e.g., [4]). Similarly, we can assert the following statement for colored automata.

Lemma 2. *Given any colored automaton, there is always another colored automaton in an ideal shape that produces the same set of output values.*

In the rest of this paper, we fix a colored automaton in an ideal shape, guaranteed by Lemma 2, which computes g correctly.

Hereafter, let us focus on inputs of the form $x\#x^R\#y$ for $x, y \in \{0,1\}^*$. For any $x \in \{0,1\}^*$, we abbreviate the set $\{y \in \{0,1\}^{|x|} \mid y \notin \{x, x^R\}\}$ as H_x. Given $n \in \mathbb{N}^+$, $D^{(n)}_{(i,j)}$ denotes a set of all strings $x \in \{0,1\}^n$ for which there exists an accepting (i,j)-computation path of M on input $x\#x^R\#x$. Obviously, it holds that $D^{(n)}_{(1,2)} \cup D^{(n)}_{(2,3)} = \{0,1\}^n$. It therefore holds, for every length n, that either $|D^{(n)}_{(1,2)}| \geq 2^n/2$ or $|D^{(n)}_{(2,3)}| \geq 2^n/2$. We will discuss the case of $|D^{(n)}_{(2,3)}| \geq 2^n/2$ in Section 3.3 and the case of $|D^{(n)}_{(1,2)}| \geq 2^n/2$ in Section 3.6.

3.3 Case 1: $D_{(2,3)}$ Is Large

Let us consider the first case where the inequality $|D^{(n)}_{(2,3)}| \geq 2^n/2$ holds for infinitely many lengths $n \in \mathbb{N}$. Take an arbitrary number $n \in \mathbb{N}$ that is significantly larger than $3^{|Q|+|\Sigma|+|\Gamma|}$ and also satisfies $|D^{(n)}_{(2,3)}| \geq 2^n/2$. We fix such a number n throughout our proof and we thus tend to drop script "n" whenever its omission is clear from the context; for instance, we often write $D_{(2,3)}$ instead of $D^{(n)}_{(2,3)}$.

By the property of the colored automaton M computing g, it follows that, for any pair $x, y \in \{0,1\}^n$, if $y \notin \{x, x^R\}$, then there always exists a certain accepting $(1,2)$-computation path on input $x\#x^R\#y$; however, there is no accepting $(1,2)$-computation path on input $x\#x^R\#x$ for every x in $D_{(2,3)}$. In addition, no accepting $(1,2)$-computation path exists on input $x\#z\#y$ if $z \neq x^R$. Since there could be a large number of accepting $(1,2)$-computation paths of M on $x\#x^R\#y$, we need to choose one of them arbitrarily and take a close look at this particular path.

For convenience, let $PATH_n$ denote a set of all possible accepting $(1,2)$-computation paths of M on inputs of the form $x\#x^R\#y$ for certain strings $x, y \in \{0,1\}^n$. We arbitrarily fix a *partial assignment* $\pi : D_{(1,2)} \times \{0,1\}^n \to PATH_n$ that, for any element (x,y), if $y \in H_x$, then π picks an accepting $(1,2)$-computation path of M on input $x\#x^R\#y$; otherwise, let $\pi(x,y)$ be undefined for simplicity. For brevity, we abbreviate $\pi(x,y)$ as $p_{x,y}$. Note that $p_{x,y}$ is uniquely determined from (x,y) whenever $\pi(x,y)$ is defined.

Given an accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$, the notation $\gamma_{i,y}^{(x)}$ denotes a stack content obtained by M just after reading off the first i symbols of $x\#x^R\#y$ along this particular path $p_{x,y}$. Furthermore, we abbreviate as $\gamma_y^{(x)}$ the stack content $\gamma_{|x\#x^R\#|,y}^{(x)}$, which is produced just after reading $x\#x^R\#$ of the input $x\#x^R\#y$. Note that, for each $x \in D_{(2,3)}$ and any $y \in H_x$, along an accepting $(1,2)$-computation path $p_{x,y}$ on input $x\#x^R\#y$, M produces unique stack contents $\gamma_{|x\#|,y}^{(x)}$ and $\gamma_y^{(x)}$.

In Sections 3.4–3.5, we plan to evaluate how many strings in $D_{(2,3)}$ satisfy each of the following conditions.

1. Strings x in $D_{(2,3)}$ that make $\gamma_y^{(x)}$ small in size for all $y \in H_x$.
2. Strings x in $D_{(2,3)}$ that make $\gamma_y^{(x)}$ relatively large in size for certain strings $y \in H_x$.

Proposition 3 gives a lower bound of the number of strings in (1), whereas Propositions 4 and 5 provide lower bounds for (2). Those bounds, moreover, guarantee the existence of a string that satisfies both conditions, clearly leading to the desired contradiction.

3.4 Fundamental Properties of a Stack History

In the following series of lemmas and propositions, we will explore fundamental properties of a stack history of M along computation path $p_{x,y}$ on input of the form $x\#x^R\#y$. Those properties are essential in proving the main theorem.

Lemma 3. *Fix $x, y \in \{0,1\}^n$. For any accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$, there is no pair (i_1, i_2) of positions such that $|x| < i_1 < i_2 \leq |x\#x^R\#|$ and $\gamma_{i_1,y}^{(x)} = \gamma_{i_2,y}^{(x)}$. Moreover, the same statement is true when $1 \leq i_1 < i_2 \leq |x|$.*

Lemma 3 can be generalized as follows.

Lemma 4. *Let $x_1, x_2, y_1, y_2 \in \{0,1\}^n$, $i_1, i_2 \in \mathbb{N}$ with $1 \leq i_1, i_2 \leq |x_1\#x_1^R\#|$. Assume that one of the following conditions holds: (i) $i_1 \neq i_2$, (ii) $1 \leq i_1 = i_2 \leq |x_1\#|$ and $(x_1)_{i_1} \neq (x_2)_{i_2}$, and (iii) $(x_1)_{|x_1\#|} = (x_2)_{|x_1\#|}$, $|x_1\#| < i_1 = i_2 \leq |x_1\#x_1^R\#|$, and $(x_1)_{i_1} \neq (x_2)_{i_2}$. It then holds that $\gamma_{i_1,y_1}^{(x_1)} \neq \gamma_{i_2,y_2}^{(x_2)}$.*

Now, we start estimating the lower bound of the number of strings x in $D_{(2,3)}$ for which their corresponding stack contents $\gamma_y^{(x)}$ are small in size for an arbitrary string y in H_x. More specifically, we will verify the following statement.

Proposition 3. *There exist two constants $d_1, d_2 \in \mathbb{N}^+$, independent of (n, x, y), such that $|\{x \in D_{(2,3)} \mid \forall y \in H_x\, [|\gamma_y^{(x)}| < d_1]\}| \geq |D_{(2,3)}| - d_2$.*

Hereafter, we will aim at proving Proposition 3.

Given two strings $u, v \in (\Gamma_{(1,2)})^*$ and a string $z \in \{0,1\}^*$, we say that M *transforms u to v while reading z (along computation (sub)path p)* if M behaves as follows along this subpath p: for a certain w, (i) M starts in state q with uZ_0 in stack, scanning the leftmost input symbol of z, (ii) M then reads z, with no endmarkers, from the input tape, (iii) after reading off z, M enters the same state q with vwZ_0 in stack, and (iv) M does not scan wZ_0. The notation $TF_M(\tau, \sigma)$ expresses a set of all strings of the form $z\#z'$ for $z, z' \in \{0,1\}^*$ such that M transforms τ to σ while reading $z\#z'$.

Lemma 5. *Given any pair (u, v), there is at most one string x' such that x' is a substring of a certain string x in $D_{(2,3)}$ and M transforms u to v while reading x' along a subpath of $p_{x,y}$ for a certain $y \in \{0,1\}^n$.*

Next, we will show a key lemma, necessary to prove Proposition 3. Given a pair (x, y), we define $MSC_{x,y}$ (minimal stack contents) to be a collection of all stack contents γ satisfying the following: there exists a position ℓ with $|x\#| \leq \ell \leq |x\#x^R\#|$ such that (i) $\gamma = \gamma_{\ell,y}^{(x)}$ and (ii) $|\gamma| \leq |\gamma_{\ell',y}^{(x)}|$ holds for any ℓ' satisfying $|x\#| \leq \ell' \leq |x\#x^R\#|$. Condition (ii) indicates that the size of γ is minimum. Note that, when $y \in H_x$, $MSC_{x,y}$ cannot be empty. In addition, by Lemma 4, all elements in $\{\gamma_{i,y}^{(x)} \mid 1 \leq i \leq |x\#x^R\#|\}$ are mutually distinct.

Lemma 6. *There exists a constant $d > 0$, independent of (n, x, y), that satisfies the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma_{\ell,y}^{(x)} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R s r'$, $\ell = |x\#z^R s|$, $\gamma_{|r|,y}^{(x)} = \tau v Z_0$, and $\gamma_{\ell,y}^{(x)} = \sigma v Z_0$ for an appropriate tuple $(r, r', z, s, \sigma, \tau, u, v)$. If $\ell \neq |x\#|$ and $z\#z^R s \in TF_M(\tau, \sigma)$, then $|\gamma_y^{(x)}| \leq d$ holds. Moreover, when n is sufficiently large, $\ell \neq |x\#|$ holds.*

Assuming that Lemma 6 is true, we can prove Proposition 3 in the following manner. Since $MSC_{x,y}$ is non-empty, take an element $\gamma_{\ell,y}^{(x)}$ from $MSC_{x,y}$ with $|x\#| \leq \ell \leq |x\#x^R\#|$. By the size-minimality of $\gamma_{\ell,y}^{(x)}$, there exists an appropriate tuple $(r, r', z, s, \sigma, \tau, u, v)$ that satisfies

(*) $x = rz$, $x^R = z^R s r'$, $\ell = |x\#z^R s|$, $\gamma_y^{(x)} = uvZ_0$, $\gamma_{|r|,y}^{(x)} = \tau v Z_0$, $\gamma_{\ell,y}^{(x)} = \sigma v Z_0$, and $z\#z^R s \in TF_M(\tau, \sigma)$.

By the second part of Lemma 6, except for a certain constant number of x's, it always holds that $\ell \neq |x\#|$. The first part of Lemma 6 provides the desired constant d_1 that upper-bounds $|\gamma_y^{(x)}|$. We therefore obtain the proposition.

To complete the proof of Proposition 3, we still need to verify Lemma 6. This lemma follows from Lemmas 7 and 8. In the first lemma, we want to show that the size of s in (*) is bounded from above by a certain absolute constant.

Lemma 7. *There exists a constant $d_1 > 0$, independent of (n, x, y), satisfying the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R sr'$, $\ell = |x\#z^R s|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, and $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$. If $\ell \neq |x\#|$ and $z\#z^R s \in TF_M(\tau,\sigma)$, then $|s| \leq d_1$ holds.*

Proof. Let $x = rz$, $x^R = z^R sr'$, $\ell = |x\#z^R s|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, and $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$. Since $\ell \neq |x\#|$, it follows that $z \neq \lambda$. Assume that $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$ and $z\#z^R s \in TF_M(\tau,\sigma)$. We first claim that s can be uniquely determined from (τ,σ).

Claim. Let $z_1 \in \{0,1\}^+$ and $s_1 \in \{0,1\}^*$. If $z_1\#z_1^R s_1 \in TF_M(\tau,\sigma)$, then $s = s_1$.

Let us show this claim. Toward a contradiction, we assume that $s \neq s_1$. Assume that M has an accepting $(1,2)$-computation path p_1 while reading $rz\#z^R sr'$. Replace a portion of this path associated with $z\#z^R s$ by a subpath corresponding to $z_1\#z_1^R s_1$. We then obtain a new accepting $(1,2)$-computation path on $rz_1\#z_1^R s_1 r'$. However, we obtain $(rz_1)^R = z_1^R r^R = z_1 sr' \neq z_1^R s_1 r'$ because $s \neq s_1$. This means that there is no accepting $(1,2)$-computation path on $rz_1\#z_1^R s_1 r'$, a contradiction. Therefore, the claim is true.

The above claim helps us define a map from (τ,σ) to s. Thus, the number of all possible strings s is at most $|\Gamma'_{(1,2)}|^2$. This implies that $|s|$ is upper-bounded by an appropriately chosen constant, independent of (n, x, y). □

In the second lemma, we want to show that the size of r' in (*) is also upper-bounded by a certain absolute constant.

Lemma 8. *There exists a constant $d_2 > 0$, independent of (n, x, y), that satisfies the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R sr'$, $y = r''z'$, $\ell = |x\#z^R s|$, $\ell' = |x\#x^R\#r''|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$, and $\gamma^{(x)}_{\ell',y} = v Z_0$. If $r'\#r'' \in TF_M(\sigma,\lambda)$, then $|r'| \leq d_2$ holds.*

Finally, we will prove Lemma 6 with the help of Lemmas 7 and 8.

Proof of Lemma 6. Let $x = rz$ and $x^R = z^R sr'$. Let $\gamma^{(x)}_y = uvZ_0$ and $\gamma^{(x)}_\ell = \sigma v Z_0$ with $\ell = |x\#z^R s|$. Assume that M transforms σ to u while reading r'. We first claim that $\ell \neq |x\#|$. Assume that $\ell = |x\#|$. This implies that $z = s = \lambda$. Hence, $x^R = r'$. By Lemma 8, we obtain $|r'| \leq d_2$. However, x must be sufficiently large in size, a contradiction. Therefore, $\ell \neq |x\#|$ holds.

Lemma 7 yields an appropriate constant d_1 such that $|s| \leq d_1$. Lemma 8 also shows that $|r'|$ is upper-bounded by a certain constant, say, d_2. Since $|r| = |sr'| = |s| + |r'|$ by definition, $|r|$ is bounded from above by $d_1 + d_2$. Let σ_0 be the stack symbol pushed into the stack at the first step of M. Since M transforms σ_0 to τv while reading r and the stack increases by at most one, it follows that $|v|$ (and therefore $|uvZ_0|$) is upper-bounded by an appropriately chosen constant. □

In the subsequent argument, the notation E_x expresses a collection of all stack contents $\gamma^{(x)}_y$ at the $|x\#x^R\#|$-th position (i.e., just after reading off $x\#x^R\#$)

along any accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$ for an *arbitrary* string $y \in H_x$. Since π is fixed, it holds that $1 \leq |E_x| \leq 2^{|x|} - 2$.

Before proceeding further, we want to prove a useful lemma.

Lemma 9. *Let $x_1, x_2, y \in \{0,1\}^n$. If $x_2 \in D_{(2,3)}$ and $x_1 \neq x_2$, then there is no position i such that $|x_1| \leq i \leq |x_1\#x_1^R\#|$ and $\gamma_{i,x_2}^{(x_1)} = \gamma_{i,y}^{(x_2)}$.*

Proof. Assume that such a position i actually exists. We then swap between substrings $x_1\#(x_1^R)_j$ and $x_2\#(x_2^R)_j$, where $j = |x_1\#x_1^R\#| - i$, and we then obtain another accepting $(1,2)$-computation path on input $x_2\#(x_2^R)_j(x_1^R)_{n-j}\#x_2$. (Case 1) If $(x_2^R)_j(x_1^R)_{n-j} \neq x_2^R$, then such an accepting path cannot be a $(1,2)$-computation path, a contradiction. (Case 2) If $(x_2^R)_j(x_1^R)_{n-j} = x_2^R$, then the obtained accepting $(1,2)$-computation path on $x_2\#x_2^R\#x_2$ must be a rejecting path by the choice of $x_2 \in D_{(2,3)}$, a contradiction. □

3.5 Size of Stack Contents

Notice that $|E_x| \geq 1$ holds for all $x \in D_{(2,3)}$. Prior to a discussion on this general case, we intend to consider a special case, which exemplifies an essence of our proof, where $|E_x| = 1$ holds for any $x \in D_{(2,3)}$.

I) Special Case of $|E_x| = 1$. Since the choice of $y \in H_x$ is irrelevant, it is possible to drop subscript "y" and express $\gamma_{i,y}^{(x)}$, $\gamma_y^{(x)}$, and $u_{x,y}$, as $\gamma_i^{(x)}$, $\gamma^{(x)}$, and u_x, respectively. To lead to the desired contradiction, let us examine two stack contents, $\gamma_{|x\#|}^{(x)}$ and $\gamma^{(x)}$.

Proposition 4. *Given any number $\epsilon \geq 0$, it holds that $|\{x \in D_{(2,3)} \mid \exists y \in H_x\, [|\gamma_y^{(x)}| \geq (n-2-\epsilon)/\log|\Gamma'_{(1,2)}|]\}| \geq |D_{(2,3)}|(1-2^{-\epsilon})$.*

To prove Proposition 4, let us consider two stack contents $\gamma_{x_2}^{(x_1)}$ and $\gamma_{x_1}^{(x_2)}$ for any distinct pair $x_1, x_2 \in D_{(2,3)}$. Lemma 9 implies that $\gamma_{x_2}^{(x_1)} \neq \gamma_{x_1}^{(x_2)}$. We thus obtain the following.

Lemma 10. *For every distinct pair $x_1, x_2 \in D_{(2,3)}$, it holds that $\gamma^{(x_1)} \neq \gamma^{(x_2)}$.*

Recall the set $\Gamma'_{(1,2)} = \Gamma_{(1,2)} \cup \{Z_0\}$. Given a number $d \in \mathbb{N}^+$, we further define $A_d = \{x \in D_{(2,3)} \mid \exists y \in H_x\, [|\gamma_y^{(x)}| \geq d]\}$.

Lemma 11. *For any constant $d \in \mathbb{N}^+$, it holds that $|A_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(1,2)}|^d$.*

Proof. Let $B_d = \{x \in D_{(2,3)} \mid \forall y \in H_x\, [|\gamma_y^{(x)}| < d]\}$. Notice that B_d coincides with $\{x \in D_{(2,3)} \mid |\gamma^{(x)}| < d\}$. It holds that $\gamma^{(x)}$ belongs to $(\Gamma'_{(1,2)})^m$ for a certain number m with $m \leq d-1$. Consider a mapping h from x to $\gamma^{(x)}$. The function h is 1-to-1 on the domain B_d of h by Lemma 10. Hence, it follows that

$|B_d|/2 \leq \sum_{j=1}^{d-1} |\Gamma'_{(1,2)}|^j = |\Gamma'_{(1,2)}|^d$. We conclude that, since $D_{(2,3)} = A_d \cup B_d$, $|A_d| = |D_{(2,3)}| - |B_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(2,3)}|^d$, as requested. □

With the help of Lemma 11, Proposition 4 can be easily proven as follows.

Proof of Proposition 4. For simplicity, write d for $(n-2-\epsilon)/\log|\Gamma'_{(1,2)}|$, which equals $\log_{|\Gamma'_{(1,2)}|} 2^{n-2-\epsilon}$. It suffices to show that $|A_d| \geq |D_{(2,3)}|(1-2^{-\epsilon})$. By Lemma 11, we obtain $|A_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(1,2)}|^d \geq |D_{(2,3)}|(1-2^{-\epsilon})$. □

To complete the proof for the special case, let $x = rz$, $x^R = z^R s r'$, $\gamma^{(x)} = uvZ_0$, and $\gamma_\ell^{(x)} = \sigma v Z_0$ with $\ell = |x\#z^R s|$. Assume that M transforms σ to u while reading r'. Proposition 3 shows that, for most of x's, $|uvZ_0|$ is upper-bounded by a certain constant, independent of (n, x, y). However, by setting, e.g., $\epsilon = 98$, Proposition 4 implies that $|uvZ_0| \geq (n-100)/\log|\Gamma'_{(1,2)}|$ for at least the two-thirds of x's in $D_{(2,3)}$. Since n is sufficiently large, we obtain a clear contradiction.

II) General Case of $|E_x| \geq 1$. We have already shown how to deal with the case where $|E_x| = 1$ holds for all $x \in D_{(2,3)}$. Now, let us discuss a general case where $|E_x| \geq 1$ holds for any $x \in D_{(2,3)}$. Our goal is to show the following statement, which replaces Proposition 4.

Proposition 5. *There are at least half of x's in $D_{(2,3)}$ such that, for a certain stack content $\tau \in E_x$, τ contains at least $\log_{|\Gamma|} n/2$ symbols.*

We start with the following lemma regarding E_x's, which can be seen as a generalization of Lemma 10.

Lemma 12. *Let $x_1, x_2 \in D_{(2,3)}$. If $x_2 \in H_{x_1}$, then $E_{x_1} \neq E_{x_2}$.*

Proof. Assume to the contrary that $E_{x_1} = E_{x_2}$ holds for two particular elements $x_1, x_2 \in D_{(2,3)}$ satisfying $x_2 \in H_{x_1}$. Take a stack content $\tau \in E_{x_1}$ satisfying $\tau = \gamma_{x_2}^{(x_1)}$ for a certain accepting $(1,2)$-computation path p_{x_1,x_2} of M on $x_1\#x_1^R\#x_2$. Since $E_{x_1} = E_{x_2}$, there exists another y in H_{x_2} that satisfies $\tau = \gamma_y^{(x_2)}$ along an appropriate accepting $(1,2)$-computation path $p_{x_2,y}$ on $x_2\#x_2^R\#y$. By swapping two parts of the above computation paths p_{x_1,x_2} and $p_{x_2,y}$ properly, we then obtain another accepting $(1,2)$-computation path of M on $x_2\#x_2^R\#x_2$ satisfying $\tau = \gamma_{x_2}^{(x_2)}$. This is an obvious contradiction against the choice of $x_2 \in D_{(2,3)}$. □

Write U_n for $\{x \in D_{(2,3)} \mid |E_x| > n/2\}$ and consider two separate cases.

Case 1: Assume that $|U_n| \geq |D_{(2,3)}|^{1/2}$. By taking an arbitrary $x \in U_n$, we want to claim that a certain stack content $\tau \in E_x$ must be made up of more than $\log_{|\Gamma|} n/2$ symbols. For this purpose, let us assume otherwise. Since any τ in E_x has at most $\log_{|\Gamma|} n/2$ symbols, there must be at most $n/2$ different elements in E_x. This implies that $x \notin U_n$, a contradiction against the choice of x. Hence, we obtain $|\tau| > \log_{|\Gamma|} n/2$, as stated in Proposition 5.

Case 2: Next, we assume that $|U_n| < |D_{(2,3)}|^{1/2}$. We first prove the following combinatorial lemma.

Lemma 13. *Let $n \in \mathbb{N}^+$ be sufficiently large and let X, Y satisfy $X \subseteq Y$. Let A be an $X \times Y$ matrix whose entries are taken from Θ^*, where Θ is an alphabet. Assume that (i) $|X| \geq 2^{n-2}$ and $|Y| = 2^n$, (ii) for any $(x, y) \in X \times Y$, $A_{(x,y)} = \lambda$ iff $y \in \{x, x^R\}$, and (iii) for any $x, y \in X$, if $A_{(x,y)} \neq \lambda$, then $A_{(x,y)} \neq A_{(y,z)}$ for any $z \in Y$. Then, the set $\tilde{X} = \{x \in X \mid \exists y \in Y\, [|A_{(x,y)}| \geq \log_{|\Theta|} \log_{|\Theta|} n\,]\}$ has cardinality at least $|X|/2$.*

Proof. Let us assume that the premise of the lemma is satisfied. For convenience, we define $X' = \{x \in X \mid \max_{y \in Y}\{|A_{(x,y)}|\} < \log_{|\Theta|} \log_{|\Theta|} n\}$, which satisfies $X = \tilde{X} \cup X'$. To show that $|\tilde{X}| \geq |X|/2$, we assume to the contrary that $|\tilde{X}| < |X|/2$. This implies that $|X'| = |X| - |\tilde{X}| > |X| - |X|/2 = |X|/2 \geq 2^{n-3}$ since $|X| \geq 2^{n-2}$. Let $E'_x = \{A_{(x,y)} \mid y \in Y\}$ for every $x \in X$. Analogously to Lemma 12, it holds that $E'_{x_1} \neq E'_{x_2}$ for every distinct pair $x_1, x_2 \in X$.

Let $x \in X'$. Since $|A_{(x,y)}| < \log_{|\Theta|} \log_{|\Theta|} n$ for all $y \in Y$, the total number of strings $A_{(x,y)}$ in E'_x is upper-bounded by $|\Theta|^{\log_{|\Theta|} \log_{|\Theta|} n} = \log_{|\Theta|} n$; that is, $|E'_x| \leq \log_{|\Theta|} n$. For convenience, let $\mathcal{E} = \bigcup_{x \in X'} E'_x$ and set $\alpha = |\mathcal{E}|$. Notice that $\alpha \geq 2$. Hereafter, we want to claim that $\alpha \geq 2^{(n-3)/(2 \log_{|\Theta|} n)}$. Toward a contradiction, we assume that $\alpha < 2^{(n-3)/(2 \log_{|\Theta|} n)}$. Now, let us estimate the upper bound of $|X'|$. Note that there are $|X'|$ different E'_x's in $\mathcal{E}$ and that E'_x is a subset of $\mathcal{E}$ of cardinality at most $\log_{|\Theta|} n$. It follows that $|X'|$ does not exceed the total number of $\mathcal{E}$'s nonempty subsets of size at most $\log_{|\Theta|} n$. We then conclude that $|X'| \leq \sum_{i=1}^{\log_{|\Theta|} n} \binom{\alpha}{i} \leq (\log_{|\Theta|} n) \cdot \alpha^{\log_{|\Theta|} n} \leq \alpha^{2 \log_{|\Theta|} n} \leq 2^{n-3}$, where the second inequality comes from $i < n/2$ and $\binom{\alpha}{i} \leq \alpha^i / i!$. This is a clear contradiction against $|X'| \geq 2^{n-3}$. Therefore, we obtain $\alpha \geq 2^{(n-3)/(2 \log_{|\Theta|} n)}$.

However, this contradicts the bound of $|\mathcal{E}| \leq \log_{|\Theta|} n$. □

Let us return to the proof of Proposition 5. To apply Lemma 13, we simply set X to be $D_{(2,3)}$, Y to be $\{0,1\}^n$, and $A_{(x,y)}$ to be $\tilde{\gamma}^{(x)}_{(}y)$, where $\tilde{\gamma}^{(x)}_y$ is obtained from $\gamma^{(x)}_y$ by simply deleting Z_0. It is not difficult to show that the obtained triplet (A, X, Y) satisfies Conditions (i)–(iii) of the lemma. The lemma ensures that there are at least half of x's in $D_{(2,3)}$ satisfying $|\gamma^{(x)}_y| \geq \log_{|\Theta|} \log_{|\Theta|} n$ for a certain string $y \in \{0,1\}^n$.

Nonetheless, Proposition 3 indicates that $|\gamma^{(x)}_y| \leq d_1$ for all $y \in H_x$. We then obtain a contradiction, as requested, and therefore this closes Case 1.

3.6 Case 2: $D_{(1,2)}$ Is Large

We have already proven Case 1 in Sections 3.3–3.5. To complete the proof of Proposition 2, however, we still need to examine the remaining case where $\{n \in \mathbb{N}^+ \mid |D_{(2,3)}| \geq 2^n/2\}$ is a finite set; in other words, $|D_{(1,2)}| > 2^n/2$ holds for all but finitely many n. Recall from Section 3.2 the introduction of our colored automaton $M = (Q, \Sigma, \{\not\vdash, \$\}, \Gamma, I_3, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with $Q_{acc} = \{q_{acc}\}$ and

$Q_{rej} = \{q_{rej}\}$ that computes g. Before starting the intended proof, we present a general lemma regarding inputs in reverse form.

Lemma 14. *There exists a colored automaton M^R that satisfies the following: M accepts $x_1\#x_2\#x_3$ along an accepting (i,j)-computation path if and only if M^R accepts $x_3^R\#x_2^R\#x_1^R$ along an accepting $(4-j,4-i)$-computation path.*

Let us return to our proof for the case of $|D_{(1,2)}| > 2^n/2$. Note that, by running M on inputs of the form $x\#y\#z$ for $x, y, z \in \{0,1\}^n$, we then obtain $|D_{(1,2)}| > 2^n/2$. We consider a counterpart of $D_{(1,2)}$, denoted by $D^R_{(2,3)}$, which is obtained by running M^R instead of M. Lemma 14 implies that $|D^R_{(2,3)}| > 2^n/2$. Apply to $D^R_{(2,3)}$ an argument used for Case 1. This is an obvious contradiction. We have therefore completed the proof of Proposition 2.

References

1. Choffrut, C., Culik, K.: Properties of finite and pushdown transducers. SIAM J. Comput. 12, 300–315 (1983)
2. Evey, R.J.: Application of pushdown-store machines. In: Proc. 1963 Fall Joint Computer Conference, pp. 215–227. AFIPS Press (1963)
3. Fisher, P.C.: On computability by certain classes of restricted Turing machines. In: Proc. 4th Annual IEEE Symp. on Switching Circuit Theory and Logical Design (SWCT 1963), pp. 23–32. IEEE Computer Society (1963)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley (2001)
5. Kobayashi, K.: Classification of formal langauges by functional binary transductions. Inform. Control 15, 95–109 (1969)
6. Konstantinidis, S., Santean, N., Yu, S.: Representation and uniformization of algebraic transductions. Acta Inform. 43, 395–417 (2007)
7. Selman, A.L.: A taxonomy of complexity classes of functions. J. Comput. System Sci. 48, 357–381 (1994)
8. Selman, A.L.: Much ado about functions. In: Proc. of the 11th Annual IEEE Conference on Computational Complexity, pp. 198–212 (1996)
9. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. Theoret. Comput. Sci. 411, 22–43 (2010)
10. Yamakami, T.: Swapping lemmas for regular and context-free languages (2008), Available at arXiv:0808.4122
11. Yamakami, T.: Pseudorandom generators against advised context-free languages (2009), See arXiv:0902.2774
12. Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci. 412, 6432–6450 (2011)
13. Yamakami, T.: Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 514–525. Springer, Heidelberg (2014)
14. Yamakami, T.: Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata (2014) (unpublished manuscript)

Characterizing Polynomial and Exponential Complexity Classes in Elementary Lambda-Calculus*

Patrick Baillot[1], Erika De Benedetti[1,2], and Simona Ronchi Della Rocca[2]

[1] CNRS, ENS de Lyon, INRIA, UCBL, Université de Lyon, LIP - Lyon, France
[2] Università degli Studi di Torino, Dipartimento di Informatica - Torino, Italy

Abstract. In this paper an implicit characterization of the complexity classes `k-EXP` and `k-FEXP`, for $k \geq 0$, is given, by a type assignment system for a stratified λ-calculus, where types for programs are witnesses of the corresponding complexity class. Types are formulae of Elementary Linear Logic (`ELL`), and the hierarchy of complexity classes `k-EXP` is characterized by a hierarchy of types.

Keywords: Implicit computational complexity, Linear logic, Lambda-calculus.

1 Introduction

Context. Early work on the study of complexity classes by means of programming languages has been carried out by Neil Jones [10,11], in particular using functional programming. The interest of these investigations is twofold: from the computational complexity point of view, they provide new characterizations of complexity classes, which abstract away from machine models; from the programming language point of view, they are a way to analyze the impact on complexity of various programming features (higher-order types, recursive definitions, read/write operations). This fits more generally in the research line of *implicit computational complexity* (ICC), whose goal is to study complexity classes without relying on explicit bounds on resources but instead by considering restrictions on programming languages and calculi. Seminal research in this direction has been carried out in the fields of recursion theory [4,13], λ-calculus [15] and linear logic [9]. These contributions usually exhibit a new specific language or logic for each complexity class, for instance `PTIME`, `PSPACE`, `LOGSPACE`: let us call *monovalent* the characterizations of this kind. We think however that the field would benefit from some more uniform presentations, which would consist in both a general language and a family of static criteria on programs of this language, each of which characterizing a particular complexity class. We call

* This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 151–163, 2014.

such a setting a *polyvalent* characterization; we believe that this approach is more promising for providing insights on the relationships between complexity classes. Polyvalent characterizations of this nature have been given in [11,14], but their criteria used for reaching point (2) referred to the construction steps of the programs. Here we are interested in defining a polyvalent characterization where (2) is expressed by means of the program's type in a dedicated system.

Stratification and Linear Logic. An ubiquitous notion in implicit complexity is that of *stratification*, by which we informally designate here the fact of organizing computation into distinct strata. This intuition underlies several systems: ramified and safe recursion [13,4], in which data is organized into strata; stratified comprehension [14], where strata are used for quantification; variants of linear logic [9] where programs are divided into strata thanks to a modality. More recently stratification of data has been related fruitfully to type systems for non-interference [18].

The linear logic approach to ICC is based on the proofs-as-programs correspondence. This logic indeed provides a powerful system to analyse the duplication and sharing of arguments in functional computation: this is made possible by a specific logical connective for the duplication of arguments, the ! modality. As in functional computation the reuse of an argument can cause a complexity explosion, the idea is to use weak versions of ! to characterize complexity classes. This intuition is illustrated by elementary linear logic (ELL) [9,8], a simple variant of linear logic which provides a monovalent characterisation of elementary complexity, that is to say computation in time bounded by a tower of exponentials of fixed height. Other variants of linear logic provide characterizations of PTIME, but they use either a more complicated language [9] or a more specific programming discipline [12].

Contribution and Comparison. In [2] a polyvalent characterization in ELL proof-nets of the complexity classes $\texttt{k-EXP} = \cup_{i\in\mathbb{N}}\texttt{DTIME}(2_k^{n^i})$ for all $k \geq 0$ has been obtained. However this approach has some shortcomings:

1. The complexity soundness proof uses a partly semantic argument ([2] Lemma 3 p. 10) and so it does not provide a syntactic way to evaluate the programs with the given complexity bound.
2. The characterization is given for classes of predicates, and not for classes of functions. Moreover it is not so clear how to extend this result to functions because of the semantic argument mentioned above.
3. The language of proof-nets is not as standard and widespread as say that of λ-calculus.

In the present work, we wish to establish an analogous polyvalent characterization in the setting of λ-calculus, with a stronger complexity soundness result based on a concrete evaluation procedure. We think this could provide a more solid basis to explore other characterizations of this kind.

In particular we define the $\lambda^!$-calculus, a variant of λ-calculus with explicit stratifications, which allows both to recover the results of [2] and to characterize also the function complexity classes k-FEXP, by two distinct hierarchies of

types. In fact, the characterization obtained through a standard representation of data-types like in [2] does not account for some closure properties of the function classes k-FEXP, in particular composition, so we propose a new, maybe less natural, representation in order to grasp these properties. Our language makes it easier to define such non-standard representation.

Technical Approach. One could expect that the results of [2] might be extended to $\lambda^!$-calculus by considering a translation of terms into proof-nets. However it is not so straightforward: term reduction cannot be directly simulated by the evaluation procedure in [2], because (i) it follows a specific cut-elimination strategy and (ii) ultimately it uses a semantic argument. For this reason we give here a direct proof of the result in $\lambda^!$-calculus, which requires defining new measures on terms and is not a mere adaptation of the proof-net argument.

Related Works. The first results on ELL [9,8] as well as later works [19,6] have been carried out in the setting of proof-nets. Other syntaxes have then been explored. First, specific term calculi corresponding to the related system LLL and to ELL have been proposed [22,17,16]. Alternatively [5] used standard λ-calculus with a type system derived from ELL. The $\lambda^!$-calculus we use here has a syntax similar to e.g. [21,7], and our type system is inspired by [5].

Outline. In the following we first introduce the $\lambda^!$-calculus as an untyped calculus, delineate a notion of well-formed terms and study the complexity of the reduction of these terms (Sect. 2). We then define a type system inspired by ELL and exhibit two families of types corresponding respectively to the hierarchies k-EXP and k-FEXP for $k \geq 0$ (Sect. 3). Finally we introduce a second characterization of this hierarchy, based on a non-standard data-type (Sect. 4). A conclusion follows.

A version of this work with a technical appendix containing detailed proofs is available as [3].

2 The $\lambda^!$-Calculus

2.1 Terms and Reduction

We use a calculus, $\lambda^!$-calculus, which adds to ordinary λ-calculus a ! modality and distinguishes two notions of λ-abstraction:

$$\mathtt{M}, \mathtt{N} ::= \mathtt{x} \mid \lambda\mathtt{x}.\mathtt{M} \mid \lambda^!\mathtt{x}.\mathtt{M} \mid \mathtt{MN} \mid !\mathtt{M}$$

where x ranges over a countable set of term variables Var. The usual notions of free variables is extended with $\mathrm{FV}(\lambda^!\mathtt{x}.\mathtt{M}) = \mathrm{FV}(\mathtt{M}) \setminus \{\mathtt{x}\}$, $\mathrm{FV}(!\mathtt{M}) = \mathrm{FV}(\mathtt{M})$. As usual, terms are considered modulo renaming of bound variables, and $=$ denotes the syntactic equality modulo this renaming.

Contexts. We consider the class of (one hole) contexts generated by the following grammar:

$$\mathcal{C} ::= \Box \mid \lambda \mathtt{x}.\mathcal{C} \mid \lambda^{!}\mathtt{x}.\mathcal{C} \mid \mathcal{C}\mathtt{M} \mid \mathtt{M}\mathcal{C} \mid !\mathcal{C}$$

As usual, capture of variables may occur. The *occurrence* of a term $\mathtt{N}$ in $\mathtt{M}$ is a context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]$; in practice we simply write $\mathtt{N}$ for the occurrence if there is no ambiguity and call it a subterm of $\mathtt{M}$.

Depth. The *depth of the occurrence* $\mathcal{C}$ in $\mathtt{M}$, denoted by $\delta(\mathcal{C}, \mathtt{M})$, is the number of ! modalities surrounding the hole of $\mathcal{C}$ in $\mathtt{M}$.

Moreover, the *depth* $\delta(\mathtt{M})$ of a term $\mathtt{M}$ is the maximal nesting of ! in $\mathtt{M}$.

Example 1. $\mathtt{M} = !((\lambda \mathtt{x}.\mathtt{x})\ !!\mathtt{y}\ !\mathtt{y})$. Then $\delta(!((\lambda \mathtt{x}.\mathtt{x})\ !!\Box\ !\mathtt{y}), \mathtt{M}) = 3$ and $\delta(!((\lambda \mathtt{x}.\mathtt{x})\ !!\mathtt{y}\ !\Box), \mathtt{M}) = 2$; moreover, $\delta(\mathtt{M}) = 3$.

Dynamics. The reduction $\rightarrow$ is the contextual closure of the following rules:

$$(\lambda \mathtt{x}.\mathtt{M})\mathtt{N} \longrightarrow \mathtt{M}[\mathtt{N}/\mathtt{x}] \quad (\beta\text{-rule}) \qquad (\lambda^{!}\mathtt{x}.\mathtt{M})!\mathtt{N} \longrightarrow \mathtt{M}[\mathtt{N}/\mathtt{x}] \quad (!\text{-rule})$$

where $[\mathtt{N}/\mathtt{x}]$ denotes the capture free substitution of $\mathtt{x}$ by $\mathtt{N}$, whose definition is the obvious extension of the corresponding one for λ-calculus. Observe that a term such as $(\lambda^{!}\mathtt{x}.\mathtt{M})\mathtt{P}$ is a redex only if $\mathtt{P} = !\mathtt{N}$ for some $\mathtt{N}$; the intuition behind these two kinds of redexes is that the abstraction λ expects an input at depth 0, while $\lambda^{!}$ expects an input at depth 1.

A *subterm at depth* i in $\mathtt{M}$ is an occurrence $\mathcal{C}$ in $\mathtt{M}$ such that $\delta(\mathcal{C}, \mathtt{M}) = i$; we denote by $\rightarrow_i$ the reduction of a redex occurring at depth i. As usual, $\stackrel{*}{\rightarrow}$ ($\stackrel{*}{\rightarrow_i}$) denotes the reflexive and transitive closure of $\rightarrow$ ($\rightarrow_i$). We say that a term is in *i-normal form* if it does not have any redex at depth less than or equal to i; then $\mathtt{M}$ is in normal form iff it is in $\delta(\mathtt{M})$-normal form. We denote as $\mathbf{nf}_i$ the set of terms in i-normal form.

We have a confluence property, whose proof is adapted from [20], taking into account the notion of depth:

Proposition 1.

(i) *Let* $\mathtt{M} \in \mathbf{nf}_i$ *and* $\mathtt{M} \rightarrow_j \mathtt{M}'$, *with* $j \geq i+1$, *then* $\mathtt{M}' \in \mathbf{nf}_i$.
(ii) [Confluence at fixed depth] *Let* $\mathtt{M} \rightarrow_i \mathtt{P}$ *and* $\mathtt{M} \rightarrow_i \mathtt{Q}$, *then there is a term* $\mathtt{N}$ *such that* $\mathtt{P} \stackrel{*}{\rightarrow_i} \mathtt{N}$ *and* $\mathtt{Q} \stackrel{*}{\rightarrow_i} \mathtt{N}$.
(iii) [Confluence] *Let* $\mathtt{M} \rightarrow \mathtt{P}$ *and* $\mathtt{M} \rightarrow \mathtt{Q}$, *then there is a term* $\mathtt{N}$ *such that* $\mathtt{P} \stackrel{*}{\rightarrow} \mathtt{N}$ *and* $\mathtt{Q} \stackrel{*}{\rightarrow} \mathtt{N}$.

We consider a specific subclass of terms, inspired by elementary linear logic (ELL) [9,17]:

Definition 1 (Well-formed Term). *A term* $\mathtt{M}$ *is well-formed (w.f.) if and only if, for any subterm* $\mathtt{N}$ *of* $\mathtt{M}$ *which is an abstraction, we have:*

1. *if* $\mathtt{N} = \lambda \mathtt{x}.\mathtt{P}$, *then* $\mathtt{x}$ *occurs at most once and at depth* 0 *in* $\mathtt{P}$;
2. *if* $\mathtt{N} = \lambda^{!}\mathtt{x}.\mathtt{P}$, *then* $\mathtt{x}$ *can only occur at depth* 1 *in* $\mathtt{P}$.

Example 2. $\lambda f.\lambda \mathtt{x}.f(f\mathtt{x})$, the standard representation of the Church integer 2, is not w.f.; its w.f. counterpart is $\lambda^! f.!(\lambda \mathtt{x}.f(f\mathtt{x}))$.

The motivation behind such definition is that the depth of subterms in a w.f. term does not change during reduction: if an abstraction expects an input at depth 0 (resp. 1), which is the case of λ (resp. $\lambda^!$), then the substitutions occur at depth 0 (resp. 1), as each occurrence of its bound variable is at depth 0 (resp. 1).

The class of w.f. terms is preserved by reduction and their depth does not increase during reduction:

Lemma 1. *If* $\mathtt{M}$ *is w.f. and* $\mathtt{M} \to \mathtt{M}'$*, then* $\mathtt{M}'$ *is w.f., and* $\delta(\mathtt{M}') \leq \delta(\mathtt{M})$.

From now on, we assume that all terms are well formed.

Sizes. In order to study the reduction, it is useful to examine the *size of* $\mathtt{M}$ *at depth* i, denoted by $|\mathtt{M}|_i$, defined as follows:

- If $\mathtt{M} = \mathtt{x}$, then $|\mathtt{x}|_0 = 1$ and $|\mathtt{x}|_i = 0$ for $i \geq 1$;
- If $\mathtt{M} = \lambda \mathtt{x}.\mathtt{N}$ or $\mathtt{M} = \lambda^! \mathtt{x}.\mathtt{N}$, then $|\mathtt{M}|_0 = |\mathtt{N}|_0 + 1$ and $|\mathtt{M}|_i = |\mathtt{N}|_i$ for $i \geq 1$;
- If $\mathtt{M} = \mathtt{NP}$, then $|\mathtt{M}|_0 = |\mathtt{N}|_0 + |\mathtt{P}|_0 + 1$ and $|\mathtt{M}|_i = |\mathtt{N}|_i + |\mathtt{P}|_i$ for $i \geq 1$;
- If $\mathtt{M} = !\mathtt{N}$, then $|\mathtt{M}|_0 = 0$ and $|\mathtt{M}|_{i+1} = |\mathtt{N}|_i$ for $i \geq 0$;

Let $\delta(\mathtt{M}) = d$; then $|\mathtt{M}|_{i+} = \sum_{j=i}^{d} |\mathtt{M}|_j$ and the *size of* $\mathtt{M}$ is $|\mathtt{M}| = \sum_{i=0}^{d} |\mathtt{M}|_i$. The definition is extended to contexts, where $|\Box|_i = 0$ for $i \geq 0$. We consider how the size of a term changes during reduction:

Lemma 2. *If* $\mathtt{M} \to_i \mathtt{M}'$*, then* $|\mathtt{M}'|_i \leq |\mathtt{M}|_i - 1$*, and* $|\mathtt{M}'|_j = |\mathtt{M}|_j$ *for* $j < i$.

Strategy. The fact that by Prop. 1.(i) reducing a redex does not create any redex at strictly lower depth suggests considering the following, non-deterministic, *level-by-level* reduction strategy: if the term is not in normal form reduce (non deterministically) a redex at depth i, where $i \geq 0$ is the minimal depth such that $\mathtt{M} \notin \mathbf{nf}_i$. A *level-by-level reduction sequence* is a reduction sequence following the level-by-level strategy. We say that a reduction sequence is *maximal* if either it is infinite, or if it finishes with a normal term.

Proposition 2. *Any reduction of a term by the level-by-level strategy terminates.*

It follows that a maximal level-by-level reduction sequence of a term $\mathtt{M}$ has the shape shown in (1), where $\rightsquigarrow_i$ denotes one reduction step according to the level-by-level strategy, performed at depth i. We call *round* i the subsequence of $\rightsquigarrow_i$ starting from $\mathtt{M}_i^1$. Note that, for all i and $j > i$, $\mathtt{M}_j^1 \in \mathbf{nf}_i$. We simply write $\rightsquigarrow$ when we do not refer to a particular depth.

$$\mathtt{M}_0^1 \rightsquigarrow_0 \ldots \rightsquigarrow_0 \mathtt{M}_0^{n_0} = \mathtt{M}_1^1 \rightsquigarrow_1 \ldots \rightsquigarrow_1 \mathtt{M}_1^{n_1} = \ldots = \mathtt{M}_\delta^1 \rightsquigarrow_\delta \ldots \rightsquigarrow_\delta \mathtt{M}_\delta^{n_\delta} \tag{1}$$

In a particular case, namely in Lemma 3, we use a deterministic version of the level-by-level strategy, called *leftmost-by-level*, which proceeds at every level

from left to right, taking into account the shape of the different redexes in our calculus. That is to say, it chooses at every step the leftmost subterm of the shape $\mathtt{MN}$, where $\mathtt{M}$ is an abstraction, and, in case it is already a redex it reduces it, in case it is of the shape $(\lambda^!\mathtt{x}.\mathtt{P})\mathtt{N}$, where $\mathtt{N} \neq !\mathtt{Q}$, for some $\mathtt{Q}$, then it looks for the next redex in $\mathtt{N}$. This corresponds to using the call-by-name discipline for β-redexes and the call-by-value for !-redexes [20].

$\mathtt{M} \Longrightarrow \mathtt{N}$ denotes that $\mathtt{N}$ is obtained from $\mathtt{M}$ by performing one reduction step according to the leftmost-by-level strategy. All the notations for $\to$ are extended to $\rightsquigarrow$ and $\Longrightarrow$ in a straightforward way.

2.2 Representation of Functions

In order to represent functions, we first need to encode data. For booleans we can use the familiar encoding $\mathtt{true} = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x}$ and $\mathtt{false} = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y}$. For tally integers, the usual encoding of Church integers does not give w.f. terms; instead, we use the following encodings for Church integers and Church binary words:

$$n \in \mathbb{N}, \qquad \underline{n} = \lambda^! f.!(\lambda\mathtt{x}.f\ (f\ ...(f\ \mathtt{x})...))$$
$$w \in \{0,1\}^\star, \quad w = \langle i_1, ..., i_n \rangle, \quad \underline{w} = \lambda^! f_0.\lambda^! f_1.!(\lambda\mathtt{x}.f_{i_1}\ (f_{i_2}\ ...(f_{i_n}\ \mathtt{x})...))$$

By abuse of notation we also denote by $\underline{1}$ the term $\lambda^! f.!f$. Observe that the terms encoding booleans are of depth 0, while those representing Church integers and Church binary words are of depth 1. We denote the length of a word $w \in \{0,1\}^\star$ by $\mathbf{length}(w)$.

We represent computation on a binary word by considering applications of the form $\mathtt{P}!\underline{w}$, with a ! modality on the argument, because the program should be able to duplicate its input. Concerning the form of the result, since we want to allow computation at arbitrary depth, we require the output to be of the form $!^k\mathtt{D}$, where $k \in \mathbb{N}$ and $\mathtt{D}$ is one of the data representations above.

We thus say that a function $f : \{0,1\}^\star \to \{\mathtt{true}, \mathtt{false}\}$ is represented by a term (*program*) $\mathtt{P}$ if $\mathtt{P}$ is a closed normal term and there exists $k \in \mathbb{N}$ such that, for any $w \in \{0,1\}^\star$ and $\mathtt{D} = f(w) \in \{\mathtt{true}, \mathtt{false}\}$ we have: $\mathtt{P}!\underline{w} \overset{*}{\to} !^k\mathtt{D}$. This definition can be adapted to functions with other domains and codomains.

2.3 Complexity of Reduction

We study the complexity of the reduction of terms of the form $\mathtt{P}!\underline{w}$. Actually it is useful to analyze the complexity of the reduction of such terms to their k-normal form, i.e. by reducing until depth k, for $k \in \mathbb{N}$. We define the notation 2_i^n in the following way: $2_0^x = x$ and $2_{i+1}^x = 2^{2_i^x}$.

Proposition 3. *Given a program* $\mathtt{P}$*, for any* $k \geq 2$*, there exists a polynomial* q *such that, for any* $w \in \{0,1\}^\star$*,* $\mathtt{P}!\underline{w} \overset{*}{\rightsquigarrow} \mathtt{M}_k^1 \in \mathbf{nf}_{k-1}$ *in at most* $2_{k-2}^{q(n)}$ *steps, and* $|\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$*, where* $n = \mathbf{length}(w)$*. In particular, in the case where* $k = 2$ *we have a polynomial bound* $q(n)$*.*

In the rest of this section we prove Prop. 3.

Let $\mathtt{M} = \mathtt{P}!\underline{w}$ and consider a level-by-level reduction sequence of $\mathtt{M}$, using the notations of (1). By Lemma 2 we know that the number of steps at depth i is bounded by $|\mathtt{M}_i^1|$ and that there are $(d+1)$ rounds. In order to bound the total number of steps it is thus sufficient to bound $|\mathtt{M}_i^1|$ by means of $|\mathtt{M}|$:

Lemma 3 (Size-Growth). *If* $\mathtt{M} \overset{*}{\Longrightarrow}_i \mathtt{M}'$ *by* c *reduction steps, then* $|\mathtt{M}'| \leq |\mathtt{M}| \cdot (|\mathtt{M}|+1)^c$ $(0 \leq i \leq \delta(\mathtt{M}))$.

Proof (Prop. 3). We proceed by induction on $k \geq 2$. We assume that $\mathtt{P}$ is of the form $\lambda^!\mathtt{y}.\mathtt{Q}$ (otherwise $\mathtt{P}!\underline{w}$ is already a normal form).

- Case $k = 2$:
  We consider a level-by-level reduction sequence of $\mathtt{P}!\underline{w}$. We need to examine reduction at depths 0 and 1. At depth 0 we have $(\lambda^!\mathtt{y}.\mathtt{Q})!\underline{w} \to \mathtt{Q}[\underline{w}/\mathtt{y}] = \mathtt{M}_1^1$. Observe that $\mathtt{M}_1^1 \in \mathbf{nf}_0$ because the occurrences of $\mathtt{y}$ in $\mathtt{Q}$ are at depth 1; denote by b the number of occurrences of $\mathtt{y}$ in $\mathtt{Q}$, which does not depend on n.

  Since $|\mathtt{Q}[\underline{w}/\mathtt{y}]|_1 \leq |\mathtt{Q}|_1 + b \cdot |\underline{w}|_0$ and $|\underline{w}|_0 = 2$ (by definition of the encoding), we have that $|\mathtt{M}_1^1|_1 = |\mathtt{Q}[\underline{w}/\mathtt{y}]|_1 \leq |\mathtt{Q}|_1 + 2b$. Let c be $|\mathtt{Q}|_1 + 2b$, which does not depend on n: then, by Lemma 2, the number of steps at depth 1 is bounded by c. This proves the first part of the statement.

  Let $\mathtt{M}_2^1 \in \mathbf{nf}_1$ be the term obtained after reduction at depth 1. By Prop. 1.(ii) we have that $\mathtt{M}_1^1 \overset{*}{\Longrightarrow}_1 \mathtt{M}_2^1$ and by Lemma 2 this reduction is done in c' steps, where $c' \leq |\mathtt{M}_1^1|_1 \leq c$, so by Lemma 3 we have that $|\mathtt{M}_2^1| \leq |\mathtt{M}_1^1| \cdot (|\mathtt{M}_1^1|+1)^c$. Moreover $|\mathtt{M}_1^1| \leq |\mathtt{Q}| + b|\underline{w}|$, so it is polynomial in n, and the statement is proved for $k = 2$.
- Assume the property holds for k and let us prove it for $k+1$.
  By assumption $\mathtt{M}$ reduces to $\mathtt{M}_k^1$ in at most $2_{k-2}^{q(n)}$ steps and $|\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$. Let $\mathtt{M}_k^1 \overset{*}{\rightsquigarrow}_k \mathtt{M}_{k+1}^1 \in \mathbf{nf}_k$. By Lemma 2 this reduction sequence has at most $|\mathtt{M}_k^1|_k$ steps, and $|\mathtt{M}_k^1|_k \leq |\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$. So on the whole $\mathtt{M}$ reduces to $\mathtt{M}_{k+1}^1$ in at most $2 \cdot 2_{k-2}^{q(n)} \leq 2_{k-2}^{2q(n)}$ steps. Moreover by Prop. 1.(ii) we have that $\mathtt{M}_k^1 \overset{*}{\Longrightarrow} \mathtt{M}_{k+1}^1$ and by Lemma 2 and Lemma 3 we get

  $$|\mathtt{M}_{k+1}^1| \leq |\mathtt{M}_k^1| \cdot (|\mathtt{M}_k^1|+1)^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot (2_{k-2}^{2q(n)})^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot 2^{2_{k-2}^{3q(n)}} \leq 2_{k-1}^{q'(n)}$$

  for some polynomial $q'(n)$.

Approximations. From Prop. 3 we can easily derive a $2_{k-2}^{q(n)}$ bound on the number of *steps* of the reduction of $\mathtt{P}!\underline{w}$ not only to its $(k-1)$-normal form, but also to its k-normal form $\mathtt{M}_{k+1}^1$. Unfortunately this does not yield directly a *time* bound $O(2_{k-2}^{q(n)})$ for the simulation of this reduction on a Turing machine, because during round k the size of the term at depth $k+1$ could grow exponentially. However if we are only interested in the result at depth k, the subterms at depth $k+1$ are actually irrelevant. For this reason we introduce a notion of *approximation*,

inspired by the semantics of stratified coherence spaces [1], which allows us to compute up to a certain depth k, while ignoring what happens at depth $k+1$.

We extend the calculus with a constant $*$; its sizes $|*|_i$ are defined as for variables. If $\mathtt{M}$ is a term and $i \in \mathbb{N}$, we define its *i-th approximation* $\overline{\mathtt{M}}^i$ by: $\overline{!\mathtt{M}}^0 = !*$, $\overline{!\mathtt{M}}^{i+1} = !\,\overline{\mathtt{M}}^i$, $\overline{\mathtt{x}}^i = \mathtt{x}$, and for all other constructions $\overline{(\cdot)}^i$ acts as identity, e.g. $\overline{\mathtt{MN}}^i = \overline{\mathtt{M}}^i\overline{\mathtt{N}}^i$.

So $\overline{\mathtt{M}}^i$ is obtained by replacing in $\mathtt{M}$ all subterms at depth $i+1$ by $*$. For instance we have $\overline{\underline{w}}^0 = \lambda^! f_0.\lambda^! f_1.!*$ and $\overline{\underline{w}}^{i+1} = \underline{w}$ for $i \geq 0$.

Lemma 4. *(i) Let $\mathtt{M} \to_j \mathtt{M}'$: if $j \leq i$ then $\overline{\mathtt{M}}^i \to_j \overline{\mathtt{M}'}^i$, otherwise $\overline{\mathtt{M}}^i = \overline{\mathtt{M}'}^i$.*
(ii) Let $\overline{\mathtt{M}}^i \to_i \overline{\mathtt{M}'}^i$: then $|\overline{\mathtt{M}'}^i| < |\overline{\mathtt{M}}^i|$.

Proposition 4. *Given a program $\mathtt{P}$, for any $k \geq 2$, there exists a polynomial q such that for any $w \in \{0,1\}^\star$, the reduction of $\overline{\mathtt{P}!\underline{w}}^k$ to its k-normal form can be computed in time $O(2_{k-2}^{q(n)})$ on a Turing machine, where $n = \mathbf{length}(w)$.*

Proof. Observe that $\overline{\mathtt{P}!\underline{w}}^k = \overline{\mathtt{P}}^k!\underline{w}$. By Prop. 3 and Lemma 4.(i), it reduces to its $(k-1)$-normal form $\overline{\mathtt{M}_k^!}^k$ in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$. Now by Lemma 4.(ii) the reduction of $\overline{\mathtt{M}_k^!}^k$ at depth k is done in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$. We can then conclude by using the fact that one reduction step in a term $\mathtt{M}$ can be simulated in time $p(|\mathtt{M}|)$ on a Turing machine, for a suitably chosen polynomial p.

3 Type System

We introduce a type assignment system for $\lambda^!$-calculus, based on ELL, such that all typed terms are also w.f. and the previous results are preserved.

The set $\mathcal{T}$ of types are generated by the grammar

$$\begin{array}{lll} \mathtt{A} ::= \mathtt{a} \mid \mathtt{S} & & \text{(linear types)} \\ \mathtt{S} ::= \sigma \multimap \sigma \mid \forall \mathtt{a}.\mathtt{S} \mid \mu\mathtt{a}.\mathtt{S} & & \text{(strict linear types)} \\ \sigma ::= \mathtt{A} \mid !\sigma & & \text{(types)} \end{array}$$

where $\mathtt{a}$ ranges over a countable set of type variables. Observe that we consider both polymorphic types ($\forall\mathtt{a}.\mathtt{S}$) and type fixpoints ($\mu\mathtt{a}.\mathtt{S}$); the restriction of both abstractions to act on strict linear types is necessary for the subject reduction property.

A *basis* is a partial function from variables to types, with finite domain; given two bases Γ_1 and Γ_2, let $\Gamma_1 \# \Gamma_2$ iff $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$. Following the work of [5], we consider three different bases $\Gamma \mid \Delta \mid \Theta$, called respectively the *linear*, *modal* and *parking* basis, such that $\Gamma \# \Delta$, $\Gamma \# \Theta$ and $\Delta \# \Theta$. The premises in Γ assign to variables linear types, while the premises in Δ assign modal types.

The typing system proves statements of the shape $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$, and derivations are denoted by Π, Σ. The rules are given in Table 1. Observe that,

Table 1. Derivation rules

$$\frac{}{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathtt{x} : \mathtt{A}}\ (Ax^L) \qquad \frac{}{\Gamma \mid \Delta \mid \mathtt{x} : \sigma, \Theta \vdash \mathtt{x} : \sigma}\ (Ax^P)$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \tau}\ (\multimap I^L) \qquad \frac{\Gamma \mid \Delta, \mathtt{x} :! \sigma \mid \Theta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! \mathtt{x}.\mathtt{M} :! \sigma \multimap \tau}\ (\multimap I^I)$$

$$\frac{\Gamma_1 \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma \multimap \tau \quad \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{N} : \sigma \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{MN} : \tau}\ (\multimap E) \qquad \frac{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{M} : \sigma}{\Gamma \mid !\Theta', \Delta \mid \Theta \vdash !\mathtt{M} :! \sigma}\ (!)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S} \quad \mathtt{a} \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(\Delta) \cup \mathrm{FTV}(\Theta)}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{S}}\ (\forall I) \qquad \frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\sigma/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\mu \mathtt{a}.\mathtt{S}/\mathtt{a}]}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mu \mathtt{a}.\mathtt{S}}\ (\mu I) \qquad \frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mu \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\mu \mathtt{a}.\mathtt{S}/\mathtt{a}]}\ (\mu E)$$

in rule $(\multimap E)$, $\mathtt{M}$ and $\mathtt{N}$ share variables in the modal and parking basis, but their linear bases must be disjoint. Note also that there is no axiom rule for variables in the modal basis, so the only way to introduce a variable in this basis is the (!) rule, moving variables from the parking to the modal basis. Finally, observe that there is no abstraction rule for variables in the parking basis: indeed parking variables only have a "temporary" status, awaiting to be moved to the modal basis.

We say that a term $\mathtt{M}$ is *well-typed* iff there is a derivation $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathtt{M} : \sigma$ for some Γ, Δ, σ: indeed parking variables are only considered as an intermediary status before becoming modal variables. When all three bases are empty we denote the derivation by $\Pi \triangleright \vdash \mathtt{M} : \sigma$. The main difference w.r.t. the type system of [5] is the (!) rule: here we allow only the parking context to be non-empty, in order to ensure that typable terms are well formed: it is the key to obtain a $2_k^{\mathrm{poly}(n)}$ complexity bound for a specific k depending on the type, instead of just an elementary bound.

Both the type and depth of a term are preserved during reduction:

Theorem 1 (Subject Reduction). *$\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \to \mathtt{M}'$ imply $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}' : \sigma$.*

Proposition 5. *If a term is well-typed, then it is also well-formed.*

The proof comes easily from the following proposition:

Proposition 6 (Variables Depth). *Let $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$. Then:*

- *if $\mathtt{x} \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Theta)$, then $\mathtt{x}$ can only occur at depth 0 in $\mathtt{M}$;*
- *if $\mathtt{x} \in \mathrm{dom}(\Delta)$, then $\mathtt{x}$ can only occur at depth 1 in $\mathtt{M}$.*

3.1 Datatypes

In section 2.2 we introduced w.f. terms encoding data, for which we now define the following types, adapted from system F, representing respectively booleans, Church tally integers and Church binary words:

$$\mathrm{B} = \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a} \qquad \mathrm{N} = \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})$$

$$\mathrm{W} = \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})$$

We also use Scott binary words, defined inductively as

$$\widehat{\epsilon} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.\mathtt{x} \qquad \widehat{0w} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.f_0\widehat{w} \qquad \widehat{1w} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.f_1\widehat{w}$$

having type $\mathrm{W}_S \stackrel{\text{def}}{=} \mu\mathtt{b}.\forall\mathtt{a}.(\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})$.

The following properties ensure that, given a datatype, every derivation having such type reduces to a term having the desired shape:

Proposition 7. *(i) If* $\vdash \mathtt{M} :!^k\mathrm{B}$ *for* $k \geq 0$ *and* $\mathtt{M} \in \mathbf{nf}_k$*, then either* $\mathtt{M} =!^k\mathtt{true}$ *or* $\mathtt{M} =!^k\mathtt{false}$*.*

(ii) If $\vdash \mathtt{M} :!^k\mathrm{W}_S$ *for* $k \geq 0$ *and* $\mathtt{M} \in \mathbf{nf}_k$*, then* $\mathtt{M} =!^k\widehat{w}$ *for some* $\widehat{w}$*.*

3.2 Complexity Soundness and Completeness

We are interested in giving a precise account of the hierarchy of classes characterized by this typed $\lambda^!$-calculus. Denote by $\mathtt{FDTIME}(F(n))$ and by $\mathtt{DTIME}(F(n))$ respectively the class of functions and the class of predicates on binary words computable on a deterministic Turing machine in time $O(F(n))$; the complexity classes we are interested in, for $k \geq 0$, are:

$$\mathtt{k\text{-}EXP} = \cup_{i\in\mathbb{N}}\mathtt{DTIME}(2_k^{n^i}) \quad \text{and} \quad \mathtt{k\text{-}FEXP} = \cup_{i\in\mathbb{N}}\mathtt{FDTIME}(2_k^{n^i}).$$

In particular, observe that $\mathtt{PTIME} = \cup_{i\in\mathbb{N}}\mathtt{DTIME}(n^i) = \mathtt{0\text{-}EXP}$ and $\mathtt{FPTIME} = \cup_{i\in\mathbb{N}}\mathtt{FDTIME}(n^i) = \mathtt{0\text{-}FEXP}$.

Soundness. Let $\mathcal{F}(\sigma)$ denote the set of closed terms representing functions, to which type σ can be assigned: we prove that $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq k\text{-}\mathtt{EXP}$ and $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S) \subseteq k\text{-}\mathtt{FEXP}$.

Theorem 2 (Soundness). *Let* $\vdash \mathtt{P} :!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ *where* $\mathtt{P}$ *is a program, and let* $\vdash \underline{w} : \mathrm{W}$ *where* $\mathbf{length}(w) = n$*; then the reduction* $\mathtt{P}!\underline{w} \stackrel{*}{\to} !^{k+2}\mathtt{D}$ *can be computed in time* $2_k^{p(n)}$*, where* $\mathtt{D}$ *is either* `true` *or* `false` *and* p *is a polynomial.*

Proof. Recall that a program $\mathtt{P}$ is a typed closed term in normal form: we denote by $\mathtt{M}'$ the normal form of $\mathtt{P}!\underline{w}$. By Prop. 4 we know that $\overline{\mathtt{P}!\underline{w}}^{k+2}$ can be reduced to a term $\mathtt{N}$ in $\mathbf{nf}_{k+2}$ in time $O(2_k^{p(n)})$ on a Turing machine, where $n = \mathbf{length}(w)$. Moreover by Lemma 4.(i) and Prop. 1.(iii) we have that $\overline{\mathtt{M}'}^{k+2} = \mathtt{N}$. Now, as $\mathtt{P}!\underline{w}$ has type $!^{k+2}\mathrm{B}$, by Theorem 1 the term $\mathtt{M}'$ is a closed term of type $!^{k+2}\mathrm{B}$ and, by Prop. 7.(i), it is equal to $!^{k+2}\mathtt{true}$ or $!^{k+2}\mathtt{false}$. Then $\mathtt{N} = \overline{\mathtt{M}'}^{k+2} = \mathtt{M}'$, so $\mathtt{P}!\underline{w}$ can be computed in time $O(2_k^{p(n)})$.

Complexity soundness can be proved for functions by a similar proof, in which Prop. 7.(ii) is used in order to read the output as a Scott word:

Theorem 3. *Let* $\vdash \mathtt{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$ *where* $\mathtt{P}$ *is a program, and let* $\vdash \underline{w} : \mathrm{W}$ *where* $\mathbf{length}(w) = n$*; then the reduction* $\mathtt{P}!\underline{w} \xrightarrow{*} !^{k+2}\widehat{w'}$ *can be computed in time* $2_k^{p(n)}$*, where* p *is a polynomial.*

Completeness. We proved that $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq k\text{-}\mathtt{EXP}$ and $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S) \subseteq k\text{-}\mathtt{FEXP}$; now we want to strengthen this result by examining the converse inclusions. To do so we simulate k-$\mathtt{EXP}$ time bounded Turing machines, by an iteration, so as to prove the following results:

Theorem 4 (Extensional Completeness).

- *Let* f *be a binary predicate in* k-$\mathtt{EXP}$*, for any* $k \geq 0$*; then there is a term* $\mathtt{M}$ *representing* f *such that* $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$.
- *Let* g *be a function on binary words in* k-$\mathtt{FEXP}$*, for* $k \geq 0$*; then there is a term* $\mathtt{M}$ *representing* g *such that* $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$.

Note that this characterization, for $k = 0$, does not account for the fact that $\mathtt{FPTIME}$ is closed by composition: indeed, programs of type $!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$ cannot be composed, since we do not have any coercion from W_S to W. For this reason, we explore an alternative characterization.

4 Refining Types for an Alternative Characterization

Our aim is to take a pair $\langle n, w\rangle$ to represent the word w' such that:

$$w' = \begin{cases} w & \text{if } \mathbf{length}(w) \leq n, \\ \text{the prefix of } w \text{ of length } n & \text{otherwise.} \end{cases}$$

For this reason, we introduce a new data-type using the connective $\otimes$ defined by $\sigma \otimes \tau \stackrel{\text{def}}{=} \forall \mathtt{a}.((\sigma \multimap \tau \multimap \mathtt{a}) \multimap \mathtt{a})$ on types and the corresponding constructions on terms:

$$\begin{aligned} \mathtt{M}_1 \otimes \mathtt{M}_2 &\stackrel{\text{def}}{=} \lambda \mathtt{x}.\mathtt{x}\mathtt{M}_1\mathtt{M}_2 \\ \lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} &\stackrel{\text{def}}{=} \lambda \mathtt{x}.(\mathtt{x}\lambda \mathtt{y}_1\mathtt{y}_2.\lambda \mathtt{z}.\mathtt{z}\mathtt{y}_1\mathtt{y}_2)\lambda \mathtt{x}_1\mathtt{x}_2.\mathtt{M} \\ \lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} &\stackrel{\text{def}}{=} \lambda \mathtt{x}.(\mathtt{x}\lambda^! \mathtt{y}_1\mathtt{y}_2.\lambda \mathtt{z}.\mathtt{z}!\mathtt{y}_1!\mathtt{y}_2)\lambda^! \mathtt{x}_1\mathtt{x}_2.\mathtt{M} \end{aligned}$$

Note that we cannot define the abstraction in the usual way, i.e. $\lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} \stackrel{\text{def}}{=} \lambda \mathtt{x}.\mathtt{x}(\lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{M})$, otherwise we could not type pairs in a uniform way; moreover, when applied to a pair, this term reduces to the usual one.

The associated reduction rules $(\lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(\mathtt{M}_1 \otimes \mathtt{M}_2) \to \mathtt{N}[\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2]$ and $(\lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(!\mathtt{M}_1 \otimes !\mathtt{M}_2) \to \mathtt{N}[\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2]$ are derivable.

We represent a pair $\langle n, w\rangle$ through a term $!\underline{n} \otimes !^2\widehat{w}$ of type $!\mathrm{N} \otimes !^2\mathrm{W}_S$, i.e. a combined data-type containing a Church integer $!\underline{n}$ and a Scott word $!^2\widehat{w}$: in

practice, $\underline{n}$ is meant to represent the length of a list, whose content is described by $\widehat{w}$. In order to mantain this invariant, when computing on elements $!\underline{n} \otimes !^2\widehat{w}$ of this data-type, the property that the length of w is inferior or equal to n is preserved.

As before, we need to be able to extract the result, in this case a pair:

Proposition 8. *If* $\vdash \mathtt{M} : !^k \mathrm{N} \otimes !^{k+1} \mathrm{W}_S$ *for* $k \geq 0$ *and* $\mathtt{M} \in \mathbf{nf}_{k+1}$*, then there exists* $m \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathtt{M} =_! !^k \underline{m} \otimes !^{k+1} \widehat{w}$*.*

Then we are able to prove both soundness and completeness results:

Theorem 5. *Let* $\vdash \mathtt{P} : (!\mathrm{N} \otimes !^2 \mathrm{W}_S) \multimap (!^{k+1} \mathrm{N} \otimes !^{k+2} \mathrm{W}_S)$ *where* $\mathtt{P}$ *is a program, then for any* $\underline{m}$ *and* $\widehat{w}$ *the reduction of* $\mathtt{P}(!\underline{m} \otimes !^2 \widehat{w})$ *to its normal form can be computed in time* $2_k^{p(n)}$*, where* p *is a polynomial and* $n = m + \mathbf{length}(w)$*.*

Theorem 6. *Let* f *be a function on binary words in* k-FEXP*, for* $k \geq 0$*; then there is a term* $\mathtt{M}$ *representing* f *such that* $\vdash \mathtt{M} : (!\mathrm{N} \otimes !^2 \mathrm{W}_S) \multimap (!^{k+1} \mathrm{N} \otimes !^{k+2} \mathrm{W}_S)$*.*

Observe that we are able to compose two terms having type $(!\mathrm{N} \otimes !^2 \mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2 \mathrm{W}_S)$, so to illustrate the fact that FPTIME is closed by composition; moreover, if $f \in$ FPTIME and $g \in k$-FEXP, then we can compose terms representing them, which shows that $g \circ f \in k$-FEXP.

While the previous characterization of k-FEXP in Section 3.2 offers the advantage of simplicity, because it uses classical data-types (Church and Scott binary words), this second characterization offers a better account of the closure properties of these complexity classes, at the price of a slightly more involved representation of words.

5 Conclusions

We have shown how the concept of !-stratification coming from linear logic can be fruitfully employed in λ-calculus and characterize the hierarchies k-EXP and k-FEXP, including the classes PTIME and FPTIME. A nice aspect of our system with respect to former polyvalent characterizations [11,14] is that the complexity bound can be deduced by looking only at the interface of the program (its type) without refering to the constructions steps. In our proofs we have carefully distinguished the respective roles played by syntactic ingredients (well-formedness) and typing ingredients. This has allowed us to illustrate how types can provide two different characterizations of the class k-FEXP, based on the use of different data-types. We believe that the separation between syntactic and typing arguments can facilitate the possible future usage of our calculus with other type systems. As future work it would be challenging to investigate if similar characterizations could be obtained for other hierarchies, like possibly space hierarchies.

References

1. Baillot, P.: Stratified coherence spaces: a denotational semantics for light linear logic. Theor. Comput. Sci. 318(1-2), 29–55 (2004)
2. Baillot, P.: Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In: Yang, H. (ed.) APLAS 2011. LNCS, vol. 7078, pp. 337–352. Springer, Heidelberg (2011)
3. Baillot, P., De Benedetti, E., Ronchi Della Rocca, S.: Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. Tech. rep., 31 pages (2014), http://hal.archives-ouvertes.fr/hal-01015171
4. Bellantoni, S., Cook, S.A.: A new recursion-theoretic characterization of the polytime functions. Computational Complexity 2, 97–110 (1992)
5. Coppola, P., Dal Lago, U., Ronchi Della Rocca, S.: Light logics and the call-by-value lambda-calculus. Logical Methods in Computer Science 4(4) (2008)
6. Dal Lago, U.: Context semantics, linear logic, and computational complexity. ACM Trans. Comput. Log. 10(4) (2009)
7. Dal Lago, U., Masini, A., Zorzi, M.: Quantum implicit computational complexity. Theor. Comput. Sci. 411(2), 377–409 (2010)
8. Danos, V., Joinet, J.B.: Linear logic and elementary time. Inf. Comput. 183(1), 123–137 (2003)
9. Girard, J.Y.: Light linear logic. Inf. Comput. 143(2), 175–204 (1998)
10. Jones, N.D.: Computability and complexity - from a programming perspective. Foundations of computing series. MIT Press (1997)
11. Jones, N.D.: The expressive power of higher-order types or, life without cons. J. Funct. Program. 11(1), 5–94 (2001)
12. Lafont, Y.: Soft linear logic and polynomial time. Theor. Comput. Sci. 318(1-2), 163–180 (2004)
13. Leivant, D.: Predicative recurrence and computational complexity I: word recurrence and poly-time. In: Feasible Mathematics II, pp. 320–343. Birkhauser (1994)
14. Leivant, D.: Calibrating computational feasibility by abstraction rank. In: LICS, p. 345. IEEE Computer Society (2002)
15. Leivant, D., Marion, J.Y.: Lambda-calculus characterizations of poly-time. Fundam. Inform. 19(1/2), 167–184 (1993)
16. Madet, A.: Implicit Complexity in Concurrent Lambda-Calculi. Ph.D. thesis, Université Paris 7 (December 2012), http://tel.archives-ouvertes.fr/tel-00794977
17. Madet, A., Amadio, R.M.: An elementary affine lambda-calculus with multithreading and side effects. In: Ong, L. (ed.) Typed Lambda Calculi and Applications. LNCS, vol. 6690, pp. 138–152. Springer, Heidelberg (2011)
18. Marion, J.Y.: A type system for complexity flow analysis. In: LICS, pp. 123–132. IEEE Computer Society (2011)
19. Mazza, D.: Linear logic and polynomial time. Mathematical Structures in Computer Science 16(6), 947–988 (2006)
20. Ronchi Della Rocca, S., Paolini, L.: The Parametric Lambda-Calculus: a Metamodel for Computation. Texts in Theoretical Computer Science. Springer, Berlin (2004), http://www.springer.com/sgw/cda/frontpage/0,5-40356-72-14202886-0,00.html
21. Ronchi Della Rocca, S., Roversi, L.: Lambda-calculus and intuitionistic linear logic. Studia Logica 59(3), 417–448 (1997)
22. Terui, K.: Light affine lambda-calculus and polynomial time strong normalization. Arch. Math. Log. 46(3-4), 253–280 (2007)

A Hoare-Like Calculus Using the SROIQ$^\sigma$ Logic on Transformations of Graphs*

Jon Haël Brenas[1], Rachid Echahed[1], and Martin Strecker[2]

[1] CNRS and University of Grenoble, France
[2] Université de Toulouse / IRIT, France

Abstract. We tackle the problem of partial correctness of programs processing structures defined as graphs. We introduce a kernel imperative programming language endowed with atomic actions that participate in the transformation of graph structures and provide a decidable logic for reasoning about these transformations in a Hoare-style calculus. The logic for reasoning about the transformations (baptized $\mathcal{SROIQ}^\sigma$) is an extension of the Description Logic (DL) $\mathcal{SROIQ}$, and the graph structures manipulated by the programs are models of this logic. The programming language is non-standard in that it has an instruction set targeted at graph manipulations (such as insertion and deletion of arcs), and its conditional statements (in loops and selections) are $\mathcal{SROIQ}^\sigma$ formulas. The main challenge solved in this paper is to show that the resulting proof problems are decidable.

Keywords: Description Logic, Graph Transformation, Programming Language Semantics, Tableau Calculus.

1 Introduction

1.1 Problem Statement and Contribution

The work presented here has arisen out of the authors' effort to prove properties about graph transformations. These transformations are ubiquitous, among others, in traditional imperative programs that modify pointer structures. The obstacle to satisfactory solutions in this area is that traditional programming languages are too expressive and interesting problems often need to be stated in non-decidable logics.

In this paper, we focus on a class of decidable Description Logics (DLs). The spectrum of DLs [1] is well explored, there are numerous application areas, such as capturing the static semantics of modeling languages (in the style of UML) or graph database schemas (in the style of RDF).

To be effective, the transformation is defined in a programming language. We propose an imperative language annotated with pre- and postconditions and

* This work has been funded by projects CLIMT (ANR-11-BS02-016) and TGV (CNRS-INRIA-FAPERGS/156779 and 12/0997-7).

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 164–178, 2014.

loop invariants. Peculiarities of the language are conditions in *if* and *while* statements that are Boolean queries, and a non-deterministic assignment statement. The language constructs are restricted to structural transformations and have been chosen carefully so that the resulting program verification problem becomes decidable.

Here, program verification means *a priori* verification: Given a program with its pre- and postcondition, can we ascertain that every input structure satisfying the precondition is transformed into a structure satisfying the postcondition? This is in contrast to *a posteriori* verification where satisfaction of the postcondition is checked individually for each graph, once the transformation has been performed. The latter has the disadvantage that the verification has to be done for each single instance (whereas our verification ensures correctness once and for all), and the approach becomes impractical for very large structures.

Technically speaking, we present a programming language, a logic and a Hoare-style *program calculus* relating them. We only consider partial correctness, *i.e.* correctness of a program under the condition that it terminates. We establish that the program calculus is sound *wrt.* the programming language semantics (if a pre-post-relation is established by the calculus, a graph is transformed as required).

The program calculus is related to, but has to be distinguished from a *logic calculus* which is used for establishing the validity of the correctness conditions extracted with the aid of the program calculus. We show that, for the fragment of correctness conditions, there is a logic calculus that is sound, complete and terminates.

Outline of the paper. After an introductory example in Sect. 1.2 and a review of related work in Sect. 1.3, we define the logical framework used for expressing program properties and conditions in statements (Sect. 2), before presenting the syntax and semantics of the programming language (Sect. 3). We then turn to more technical issues: intuitively, the extraction of weakest preconditions in Sect. 4 takes a program and its correctness condition and derives a formula whose validity ensures correctness. In Sect. 5, we show how to prove that such formulae are valid.

1.2 Example of Program

To get an intuition of the kind of transformation we are aiming at, let's consider the ontology Friend of a friend (FOAF)[1]. It is used to describe persons, their activities and their relationships with other people and objects. Its components are individuals, sets of individuals (called concepts here) and binary relations on individuals (called roles here).

The program whose correctness we want to prove modifies a graph representing this ontology. It is shown in Fig. 2. We consider the problem of moving a researcher R to a laboratory L. As the Friend of a friend ontology is much too

[1] The website of the project can be found at `www.foaf-project.org`

big to be efficiently reproduced in an introduction, we adapt it to our needs, as shown in Fig. 1.

$$
\begin{aligned}
\mathit{onto} \;:=\; & \textsc{Researcher} \subseteq \textsc{Agent} \wedge \textsc{Lab} \subseteq \textsc{Agent} \wedge \textsc{Distinguished} \subseteq \textsc{Agent} \\
& \wedge \textsc{Topic\_interest} :: \textsc{Agent} \times \textsc{Thing} \\
& \wedge \textsc{Topic} :: \textsc{Document} \times \textsc{Thing} \\
& \wedge \textsc{Publication} :: \textsc{Researcher} \times \textsc{Document} \\
& \wedge \textsc{Member} :: \textsc{Lab} \times \textsc{Agent}
\end{aligned}
$$

Fig. 1. An ontology example

The concept Researcher is used to represent researchers, the concept Lab represents laboratories, Agent is the concept of those that can "act", Thing is the representation of topic and finally Distinguished singles out those that have received a distinction. The role Topic_interest is used to represent the topics of interest of an agent while Topic represents the subjects of a document. The role Publication links a person to her publications. The role Member lists the members of a group.

The ontology provides the formal definition of the relationships between concepts and roles. It states, for instance, that Researchers are Agents and that the role Member relates Labs with their own Agents.

The precondition of the program stipulates (see Fig. 2) that the ontology is respected before starting the program, that L is a Lab, and that R is a Researcher.

The if statement then checks if the researcher is listed as being a member of a laboratory. If it is the case, we select that laboratory PL and we remove the researcher from their roster. If, in addition, there is no researcher left with a distinction, PL loses its Distinguished quality.

Now that the researcher is available, we add the fact that he is a member of L. In case R was Distinguished, L becomes Distinguished. L may well have been Distinguished before the arrival of R but that is not relevant.

The while loop adds all the topics that R has written articles about to the subjects that interest L. This is done by going through the set of Things that are not a topic of interest for the laboratory ($\neg(L$ Topic_interest $t)$) but that are the topic of a publication by R (R : **Ex** Publication (**Ex** Topic $\{t\}$)). While this set is not empty, we select one of its elements and we add it to the topics of interest of L. The size of the set thus decreases, which is encouraging but of no great importance as our framework does not check termination. The invariant of the loop is the same as the precondition.

The postcondition states that the ontology structure is satisfied again, that L is still a Lab, that R is still a person, that all the subjects of the publications of R are topics of interests of L, that R is now a member of L and that if R is distinguished so is L.

```
vars R, L, To, PL;
concepts Researcher, Lab, Document, Agent, Thing, Distinguished;
roles Topic_interest, Topic, Publication, Member;

pre: onto ∧ R : Researcher ∧ L : Lab;

if ∃ l. l Member R then {
   select PL with PL Member R;
   delR (PL Member R);
   if PL: All Member (¬ Distinguished) then {
      delC (PL: Distinguished)
   };
};
addR(L Member R);
if R: Distinguished then {
   addC(L: Distinguished)
};
while (∃ t. ¬(L Topic_interest t) ∧ R: Ex Publication (Ex Topic {t})) {
inv: onto ∧ R : Researcher ∧ L : Lab
  select To
     with ¬(L Topic_interest To) ∧ R : Ex Publication (Ex Topic {To});
  addR(L Topic_interest To)
};

post: onto ∧ R : Researcher ∧ L : Lab
∧ R: All Publication (All Topic (Ex Topic_interest⁻ {L}))
∧ L Member R
∧ R: Distinguished ⇒ L: Distinguished;
```

Fig. 2. An example program

A small graph illustrating the transformation is shown in Fig. 3. A distinguished researcher R which belongs to lab LB moves to lab L. The arrow relating R to LB is removed and a new one relating R to L is added. Moreover, LB no longer is distinguished. New arrows are created relating L to the topics of interest of R, say T_0 and T_1.

1.3 Related Work

Reasoning about graph transformations in full generality is hard [9]. A first step towards the verification of programs operating on graphs has been made in [7] where the authors follow Dijkstra's approach to program verification by constructing the weakest preconditions for so-called high-level programs. Pre- and post-conditions are expressed as *nested graph conditions*. These conditions have also been used recently in [12] where a Hoare-style program verification on

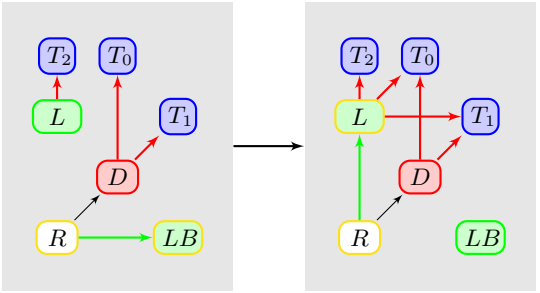


Fig. 3. Resulting transformation

graph programs has been proposed. Unfortunately the verification problem in these two proposals is undecidable in general.

Some decidable logics for graph transductions are known, such as MSO [6], but these are descriptive, applicable to a limited number of graphs and often do not match with an algorithmic notion of transformation. Some implementations of verification environments for pointer manipulating programs exist [11], but they often impose severe restrictions on the kind of graphs that can be manipulated, such as having a clearly identified spanning tree.

In [4], the authors investigated the static verification of the evolution of graph databases where integrity constraints are expressed in a description logic called $\mathcal{ALCHOIQ}br$. This work is very close to our proposal. However, the authors did consider only programs consisting of finite sequences of atomic actions. These actions may compute the union or the difference of roles and concepts. Their verification procedure is based on a transformation TR [4, Definition 5] which mimics the computation of weakest preconditions in Hoare's like calculi.

Work on Knowledge Bases (KB) updates [10] seems to approach the problem from the opposite direction: Add facts to a KB and transform the KB at the same time such that certain formulas remain satisfied. In our approach, the modification of the KB is exclusively specified by the program.

The present paper is a follow-up of a previous one by the authors [5] working on a simpler description logic (ALCQ) and a simpler programming language. In order to obtain decidable verification conditions, the logic $\mathcal{SROIQ}^{\sigma}$ requires more subtle restrictions on the form of assertions occurring in programs. The decision procedure (a tableau algorithm) differs from the one presented in [5].

2 The Logic $\mathcal{SROIQ}^{\sigma}$

In this section, we introduce a new description logic we call $\mathcal{SROIQ}^{\sigma}$. It is an extension of the description logic $\mathcal{SROIQ}$ [8] augmented with a notion of substitution. We show that the satisfiability problem in $\mathcal{SROIQ}^{\sigma}$ is decidable.

The decision procedure is intended to be as general as possible and to be adaptable to a wide variety of logics of the description logic family, under certain assumptions.

We start by some basic definitions.

Definition 1 (Concept and role names; nominals). *Let* $\boldsymbol{C}$ *be a set of* ***concept names*** *including a subset* $\boldsymbol{N}$ *of* ***nominals***, $\boldsymbol{R}$ *a set of* ***role names*** *including the universal role* U *and* $\boldsymbol{I}$ *a set of* ***individuals***. *The set of roles is* $\boldsymbol{R} \cup \{R^-|R \in \boldsymbol{R}\}$, *where a role* R^- *is called the* ***inverse role*** *of* R.

Example 1. In our example, $\mathbf{C} = \{$Researcher, Lab, Agent, Thing, Distinguished, R, L, PL, To$\}$, $\mathbf{N} = \{$ R, L, PL, To$\}$ and $\mathbf{R} = \{$Topic_interest, Topic, Publication, Member, $U\}$.

As usual, an **interpretation** $\mathcal{I} = (\Delta^\mathcal{I}, .^\mathcal{I})$ consists of a set $\Delta^\mathcal{I}$, called the **domain** of $\mathcal{I}$, and a **valuation** $.^\mathcal{I}$ which associates with every concept name C a subset $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$, which is a singleton for each nominal, with each role name R a binary relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, with the universal role U the universal relation $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ and with each individual name a an element $a^\mathcal{I} \in \Delta^\mathcal{I}$. The technical definition of interpretations could be consulted in *e.g.* [8,3].

The considered logic allows one to provide so-called *role axioms*. A role axiom can either be a role inclusion axiom or a role assertion. We will deal with the role inclusion axioms first. For that, we need to define an ordering on roles.

Definition 2. *A strict partial order* $\prec$ *on a set* A *is an irreflexive and transitive relation on* A. *A strict partial order* $\prec$ *on the set of roles is called a* ***regular order*** *if* $\prec$ *satisfies, additionally,* $S \prec R \Leftrightarrow S^- \prec R$ *for all roles* R *and* S.

Definition 3. *A* ***role inclusion axiom*** *is an expression of the form* $w \subseteq R$ *where* w *is a finite string of roles not containing the universal role* U *and* R *is a role name, with* $R \neq U$. *A* ***role hierarchy*** $\mathcal{R}_h$ *is a finite set of role inclusion axioms. A role inclusion axiom* $w \subseteq R$ *is* $\prec$***-regular*** *if* R *is a role name and* w *is defined by the following grammar:*
$w = RR \mid R^- \mid S_1 \dots S_n \mid RS_1 \dots S_n \mid S_1 \dots S_n R$ *with* $S_i \prec R$ *for all* $1 \leq i \leq n$. *Finally, a role hierarchy* $\mathcal{R}_h$ *is* ***regular*** *if there exists a regular order* $\prec$ *such that each role inclusion axiom in* $\mathcal{R}_h$ *is* $\prec$*-regular. An interpretation* ***satisfies*** *a role inclusion axiom* $w \subseteq R$ *if the interpretation of* w *is included in the interpretation of* R. *An interpretation is a* ***model*** *of a role hierarchy* $\mathcal{R}_h$ *if it satisfies all role inclusion axioms in* $\mathcal{R}_h$.

Example 2. Let us consider the roles Brother and Sibling with their intuitive meanings, it seems correct that Brother $\subseteq$ Sibling.

The second possible kind of role axiom is the role assertion.

Definition 4 (Role assertions). *For role names R,S, we call the assertions Ref(R) (role reflexivity), Irr(R) (role irreflexivity), Sym(R) (role symmetry), Asy(R) (role asymmetry), Tra(R) (role transitivity) and Dis(R,S) (role disjunction)* ***role assertions***.

Example 3. To keep with the roles previously defined, Sym(SIBLING) and Tra(SIBLING) are a correct set of role assertions.

One can observe that some of the role assertions (namely transitivity and symmetry) are simply a rewriting of some role axioms: $Sym(R)$ is equivalent to $R^- \subseteq R$, and $Tra(R)$ is equivalent to $RR \subseteq R$. For these reasons, we will henceforth only consider role assertions without Sym and Tra.

Finally, when introducing complex concepts, we will need simple roles to avoid undecidability. Intuitively, a simple role is a role that does not appear as the right-hand side of a role inclusion axiom whose left-hand side is a string composed of at least two roles.

Definition 5 (Simple role). *Given a role hierarchy $\mathcal{R}_h$ and a set of role assertions $\mathcal{R}_a$, a simple role is inductively defined as either a role name that does not occur in the right-hand side of any role inclusion axiom, or R^- for R simple, or the right-hand side of a role inclusion axiom $w \subseteq R$ where w is a simple role. $\mathcal{R}_a$ is called **simple** if all roles appearing in role assertions are simple.*

Starting from now, the only role hierarchies that we consider are regular and the only sets of role assertions that we consider are finite and simple.

Definition 6 (Concept). *A concept is defined as:*
$C ::= \bot \mid c \mid \neg\, C \mid C \sqcap D \mid C \sqcup D \mid (\geq n\ S\ C) \mid (< n\ S\ C) \mid$ **Ex** $R\ C \mid$ **All** $R\ C \mid \{o\} \mid$ **Ex** S **Self** $\mid C\ subst$
where c is a concept name, R is a role, S is a simple role, o is a nominal, C, D are concepts and subst is a substitution.

Intuitively, $\neg C$ stands for the complement of C with respect to the domain of interpretation. $C \sqcap D$ (respectively $C \sqcup D$) stands for the intersection (respectively the union) of concepts. $(\geq n\ S\ C)$ (respectively $(< n\ S\ C)$) stands for the set of elements related via role S to at least n (respectively at most *n-1*) distinct individuals of concept C. **Ex** $R\ C$ stands for the set of elements related via role R to at least one individual of concept C and **All** $R\ C$ stands for the set of elements related via role R only to elements of concept C. $\{o\}$ stands for the singleton associated to nominal o. **Ex** S **Self** stands for the set of elements related to themselves via role S. $C\ subst$ stands for the set of elements of C updated according to the substitution *subst*. Missing definitions can be found in [3].

Substitutions, that appear in the last constructor, allow one to modify roles and concepts by adding or removing individuals. Substitutions, being the difference between $\mathcal{SROIQ}^\sigma$ and $\mathcal{SROIQ}$, are defined next. As will be shown later on, the computation of weakest preconditions and verification conditions may generate substitutions of the following form.

Definition 7 (Substitution). *Given a concept name c and a role name R, a substitution is:*

subst ::= ϵ *(empty substitution)*
| $[RS]$ *(role substitution)*
| $[CS]$ *(concept substitution)*
A role substitution is defined as follows:
RS ::= $R-(i,j)$ *(deletion of relation instance)*
| $R+(i,j)$ *(insertion of relation instance)*
while a concept substitution is defined as follows:
CS ::= $c-i$ *(deletion of a concept instance)*
| $c+i$ *(insertion of a concept instance)*

Example 4. All elements of **C** are examples of concepts. Another example of concept is **All** MEMBER ¬DISTINGUISHED. It can be translated into "none of the members is distinguished".

Theorem 1. *If Φ_0 is a concept and $\mathcal{R}_h$ is a regular role hierarchy and $\mathcal{R}_a$ is a finite simple set of role assertions, the satisfiability of $\mathcal{R}_h \wedge \mathcal{R}_a \wedge \Phi_0$ is decidable.*

The proof of Theorem 1 can be found in [3].

In the following, we introduce the notions of assertions and conditions used in the rest of the paper.

Definition 8 (Assertion). *An assertion is defined as either:*
assert ::= $i : C \mid i\ R\ j \mid i\ (\neg\ R)\ j \mid i = j \mid i \neq j \mid$ *role_axiom* $\mid \neg$ *assert* $\mid$ *assert* $\wedge$ *assert* $\mid$ *assert* $\vee$ *assert* $\mid$ **All** $U\ C \mid$ **Ex** $U\ C \mid \forall$ *i.assert* $\mid \exists$ *i.assert* $\mid$ *assert subst*
where C is a concept, role_axiom is either a role inclusion axiom or a role assertion, i, j are individuals, R is a role, U is the universal role defined previously and subst is a substitution.

Example 5. Assertions without substitutions are the building blocks of ontologies. Our simplified example of FOAF (cf. Fig. 1) contains two main kinds of assertions. The first deals with the hierarchy of concepts and the second one with the concepts of the elements linked by a role.

Among others, RESEARCHER $\subseteq$ AGENT is a short way of writing the assertion **All** U ¬RESEARCHER $\sqcup$ AGENT which can be translated into "researchers are agents".

In the next definition, we introduce the notion of *conditions*, which is used in Sect. 3 in while-loops, if-statements and select-statements.

Definition 9 (Condition). *A condition is an assertion without role axioms and without quantification on individuals, that is no sub-expression of the form $\forall$ i.assert or $\exists$ i.assert.*

3 Programming Language

3.1 Syntax

In this section, we introduce the programming language for performing transformations (see the example of Fig. 2). The programming language is an imperative

language manipulating relational structures. Its distinctive features are conditions (in conditional statements and loops) in the sense of Sect. 2. These formulas can be understood as Boolean queries on a database. The language also has a non-deterministic assignment statement allowing to select an element satisfying a condition. This corresponds to a database query retrieving an element satisfying a condition. Traditional types (numbers, inductive types) are not provided in the language.

In this paper, we only consider a core language with traditional control flow constructs, but without procedures. The language has primitives for adding an individual element to a concept, or for removing it. Similarly, there are primitives for the insertion or removal of edges. Thus, it is only possible to modify a relational structure, but not to allocate or deallocate objects, in a strict sense.

The abstract syntax of statements is defined by:

stmt ::=	`Skip`	(empty stmt)
	| `addC(`i : c`)`	(insert element)
	| `delC(`i : c`)`	(delete element)
	| `addR(`i R j`)`	(insert edge)
	| `delR(`i R j`)`	(delete edge)
	| `select` i `with` *cond*	(assignment)
	| *stmt* `;` *stmt*	(sequence)
	| `if` *cond* `then` { *stmt* } `else` { *stmt* }	
	| `while` *cond* { `inv:` *assert* *stmt*}	

The non-terminals *cond* and *assert* corresponds, respectively, to conditions (defined in Def 9) and assertions (defined in Def 8). i and j stand for individuals, c stands for a concept name and R stands for a role name. There are two variants of insertion and deletion operations (for individuals and a concept name (`addC` and `delC`) and for two individuals and a relation name (`addR` and `delR`)).

A program is a statement embedded in declarations of variables, concepts and roles and a pre- and a postcondition.

prog ::= `vars` $\overrightarrow{i}$`;` `concepts` $\overrightarrow{c}$`;` `roles` $\overrightarrow{R}$`;`
`pre:` *assert*`;` *stmt*`;` `post:` *assert*`;`

3.2 Semantics

The semantics is a big-step semantics describing how a state evolves during the execution of a statement. The state is a relational structure, and the state space is just the type of interpretations. In accordance with traditional notation in semantics, we use the symbol σ to denote a state. We may therefore write $\sigma(b)$ to evaluate the condition b in state σ.

The rules have the form $(s, \sigma) \Rightarrow \sigma'$ expressing that executing statement s in state σ produces a new state σ'. The rules of the semantics are given below. Beware that we overload logical symbols such as $\exists$, $\wedge$ and $\neg$ for use in the meta-syntax and as constructors of *assert*.

The rules of the traditional control constructs are standard, apart from the fact that we do not use expressions as conditions. The invariant in the *while*-loop

$$\frac{}{(\texttt{Skip}, \sigma) \Rightarrow \sigma} \ (Skip) \qquad \frac{(s_1, \sigma) \Rightarrow \sigma'' \quad (s_2, \sigma'') \Rightarrow \sigma'}{(s_1;s_2, \sigma) \Rightarrow \sigma'} \ (Seq)$$

$$\frac{\sigma' = \sigma^{[\sigma(c):=\sigma(c)\cup\{\sigma(i)\}]}}{(\texttt{addC}(i \ : \ c), \sigma) \Rightarrow \sigma'} \ (AddC) \qquad \frac{\sigma' = \sigma^{[\sigma(c):=\sigma(c)\cap\overline{\{\sigma(i)\}}]}}{(\texttt{delC}(i \ : \ c), \sigma) \Rightarrow \sigma'} \ (DelC)$$

$$\frac{\sigma' = \sigma^{[\sigma(R):=\sigma(R)\cup\{\sigma(i_1),\sigma(i_2)\}]}}{(\texttt{addR}(i_1 \ R \ i_2), \sigma) \Rightarrow \sigma'} \ (AddR) \qquad \frac{\sigma' = \sigma^{[\sigma(R):=\sigma(R)\cap\overline{\{\sigma(i_1),\sigma(i_2)\}}]}}{(\texttt{delR}(i_1 \ R \ i_2), \sigma) \Rightarrow \sigma'} \ (DelR)$$

$$\frac{\sigma(b) \quad (s_1, \sigma) \Rightarrow \sigma'}{(\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2, \sigma) \Rightarrow \sigma'} \ (IfT) \qquad \frac{\neg\sigma(b) \quad (s_2, \sigma) \Rightarrow \sigma'}{(\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2, \sigma) \Rightarrow \sigma'} \ (IfF)$$

$$\frac{\exists vi.(\sigma' = \sigma^{[v:=vi]} \wedge \sigma'(b))}{(\texttt{select } v \texttt{ with } b, \sigma) \Rightarrow \sigma'} \ (SelAssT) \qquad \frac{\neg\sigma(b)}{(\texttt{while } b \texttt{ inv} : f \ s, \sigma) \Rightarrow \sigma} \ (WF)$$

$$\frac{\sigma(b) \quad (s, \sigma) \Rightarrow \sigma'' \quad (\texttt{while } b \texttt{ inv} : f \ ; s, \sigma'') \Rightarrow \sigma'}{(\texttt{while } b \texttt{ inv} : f \ ; s, \sigma) \Rightarrow \sigma'} \ (WT)$$

Fig. 4. Big-step semantics rules

is without operational significance. It is only used for calculating weakest preconditions (Sect. 4).

For lack of space, we do not detail all the rules here as they are quite intuitive. We roughly explain rules `addC` and `select`:

- `addC`(i : c) adds a node to a concept. Adding an already existing element has no effect (*i.e.* , is not perceived as an error). $[\sigma(c) := \sigma(c) \cup \{\sigma(i)\}]$ modifies the interpretation for c to include the element that i denotes.

$$\frac{\sigma' = \sigma^{[c:=\sigma(c)\cup\{\sigma(i)\}]}}{(\texttt{addC}(i \ : \ c), \sigma) \Rightarrow \sigma'} \ (AddC)$$

- `select` i `with` b selects an element vi from the semantic domain that satisfies condition b (note that i typically occurs in b), and assigns it to i. The subsequent statements are then executed with i bound to vi. For example, `select` a `with` $a : A \wedge (a \, R \, d)$ selects an element which is an instance of concept A and is R-related to a given element referred to by d, and assigns it to a. More formally, we pick an instance $vi \in \Delta$, check whether the condition b would be satisfied under this choice, and if this is the case, keep this assignment:

$$\frac{\exists vi.(\sigma' = \sigma^{[i:=vi]} \wedge \sigma'(b))}{(\texttt{select } i \texttt{ with } b, \sigma) \Rightarrow \sigma'} \ (SelAssT)$$

  Note that the semantics blocks (*i.e.* , there is no successor state) in case no instance satisfying the condition exists.

4 Verification Conditions

4.1 Generating Verification Conditions

We follow the standard approach for verifying that a program satisfies its specification: If the program has precondition *pre*, statement s and postcondition *post*, we compute the weakest precondition $wp(s, post)$ and then show that it is implied by the precondition. Using the terminology of Sect. 1.1, this section is thus concerned with a program calculus.

The definition of wp is given in Fig. 5. Let us insist on one point: in traditional expositions of Hoare calculi, substitution is a meta-operation which syntactically replaces a symbol by an expression. This works as long as the syntax of the logic is closed under meta-substitutions, which is not the case we consider. For example, a replacement of R by $R - (v_1, v_2)$ in $(< n\ R\ C)$ would yield a syntactically ill-formed concept expression. This motivates our introduction of explicit substitutions as a constructor.

$$
\begin{array}{l}
wp(\texttt{Skip},\ Q) = Q \\
wp(\texttt{addC}(i\ :\ c)\ Q) = Q[c := c + i] \\
wp(\texttt{delC}(i\ :\ c),\ Q) = Q[c := c - i] \\
wp(\texttt{addR}(i_1\ R\ i_2),\ Q) = Q[R := R + (i_1, i_2)] \\
wp(\texttt{delR}(i_1\ R\ i_2),\ Q) = Q[R := R - (i_1, i_2)] \\
wp(\texttt{select}\ i\ \texttt{with}\ b,\ Q) = \forall i.(b \longrightarrow Q) \\
wp(s_1; s_2,\ Q) = wp(s_1, wp(s_2,\ Q)) \\
wp(\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2,\ Q) = (b \rightarrow wp(s_1, Q)) \wedge (\neg b \rightarrow wp(s_2, Q))) \\
wp(\texttt{while}\ b\ \texttt{inv}: f\ s,\ Q) = f
\end{array}
$$

Fig. 5. Weakest preconditions

Also, our *while*-loops are supposed to be annotated with invariants. Whether these invariants necessarily have to be supplied by the human end-user or whether

$$
\begin{array}{l}
vc(\texttt{Skip},\ Q) = \top \\
vc(\texttt{add}(i\ :\ c),\ Q) = \top \\
vc(\texttt{delete}(i\ :\ c),\ Q) = \top \\
vc(\texttt{add}(i_1\ R\ i_2),\ Q) = \top \\
vc(\texttt{delete}(i_1\ R\ i_2),\ Q) = \top \\
vc(\texttt{select}\ i\ \texttt{with}\ b,\ Q) = \top \\
vc(s_1; s_2,\ Q) = vc(s_1, wp(s_2, Q)) \wedge vc(s_2, Q) \\
vc(\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2,\ Q) = vc(s_1, Q) \wedge vc(s_2, Q) \\
vc(\texttt{while}\ b\ \texttt{inv}: f\ s,\ Q) = (f \wedge \neg b \longrightarrow Q) \\
\qquad\qquad\qquad\qquad \wedge (f \wedge b \longrightarrow wp(s, f)) \wedge vc(s, f)
\end{array}
$$

Fig. 6. Verification conditions

they could be inferred automatically in a pre-processing step is not subject of concern here. In any case, program verification also has to ascertain that the given loop annotation has the desired properties of an invariant: being preserved during execution of the loop body, and ensuring the postcondition when the loop terminates. Recursively collecting these verification conditions is done by function $vc(s, post)$ for a statement s and postcondition $post$ (Fig. 6).

4.2 Correctness

The two aforementioned criteria are used to define the *correctness condition* of a program *prog* with precondition pre, statement s and postcondition $post$:

$$correct(pre,\ s,\ post) =_{def} vc(s, post) \wedge (pre \longrightarrow wp(s, post))$$

We now have the necessary notions to state the soundness of our program calculus:

Theorem 2 (Soundness). *Let prog be a program with precondition pre, statement s and postcondition. post If correct*$(pre,\ s,\ post)$ *is valid, then for all states* σ *and* σ'*, if* $(s, \sigma) \Rightarrow \sigma'$*, then* $\sigma(pre)$ *implies* $\sigma'(post)$*.*

The proof of this theorem is straightforward and is done by induction on the structure of the statements.

5 Proving Verification Conditions

Let us recapitulate the development so far: In Sect. 3, we have presented a programming language annotated with formulas specifying the correctness of programs. In Sect. 4, we have given a program calculus (embodied by function *correct*) that takes an annotated program, removes all computational contents and returns a formula, say Φ. For sake of decidability of the verification program, we focus in this section on assertions which generate a particular formula Φ we call *essentially universally quantified.*

Definition 10 (Essentially quantified). *We say that an assertion* Φ *is* ***essentially universally quantified*** *(respectively* ***essentially existentially quantified****) if the occurrences of* $\forall$ *in* Φ *are only below an even (respectively odd) number of negations and the occurrences of* $\exists$ *in* Φ *are only below an odd (respectively even) number of negations.*

Lemma 1 (Universally quantified).

1. *Let* Q *be essentially universally quantified. Assume that the invariants in statement* s *do not include negated role axioms. Then* $wp(s, Q)$ *and* $vc(s, Q)$ *are essentially universally quantified.*
2. *If pre (respectively post) is essentially existentially (respectively universally) quantified and the invariants in statement s do not include negated role axioms , then correct*$(pre,\ s,\ post)$ *is essentially universally quantified and does not contain substitutions over negated role axioms.*

We now discuss briefly a decision procedure for verifying the validity of essentially universally quantified formulae. For more details see [2].

Actually, what we have to do is to prove that we can apply Theorem 1 to prove the validity of $correct(pre,\ s,\ post)$ whenever it is essentially universally quantified.

The first thing to do is to make sure that substitutions only affect the basic components of the assertion (such as role axioms and concepts). This can be done by pushing substitutions by using the following rules.

- $(\neg\ assert)\ subst \rightsquigarrow \neg\ (assert\ subst)$
- $(assert_1\ \wedge\ assert_2)\ subst \rightsquigarrow (assert_1\ subst)\ \wedge\ (assert_2\ subst)$
- $(assert_1\ \vee\ assert_2)\ subst \rightsquigarrow (assert_1\ subst)\ \vee\ (assert_2\ subst)$
- $(\forall i.\ assert)\ subst \rightsquigarrow \forall i.\ (assert\ subst)$
- $(\exists i.\ assert)\ subst \rightsquigarrow \exists i.\ (assert\ subst)$

It happens that the formulae generated by *correct*, after pushing the substitutions, may include substitutions over role axioms as well as quantifiers which prevent the direct use of Theorem 1. To overcome this drawback we actually show that we can get rid of those substitutions by means of a set of transformation rules. Unfortunately, there is not enough room here to give all the transformations. We give below two examples of such rules. The first rule shows how to get rid of a particular substitution $[R := R + (i_1, i_2)]$ when applied to the role axiom $Asym(R)$. The second rule shows how to get rid of a particular substitution $[R := R - (i_1, i_2)]$ when applied to the role axiom $s_1 \ldots s_n R \subseteq R$.

- $Asym(R)[R := R + (i_1, i_2)] \rightsquigarrow i_2\ \neg R\ i_1 \wedge Asym(R)$ that is R will be asymmetric after adding the edge (v_1, v_2) to R if R was asymmetric before and (i_2, i_1) is not already part of R.
- $(s_1 \ldots s_n R \subseteq R)[R := R - (i_1, i_2)] \rightsquigarrow \forall x.\forall y.\forall z.x : \mathbf{All}\ s_1\ \ldots \mathbf{All}\ s_n\ \neg\{y\} \vee y\ (\neg r)\ z \vee (x\ =\ i_1 \wedge z\ =\ i_2) \vee x\ R\ z$. That can be rewritten as $\forall x.\forall y.\forall z.\ (x : \mathbf{Ex}\ s_1\ \ldots \mathbf{Ex}\ s_n\ \{y\} \wedge y\ R\ z \wedge (x\ \neq\ i_1 \vee z\ \neq\ i_2)) \Longrightarrow x\ R\ z$ that is $s_1 \ldots s_n R \subseteq R$ after removing (i_1, i_2) from R if for each couple (x, z) different from (i_1, i_2), for each element y such that there is a path $s_1 \ldots s_n$ from x to y and $y\ R\ z$, then $x\ R\ z$.

Lemma 2. *For every essentially universally quantified formula not containing substitutions over negated role axioms, there is an equivalent universally quantified formula without substitutions on role axioms.*

Now that substitutions only occur over concepts, we get a formula $\Phi_1$which is essentially universally quantified. The last step before using Theorem 1 consists in eliminating the quantifiers of Φ_1. The rough lines of the procedure for determining whether $\Phi_1$is valid are spelled out in the following.

1. Convert $\Phi_1$to an equivalent prenex normal form p, which will consist of a prefix of universal quantifiers, and a quantifier-free body: $\forall x_1 \ldots x_n.b$
2. p is valid iff its universal closure $ucl(p)$ (universal abstraction over all free variables of p) is.

3. Show the validity of $ucl(p)$ by showing the unsatisfiability of $\neg ucl(p)$.
4. $\neg ucl(p)$ has the form $\neg\forall v_1 \dots v_k, x_1 \dots x_n.b$. Pull negation inside the universal quantifier prefix, remove the resulting existential quantifier prefix, and show unsatisfiability of $\neg b$ by using Theorem 1.

Computation of prenex normal forms is standard. Care has to be taken to avoid capture of free variables, by renaming bound variables. Free variables are defined as usual; the free variables of a substitution $f[R := R - (i_1, i_2)]$ are those of f and in addition i_1 and i_2 (similarly for edge insertion). We illustrate the problem with the following statement prg:
select a **with** $a : A$; **select** b **with** $b\ R\ a$;
select a **with** $a\ \neg R\ b$; **addR**$(b\ R\ a)$

Assume the post-condition is $Asym(R)$, we obtain $wp(prg, Q) =$
$\forall a.a : A \longrightarrow \forall b.(b\ R\ a) \longrightarrow \forall a.(a\ \neg R\ b) \longrightarrow Asym(R)[R := R + (b, a)]$.
Removing the substitution yields
$wp(prg, Q) = \forall a.a : A \longrightarrow \forall b.(b\ R\ a) \longrightarrow \forall a.(a\ \neg R\ b) \longrightarrow (a\ \neg R\ b \wedge Asym(R))$
whose prenex normal form
$\forall a_1, b, a_2.\ (a_1 : A \longrightarrow (b\ R\ a_1) \longrightarrow (a_2\ \neg R\ b) \longrightarrow (a_2\ \neg R\ b \wedge Asym(R)))$
contains more logical variables than prg contains program variables.

After removing the quantifiers and taking the negation, we obtain
$\neg(a_1 : A \longrightarrow (b\ R\ a_1) \longrightarrow (a_2\ \neg R\ b) \longrightarrow (a_2\ \neg R\ b \wedge Asym(R)))$ an assertion without substitutions over role axioms and without quantifiers on individuals whose unsatisfiability is equivalent to the validity of *correct*. This assertion fits the conditions of Theorem 1 and thus the validity of *correct* is decidable.

6 Conclusions

This paper proposes a language for rewriting graphs, and methods for reasoning about the correctness of these programs, by means of a Hoare-style calculus. DL formulas are directly integrated into the statements of the programming language. The verification conditions extracted from these programs have been shown to be decidable.

The work described here is still not entirely finished, and the following points indicate directions for future investigations:

- We are in the process of coding the theory in the Isabelle proof assistant. Most proofs concerning the elimination of substitutions and the tableau algorithm still have to be done. The purpose is to obtain a framework that will allow us to experiment more easily with variations of the logic.
- We have currently focused on the logic $\mathcal{SROIQ}^\sigma$, which is one of the most expressive description logics. It might be interesting to consider less expressive logics which offer more space for optimizations. The process described in Sect. 5 is rather generic, but it remains to be seen which other DLs can be accommodated.
- In a similar vein, it would be interesting to implement a transformation engine on the basis of the language described here, also with the purpose of evaluating the practical expressiveness of the language on larger examples.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
2. Brenas, J.H., Echahed, R., Strecker, M.: A hoare-like calculus using the $\mathcal{SROIQ}^{\sigma}$ logic on transformation of graphs (extended version). CoRR, arxiv.org (2014)
3. Brenas, J.H., Echahed, R., Strecker, M.: $\mathcal{SROIQ}^{\sigma}$ is decidable. CoRR, arxiv.org (2014)
4. Calvanese, D., Ortiz, M., Simkus, M.: Evolving graph databases under description logic constraints. In: Proc. of the 26th Int. Workshop on Description Logics (DL 2013). CEUR Electronic Workshop Proceedings, vol. 1014, pp. 120–131 (2013)
5. Chaabani, M., Echahed, R., Strecker, M.: Logical foundations for reasoning about transformations of knowledge bases. In: Eiter, T., Glimm, B., Kazakov, Y., Krötzsch, M. (eds.) DL Description Logics. CEUR Workshop Proceedings, vol. 1014, pp. 616–627. CEUR-WS.org (2013)
6. Courcelle, B., Engelfriet, J.: Graph structure and monadic second-order logic, a language theoretic approach. Cambridge University Press (2011)
7. Habel, A., Pennemann, K.-H., Rensink, A.: Weakest preconditions for high-level programs. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 445–460. Springer, Heidelberg (2006)
8. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: Proc. of the 10th International Conference of Knowledge Representation and Reasoning, KR-2006, Lake District UK (2006)
9. Immerman, N., Rabinovich, A., Reps, T., Sagiv, M., Yorsh, G.: The boundary between decidability and undecidability for transitive-closure logics. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 160–174. Springer, Heidelberg (2004)
10. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artificial Intelligence 175(18), 2170–2197 (2011)
11. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: PLDI, pp. 221–231 (2001)
12. Poskitt, C.M., Plump, D.: Hoare-style verification of graph programs. Fundam. Inform. 118(1-2), 135–175 (2012)

Termination Analysis for Graph Transformation Systems

H.J. Sander Bruggink[1], Barbara König[1], and Hans Zantema[2]

[1] Universität Duisburg-Essen, Germany
{sander.bruggink,barbara_koenig}@uni-due.de
[2] Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen, The Netherlands
h.zantema@tue.nl

Abstract. We introduce two techniques for proving termination of graph transformation systems. We do not fix a single initial graph, but consider arbitrary initial graphs (uniform termination), but also certain sets of initial graphs (non-uniform termination). The first technique, which can also be used to show non-uniform termination, uses a weighted type graph to assign weights to graphs. The second technique reduces uniform termination of graph transformation systems of a specific form to uniform termination of cycle rewriting, a variant of string rewriting.

1 Introduction

Termination, the absence of infinite computations, is a property that is required in many applications, in particular in model transformation, algorithms and protocol specifications. Many of these applications, such as graphical model transformation (for example, of UML models) and algorithms that manipulate data structures on the heap, are naturally modeled by graph transformation systems. This paper is concerned with the termination of such graph transformation systems. In particular we study the following question: given a set of graph transformation rules, and possibly an infinite set of potential initial graphs, does a transformation sequence of infinite length from one of the initial graphs exist? This problem is undecidable in general [9], but it is still important to develop semi-decision procedures that correctly decide as many instances as possible (and output "unknown" in the others).

Although termination is a basic notion of any computational formalism, it has not received a lot of attention in the graph transformation community; the focus is on reachability problems – the question whether a graph with some required or unwanted property is reachable from an initial graph. However, some prior work on the topic exists, mostly applied to model transformation [2,6,10]. Similar to [3] we follow a more general approach. We consider graph transformation from a theoretical point of view. This has the disadvantage of making results harder to obtain, but the advantage of being more broadly applicable.

The paper is organized as follows. In Sect. 2 we recapitulate definitions and fix notation. The heart of the paper is formed by Sect. 3, in which we introduce the

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 179–194, 2014.

weighted type graph technique for proving termination of graph transformation systems. We define the technique, consider special cases and investigate its limits; finally we give a detailed example that demonstrates its strengths. In Sect. 4 we show that termination of graph transformation systems of a specific kind can be reduced to termination of cycle rewriting, which is a form of rewriting that is related to string rewriting. This clarifies the relation to string rewriting and provides us with an additional method for graphs. Finally, we briefly present an implementation of the techniques of this paper in Sect. 5, compare with related work in Sect. 6 and give pointers for further research in Sect. 7. A full version of this paper with proofs and additional details can be downloaded from the first author's website.

2 Preliminaries

We first introduce graphs, morphisms, and graph transformation, in particular the double pushout approach [5]. We use edge-labeled, directed graphs.

Definition 1 (Graph). *Let Λ be a fixed set of edge labels. A* Λ-labeled graph *is a tuple $G = \langle V, E, src, tgt, lab \rangle$, where V is a set of nodes, E is a set of edges, $src, tgt \colon E \to V$ assign to each edge a source and a target, and $lab \colon E \to \Lambda$ is a labeling function.*

As a notational convention, we will denote, for a given graph G, its components by V_G, E_G, src_G, tgt_G and lab_G, unless otherwise indicated. The size of a graph G, written $|G|$, is the number of nodes and edges it contains, that is $|G| = |V_G| + |E_G|$.

Definition 2 (Graph morphism). *Let G, G' be two Λ-labeled graphs. A* graph morphism *$\varphi \colon G \to G'$ consists of two functions $\varphi_V \colon V_G \to V_{G'}$ and $\varphi_E \colon E_G \to E_{G'}$, such that for each edge $e \in E_G$ it holds that $src_{G'}(\varphi_E(e)) = \varphi_V(src_G(e))$, $tgt_{G'}(\varphi_E(e)) = \varphi_V(tgt_G(e))$ and $lab_{G'}(\varphi_E(e)) = lab_G(e)$.*

We will often drop the subscripts V, E and simply write φ instead of φ_V, φ_E.

Definition 3 (Graph transformation). *A* graph transformation rule *ρ consists of two injective morphisms $L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$, consisting of the* left-hand side *L, the* right-hand side *R and the* interface *I.*

A match *of a left-hand side in a graph G is an injective morphism $m \colon L \to G$.*

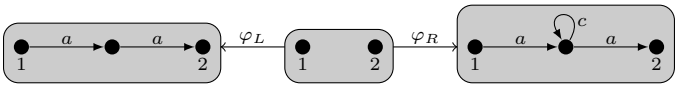
Given a rule ρ and a match $m \colon L \to G$, a graph H is the result of applying the rule at the match, written $G \Rightarrow_{m,\rho} H$ (or $G \Rightarrow_\rho H$ if m is arbitrary or clear from the context), if there exists a graph C and injective morphisms such that the two squares in the diagram on the right are pushouts in the category of graphs and graph morphisms.

$$\begin{array}{ccccc} L & \xleftarrow{\varphi_L} & I & \xrightarrow{\varphi_R} & R \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow & (\mathrm{PO}) & \downarrow \\ G & \longleftarrow & C & \longrightarrow & H \end{array}$$

A graph transformation system *$\mathcal{R}$ is a set of graph transformation rules. For a graph transformation system $\mathcal{R}$, $\Rightarrow_\mathcal{R}$ is the rewriting relation on graphs induced by those rules.*

The above formal definition of a graph transformation step can be algorithmically described as follows. Let a rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ and a match $m\colon L \to G$ be given. The rule can be applied to the match if for every edge e of G which is not in the image of m it holds, for $v \in \{src(e), tgt(e)\}$, that v has a pre-image in I (under $m \circ \varphi_L$) if v has a pre-image in L (under m). In this case we say that the *dangling edge condition* is satisfied. If the rule is applicable, all images of elements in L, whose preimages are not in the interface I, are removed from G. This gives us the "context" graph C. Furthermore the elements of R that do not have a preimage in I are added and connected with the remaining elements, as specified by the interface. This results in the graph H. The dangling edge condition ensures that nodes can only be deleted if all incident edges are deleted.

Example 1. We take as label set $\Lambda = \{a, c\}$. Consider the following graph transformation rule:



The numbers represent which nodes are mapped to each other. The following is a legal transformation step using the above rule:

There is no step replacing the *aa*-pattern at the bottom, because the middle node, although deleted by the rule, is incident to a *c*-edge not in the pattern (the dangling edge condition is not satisfied, that is, edges would be left dangling).

A graph is discrete when it does not contain any edges. A well-known result from double-pushout graph transformation is that we can restrict to rules with discrete interfaces without affecting the expressive power of the formalism: for each rule with non-discrete interface a rule with discrete interface exists which induces the same rewrite relation. As examples we will only use rules with discrete interfaces, although the results of Sect. 3 are also applicable to graph transformation systems that contain rules with non-discrete interfaces.

Let $\mathcal{L}$ be a set of graphs. We say that a set of rules $\mathcal{R}$ is *$\mathcal{L}$-terminating* if each reduction sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} \cdots$ with $G_0 \in \mathcal{L}$ is finite. The set of rules $\mathcal{R}$ is *uniformly terminating* or simply *terminating* if it is $\mathcal{G}$-terminating, where $\mathcal{G}$ is the set of all graphs.

We will specify sets of graphs, in this setting called *graph languages*, by so-called type graphs.[1] A type graph is just a graph T. The graph language accepted by T, written $\mathrm{L}(T)$, is the set of all graphs from which there exists a morphism

[1] In the literature, for example [4], typing morphisms are often considered as an intrinsic part of graph and rule definitions. We consider untyped graphs and rules, and use type graphs merely as a means to describe graph languages.

into T, that is: $\mathrm{L}(T) = \{G \mid \text{there exists a morphism } \varphi\colon G \to T\}$. A type graph T is closed under a set of rules $\mathcal{R}$, if for each rule $L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R \in \mathcal{R}$ and morphism $t_L\colon L \to T$, there exists a morphism $t_R\colon R \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$. A type graph T being closed under a set of rules $\mathcal{R}$ ensures that $\mathrm{L}(T)$ is closed under $\mathcal{R}$-reachability, that is, it ensures that if $G \Rightarrow_{\mathcal{R}} H$ and $G \in \mathrm{L}(T)$, then also $H \in \mathrm{L}(T)$.

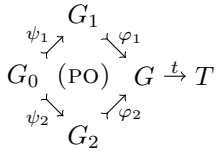
3 Termination Analysis via Weighted Type Graphs

In this section, we present a termination argument based on weighted type graphs. The technique is inspired by the semantic labeling technique for proving termination of term rewrite systems [11], where the algebra is replaced by type graphs.

3.1 Weighted Type Graphs

We assume a set W of *weights* with a binary operation $\oplus$ and a well-founded partial order $<$ such that the following holds: for $a, b, c \in W$ we have that (i) $a < b \iff a \oplus c < b \oplus c$, and (ii) $a = b \iff a \oplus c = b \oplus c$. Note that from these two conditions it follows that (iii) $a \leq b \iff a \oplus c \leq b \oplus c$.

Furthermore, for a given graph T, called the *type graph*, we have a weight function w that assigns a weight from W to every morphism $\varphi\colon G \to T$. This weight function must be stable under pushouts in the following sense: given a pushout of injective morphisms as shown on the right and an arrow $t\colon G \to T$ from the pushout object G into the type graph, we have that $w(t) \oplus w(t \circ \varphi_1 \circ \psi_1) = w(t \circ \varphi_1) \oplus w(t \circ \varphi_2)$. (Note that $\varphi_1 \circ \psi_1 = \varphi_2 \circ \psi_2$.)



The intuiton behind stability under pushouts is the following. A pushout can be seen as an operation which glues together two graphs over a common interface. Above, the graphs G_1 and G_2 are glued together over the common interface G_0. The weight of morphisms from the G to T should be obtained by adding together the weights of the corresponding morphisms from G_1 and G_2 to T. However, in $w(t \circ \varphi_1) \oplus w(t \circ \varphi_2)$ the common interface is counted twice, so we have to add $w(t \circ \varphi_1 \circ \psi_1)$ to the left-hand side to balance the equation.

Although the termination argument will be stated for weight functions of this form in general, the specific type of weight function we will use in examples and in Sect. 3.4 will be so-called *linear weight functions*, which are defined as follows. Weights are natural numbers with summation and order $<$. Let $d\colon (V_T \cup E_T) \to \mathbb{N}$ be a function which assigns a weight to each node and edge of the type graph T. The linear weight function w_d for d assigns to a morphism $\varphi\colon G \to T$ the weight $w_d(\varphi) = \sum_{x \in (V_G \cup E_G)} d(\varphi(x))$.

Proposition 1. *Let T be a type graph and $d\colon (V_T \cup E_T) \to \mathbb{N}$ a function assigning a weight to all nodes and edges of T. The linear weight function w_d as defined above is a well-defined weight function, that is, it is stable under pushouts.*

Now termination analysis works as follows.

Definition 4. *Let T be a type graph and w a weight function for T.*

(i) A rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ is tropically decreasing *with respect to T and w if for each morphism $t_L \colon L \to T$ (where t_L is not necessarily injective), there exists a morphism $t_R \colon R \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$ and $w(t_L) > w(t_R)$. A rule ρ is* tropically non-increasing *if the same condition applies, except that $w(t_L) \geq w(t_R)$.*

(ii) A rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ is arctically decreasing *with respect to T and w if for each morphism $t_R \colon R \to T$ (where t_R is not necessarily injective), there exists a morphism $t_L \colon L \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$ and $w(t_L) > w(t_R)$. A rule ρ is* arctically non-increasing *if the same condition applies, except that $w(t_L) \geq w(t_R)$.*

Note that in the definition above the morphisms into type graphs are not necessarily injective, although the morphisms used in rules and matches are. This is intended, because only for subgraphs T' of the typegraph T (or graphs isomorphic to such a subgraph) there exists an injective morphism from T' to T, and restricting to the subgraphs of T is clearly undesired.

The names *tropical* and *arctic* stem from string rewriting, where analogous termination arguments use tropical and arctic semi-rings as evaluation algebras; see for example [8].

Theorem 1. *Let $\mathcal{L}$ be a set of graphs and let $\mathcal{R}$ be a graph transformation system. Furthermore, let T be a type graph which is closed under $\mathcal{R}$, such that $\mathcal{L} \subseteq \mathrm{L}(T)$, and let w be a weight function of T.*

Finally, let $\mathcal{R}' \subseteq \mathcal{R}$ be such that one of the following conditions holds:

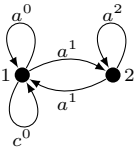
- *all rules of $\mathcal{R}'$ are tropically decreasing with respect to T and w, and all rules of $\mathcal{R} \setminus \mathcal{R}'$ are tropically non-increasing with respect to T and w; or*
- *all rules of $\mathcal{R}'$ are arctically decreasing with respect to T and w, and all rules of $\mathcal{R} \setminus \mathcal{R}'$ are arctically non-increasing with respect to T and w.*

Then $\mathcal{R}$ is $\mathcal{L}$-terminating if and only if $\mathcal{R} \setminus \mathcal{R}'$ is $\mathcal{L}$-terminating.

The above theorem allows to "remove" rules from a graph transformation system, concluding termination of the complete system from termination of a subset of the rules (this is called *relative termination* in the literature). In the case that $\mathcal{R} = \mathcal{R}'$ termination follows directly. Otherwise, a new termination argument for the simpler system is sought. Thus, we obtain iterative termination proofs (see Sect. 3.5 for an example of such an iterative proof).

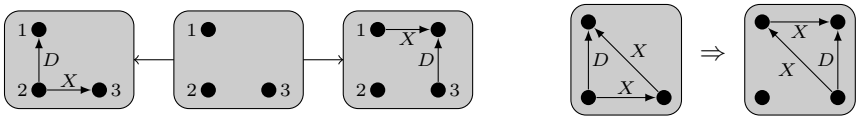
Example 2. See the graph transformation system from Ex. 1. This system is terminating because of the dangling edge condition, because the number of nodes without a c-loop strictly decreases in each transformation step. Now, we want to use the type graph argument to prove this. Since there is only one rule, it is sufficient to show that it is decreasing with respect to some type graph T and some weight function.

We take the type graph and weight function d displayed on the right. The superscripts of the edge labels denote the d-value of the edge, while $d(v) = 0$ for all nodes v. Because of the "flower structure" on the left node, every graph consisting of a- and c-labeled edges can be mapped into this type graph, so we can show uniform termination.

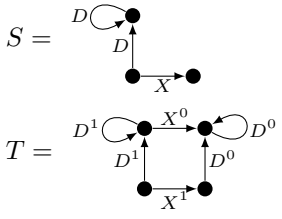


We arctically evaluate the rule with respect to the given type graph. There are a number of morphisms from the right-hand side into the type graph. For each of them a corresponding morphism of greater weight from the left-hand side into the type graph, which agrees on the interface nodes, can be found. For example, one possibility is to map nodes 1 and 2 of the right-hand side of the rule to node 2 of the type graph and the middle node of the right-hand side to node 1 of the type graph; the edges are mapped accordingly (weight: 2). A compatible morphism from the left-hand side to the type graph maps all nodes to node 2 of the type graph (weight: 4). Thus, the system terminates by Theorem 1.

Example 3. Let $\Lambda = \{D, X\}$. Consider the graph transformation system which consists of the rule depicted below on the left. This graph transformation system is not uniformly terminating, as is witnessed by the step displayed on the right, the target of which contains its source as a subgraph:



It is L(S)-terminating, however, where S is given on the right. This follows by considering the weighted type graph T, where the weights of the edges are again given by the superscripts. For each morphism from the left-hand side into T, a smaller morphism from the right-hand side into T, which agrees on the interface nodes, can be found, so the rule is tropically decreasing. Since T contains S as a subgraph and is closed under the transformation rule, the transformation system is L(S)-terminating by Theorem 1.



3.2 Special Case: Node and Edge Counting

A simple (but weak) termination argument for graph transformation (previously considered in, among others, [1,2]) is the counting of nodes and edges. We consider a somewhat more general variant, *weighted* node and edge counting.

Let $d\colon \Lambda \to \mathbb{N}$ be a function which maps each label to a weight. Then a graph G can be assigned a weight by taking the sum of the weights of all labels occurrences in it: $w(G) = \sum_{e \in E_G} d(\ell_G(e))$. If for each production $L \leftarrow I \rightarrow R$ it holds that $w(L) > w(R)$, then the graph transformation system terminates.

This termination argument is a special case of the weighted type graph argument. As type graph we take the "flower" graph $F_\Lambda = \langle \{v\}, \Lambda, s, t, \ell \rangle$ with for

each $A \in \Lambda$: $s(A) = v$, $t(A) = v$ and $\ell(A) = A$. Now we can use the linear weight function w_d, as defined before Prop. 1. This works with both the tropical and the arctic variant of the weighted type graph termination argument.

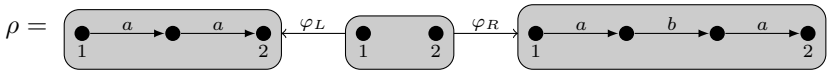
With an even simpler argument in the same style we can view node counting as a special case of the weighted type graph approach.

3.3 Special Case: Match-Bounds

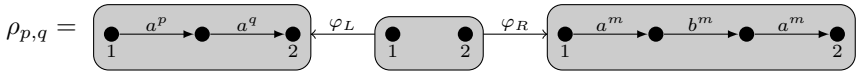
In [3] a method for proving termination of graph transformation systems based on the match-bounds approach for string rewrite systems [7] was introduced. Here, we briefly recapitulate the termination argument of this paper and show that it can be considered as a special case of the weighted type graph approach.

The idea of match-bounds is to annotate the edges with a "creation height", which is a measure of how many previous transformation steps were responsible for creating the edge. In particular, when an occurrence of a left-hand side is replaced by a right-hand side, the new edges are annotated with a creation height which is equal to the smallest creation height of the left-hand side plus one. Now, the termination argument is as follows: if there exists a type graph (with annotated edges) which is closed under the annotated graph transformation system, then the original graph transformation system is terminating. See [3] for more details and formal definitions and proofs.

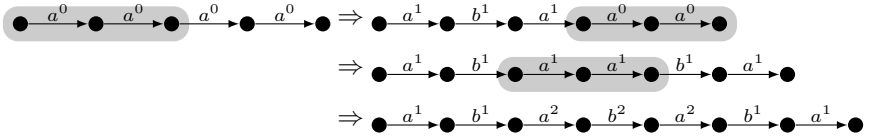
Example 4. Consider the graph transformation consisting of the following rule:



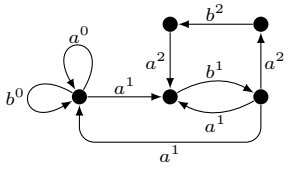
This rule is replaced by the infinitely many annotated rules of the following form, where $m = \min(p, q)+1$. The superscripts denote the creation height annotations of the edges.



The following is an annotated transformation sequence of this system:



The following is a type graph which is closed under the annotated system:



So, by the match-bounds technique, we conclude that the system is terminating.

The match-bounds approach is a special case of the type graph approach, in particular of the tropical variant. The annotated type graph plays the role of the type graph by which graphs are assigned weights. Consider an annotated type graph T with maximum annotation c (in Ex. 4 above, $c = 2$). Let T' be the graph which is equal to T, except that the annotations are removed from the labels. As weights we take strings of natural numbers of length $c + 1$, lexicographically ordered. A morphism $\varphi\colon G \to T'$ is now assigned a weight as follows: $w(\varphi) = n_0 \dots n_c$, where n_i is the number of edges in G which are mapped to an edge of T which has annotation i. Analogously to Prop. 1 we can show that this weight function is stable under pushouts and thus well-defined. By construction, all rules are decreasing with respect to this type graph.

3.4 Derivational Complexity

In this subsection we consider the type graph method with linear weight functions.

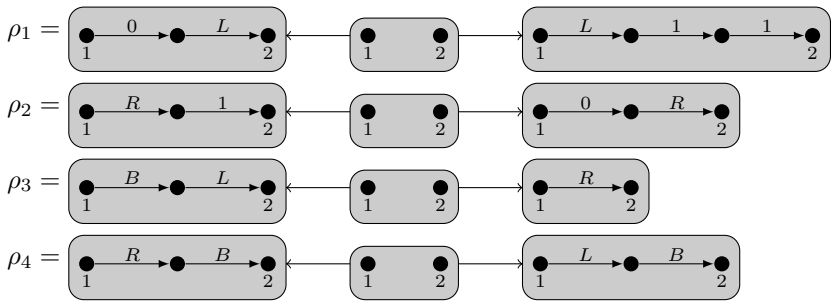
First, we show that transformation sequences of graph transformation systems which can be proved to be terminating with a single application of the type graph technique are linearly bounded (with respect to the size of the initial graph). Because we restrict to linear weight functions, this result is not very surprising.

Secondly, however, we show by example that graph transformation systems that can be proven terminating by a *repeated* application of the type graph technique may even have transformation sequences of exponential length.

Proposition 2. *Let T be a type graph and $d\colon (V_T \cup E_T) \to \mathbb{N}$ a weight function. Furthermore, let $\mathcal{R}$ be graph transformation system such that all rules $r \in \mathcal{R}$ are decreasing with respect to T and w. Then there exists a $c \in \mathbb{N}$ such that for each $\mathcal{R}$-reduction sequence $\boldsymbol{G} = G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \cdots$ it holds that $|\boldsymbol{G}| \leq c \cdot |G_0|$.*

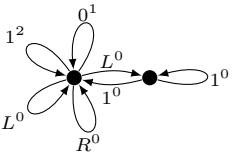
Although proving termination by a single type graph implies a linear reduction bound, by repeating the type graph argument we can show termination of systems with even exponential reduction bounds, as the following example shows:

Example 5. Consider the following graph transformation system, adapted from cycle rewriting [12], which consists of the following graph transformation rules:



This graph transformation system has exponential derivational complexity: starting from a string graph of the form $B^n R 1 B$ the string graph of the form $L 1^{2^n} B$ is reachable in exponentially many steps.

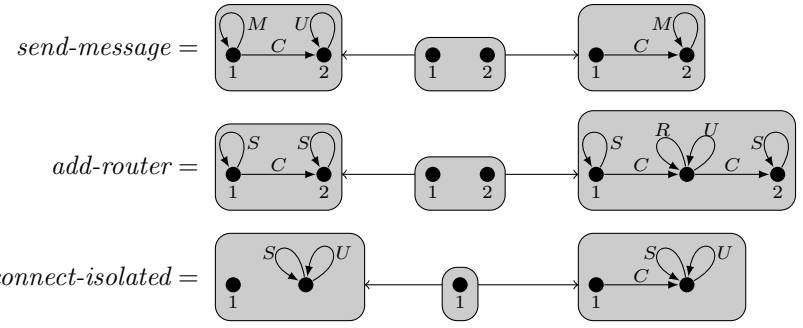
We can show that this graph transformation system terminates. First, ρ_3 can be removed by counting B's, then ρ_4 by counting R's – as shown in Sect. 3.2, label counting is an instance of the type graph technique. For ρ_1 and ρ_2, which indeed by itself have a linear derivational complexity, we construct the type graph on the right (using tropical evaluation).



Note that, although the example derives from a similar example in cycle rewriting, the termination proof is stronger, since it shows uniform termination for all possible start graphs.

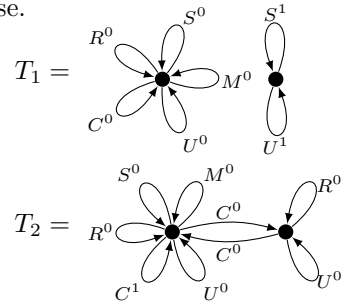
3.5 Detailed Example: Ad-hoc Routing Protocol

We conclude the paper by demonstrating the weighted type graph technique on a simple ad-hoc routing protocol in a dynamically changing network. A message (M) traverses a network of servers (S), routers (R) and directed connections (C) between them. The message can only be sent to unvisited (U) nodes. In addition, rules which modify the network's layout are active. The graph transformation system which models the protocol consists of the following rules:



Note that, due to the dangling edge condition, the *connect-isolated* rule cannot be applied to patterns where the right node is connected to any other nodes; in fact this condition ensures termination in this case.

The fact that this system is uniformly terminating can be shown using a sequence of weighted type graph arguments. First, we arctically evaluate the rules with respect to T_1 (again, the superscripts denote the weights of the edges). For *connect-isolated*, the right-hand side can only be mapped to the flower structure on the left node (weight 0), while for the left-hand side the nodes with the S- and the U-loop can be



matched to the right node (weight 2). For the other rules, both the left-hand side and the right-hand side can only be mapped to the flower, so all morphisms have weight 0. Thus, *connect-isolated* can be removed.

Now, we tropically evaluate *send-message* and *add-router* with respect to T_2. In both cases, the left-hand side can only be mapped to the flower structure on the left node (weight: 1). But for *add-router*, a compatible morphism from the right-hand side to T_2 can be found which maps the middle node to the right node of T_2 (weight: 0). Thus, *add-router* can be removed.

Finally, the system consisting of only *send-message* is terminating because the number of U-edges strictly decreases in each step.

4 Termination Analysis via Cycle Rewriting

In this section we consider graph transformation systems of a specific string-like form, and show that termination of such systems reduces to termination of cycle rewriting [12], which is a variant of string rewriting.

The result of this section has both theoretical and practical relevance. From a theoretical point of view, it shows that graph transformation shares some characteristics (with respect to termination) with a string-based rewriting formalism, which motivates considering cycle rewriting as a step between graph transformation and term rewriting. In fact, [12] uses an approach similar to the type graph method for proving termination of cycle rewrite systems. For cycle rewrite systems finding termination arguments is easier because of the more restricted format.

From a practical point of view it is useful for proving termination of actual graph transformation systems. Although graph transformation systems which consist only of string-like rules are rare "in the wild", such rules do occur quite often. We can try to use the weighted type graph method to first remove the non-string-like rules from the system, and then, when only string-like rules are left, apply the easier (faster) techniques for cycle rewriting to finish the termination proof.

4.1 Cycle Rewriting with Graph Transformation Systems

Cycle rewriting, introduced in [12], is a variant of string rewriting where strings are considered modulo cyclic shift. Let Σ be an alphabet (that is, a finite set of symbols). Fur $u, v \in \Sigma^*$, we write $u \sim v$ if there are $u_1, u_2 \in \Sigma^*$ such that $u = u_1u_2$ and $v = u_2u_1$. A *cycle rewrite rule* is a pair $\langle \ell, r \rangle$ of strings, written $\ell \to r$, and a *cycle rewrite system* is a finite set $\mathcal{R}$ of cycle rewrite rules. A string s rewrites to a string t (written $s \Rightarrow_{\mathcal{R}} t$) if there are a rule $\ell \to r \in \mathcal{R}$ and $s', t', x, y \in \Sigma^*$ such that $s \sim s'$, $t \sim t'$, $s' = x\ell y$ and $t' = xry$.

First, we have to encode cycle rewrite systems as graph transformation systems. The natural way to do this is to represent a string by a "string graph", a graph consisting of a single path, and a cycle by a "cycle graph", a graph consisting of a single cycle.

Let $w = a_1 \cdots a_n$ (where $a_i \in \Sigma$ for $1 \leq i \leq n$) be a string. We define $path(w) = \langle V, E, src, tgt, lab \rangle$, where $V = \{v_0, \ldots, v_n\}$, $E = \{e_1, \ldots, e_n\}$, $src(e_k) = v_{k-1}$, $tgt(e_k) = v_k$ and $lab(e_k) = a_k$ for $1 \leq k \leq n$.

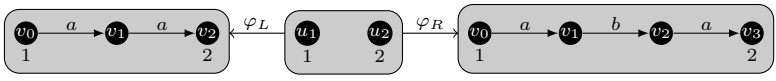
Furthermore, we define $cycle(w) = \langle V, E, src, tgt, lab \rangle$, where $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_n\}$, $src(e_1) = v_n$, $tgt(e_1) = v_1$, $lab(e_1) = a_1$, and $src(e_k) = v_{k-1}$, $tgt(e_k) = v_k$ and $lab(e_k) = a_k$ for $2 \leq k \leq n$.

To encode a cycle rewrite rule, it is natural to encode the left-hand side and the right-hand side with a string graph, and associate via the interface the left-most node and right-most node of the left-hand side with left-most node and right-most node of the right-hand side, respectively. Still, there are two natural choices of what to do with the middle nodes of the left-hand side. Either they are deleted (have no corresponding node in the right-hand side) or they are kept (for each middle node we add an isolated node to the right-hand side). First, we show that under the first encoding termination is preserved. Then, we extend the result to the second encoding by showing that in this case the isolated nodes can be removed from the right-hand side without affecting termination.

Let $\rho = \ell \to r$ be a cycle rewrite rule. We define $graph(\rho) = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$, where $L = path(\ell)$ and $R = path(r)$; $I = \langle \{u_1, u_2\}, \varnothing, \varnothing, \varnothing, \varnothing \rangle$; $\varphi_L(u_1) = v_0$ and $\varphi_L(u_2) = v_{|\ell|}$; and $\varphi_R(u_1) = v_0$ and $\varphi_R(u_2) = v_{|r|}$.

For a cycle rewrite system $\mathcal{R}$, $graph(\mathcal{R})$ consists of the graph transformation rules corresponding to its rules: $graph(\mathcal{R}) = \{graph(\rho) \mid \rho \in \mathcal{R}\}$.

Example 6. Let the rule $aa \to aba$ be given. The corresponding graph transformation rule is:

$\rho =$ 

Above, the white labels inside the nodes are the names given to the nodes in the definitions above, while the numbers below represent the morphisms φ_L and φ_R.

Termination of graph transformation systems of this specific form can now be reduced to termination of cycle rewriting – which, because of the more restricted form, is in some cases slightly easier. Techniques for proving termination of cycle rewrite systems were developed in [12]. Note that here we consider *uniform termination*: the rules in $graph(\mathcal{R})$ are applied to arbitrary graphs, not only to cycles. Hence, proving such a result is non-trivial since we have to derive termination on all graphs from the fact that the rules terminate on all possible cycles.

Theorem 2. *Let $\mathcal{R}$ be a cycle rewrite system. $\mathcal{R}$ is terminating if and only if $graph(\mathcal{R})$ is terminating.*

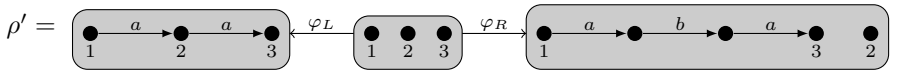
Since termination of cycle rewriting is undecidable (proved in [12]; basically it is a consequence of the undecidability of termination of string rewriting), we obtain an alternative proof of the following result (previously proved in [9]) as a small bonus:

Corollary 1. *Uniform termination of graph transformation systems is undecidable.*

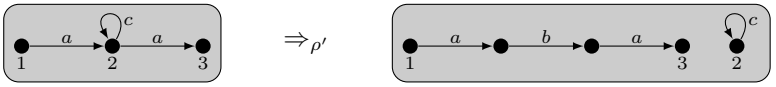
4.2 Removing Isolated Nodes from the Right-Hand Side

Above, we mentioned two different encodings for string rewrite rules: either the middle nodes are deleted or they are maintained by adding corresponding isolated nodes to the left-hand side. (In fact, we can make this choice independently for each middle node.) Which of the encodings we adopt is significant, as the following example shows.

Example 7. Consider the rule ρ of Ex. 6 and the following rule ρ':



The rules ρ and ρ', although similar, generate a different transformation relation, even if we ignore isolated nodes. The following transformation step is possible with ρ':



Because of the dangling edge condition, there are no ρ-steps at all from the source of the above step.

It turns out that, for a class of graph transformation systems which includes "string-like" systems, isolated nodes can be removed from the right-hand sides without affecting termination of the system.

Let $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ be a graph transformation rule with a discrete interface I. The graph transformation rule $\mathit{deiso}(\rho)$ is obtained by removing from R all isolated nodes, removing from I all nodes which are mapped by φ_R to an isolated node, and restricting φ_L and φ_R to the new smaller I. For a graph transformation system $\mathcal{R}$, $\mathit{deiso}(\mathcal{R}) = \{\mathit{deiso}(\rho) \mid \rho \in \mathcal{R}\}$.

Proposition 3. *Let $\mathcal{R}$ be a graph transformation system and $\mathcal{C}$ a cycle rewrite system. If $\mathit{deiso}(\mathcal{R}) = \mathit{graph}(\mathcal{C})$, then $\mathcal{R}$ is terminating if and only if $\mathcal{C}$ is terminating.*

Example 8. Consider the graph transformation system $\mathcal{R}$ consisting of the rule ρ' of Ex. 7. The graph transformation system $\mathit{deiso}(\mathcal{R})$ is (isomorphic to) the graph transformation system of Ex. 6 (consisting of ρ), which is $\mathit{graph}(aa \to aba)$. Since $\{aa \to aba\}$ is a terminating cycle rewrite system (see [12]), $\mathcal{R}$ is terminating by Prop. 3.

5 Implementation

A prototype Java-based tool, called *Grez*, has been written, which implements, among others, the termination techniques presented in this

paper. The tool may be downloaded from the following web page: `www.ti.inf.uni-due.de/research/tools/grez`.

The tool concurrently runs a number of algorithms. Each of the algorithms tries to prove uniform termination of a graph transformation system. As soon as one of the algorithms successfully finds a termination proof, all algorithms are interrupted and the proof is reported to the user. If the found termination proof is relative, that is, termination of a smaller system must still be proved, this procedure is repeated.

The key algorithms implemented in the tool are:

- *Weighted type graphs.* For all weighted type graphs with a user-specified number of nodes and a user-specified maximum weight it is checked whether they prove the graph transformation system terminating.
- *Cycle rewriting.* If the graph transformation system is of the correct form, it is translated into a cycle rewrite system. Then, the tool TORPAcyc, developed in the context of [12], is run as an external program to prove termination.
- *Node counting.* It is checked whether all left-hand sides have more nodes than the corresponding right-hand side.
- *Label counting.* For all one- and two-element subsets of the labels, and for the set of all labels, it is checked that all left-hand sides have more edges labeled with such a label than the corresponding right-hand side.
- *Match bounds.* The algorithm of [3] is implemented. Additionally, nodes are optionally merged according to two rules: *target-merging*: if a node of the type graph has two outgoing edges with the same label, the target nodes of the edges are merged; and *source-merging*: if a node of the type graph has two incoming edges with the same label, the source nodes of the edges are merged.

Note that the last three techniques are subsumed by the weighted type graph technique. However, specialized algorithms make them often substantially faster than the type graph technique.

We ran the tool on the (uniformly terminating) examples of this paper, using a Linux workstation with a 2.67 Mhz, 4-core CPU. To be better able to compare run times, we only enabled the *weighted type graphs* algorithm (using linear weight functions and both tropical and arctic evaluation). Note that with a single running algorithm the tool is essentially single-threaded. The parameters we used were: generate weighted type graphs with at most two nodes and a maximum weight of 1 (that is: 0 or 1). The run times are listed in Table 1. Note that the ad-hoc routing protocol takes significantly longer than the other examples; this is due to the larger number of labels and thus larger number of type graphs that the exhaustive algorithm needs to generate. It is a direction for further research to develop better heuristical algorithms to find suitable weighted type graphs.

Table 1. Run times (in seconds) of running the weighted type graph finder of *Grez* on various examples of the paper (average of 5 tries)

Example	**Run time (s)**
Adding c-loop (Ex. 2)	0.20
$graph(aa \to aba)$ (Ex. 4 and Ex. 6)	0.15
Exponential transformation complexity (Ex. 5)	0.93
Ad-hoc routing protocol (Sec. 3.5)	144.8

6 Related Work

As mentioned in the introduction, various other works concern themselves with termination of graph transformation, more specifically, of graph transformation as a model transformation formalism.

The paper [2] considers high-level replacement units (HLRU), which are transformation systems with external control expressions. The paper introduces a general framework for proving termination of such HLRUs, but the only concrete termination criteria considered are node and edge counting, which are subsumed by the weighted type graph method of this paper (see Sect. 3.2).

In [6] layered graph transformation systems are considered, which are graph transformation systems where interleaving creation and deletion of edges with the same label is prohibited and creation of nodes is bounded. The paper shows such graph transformation systems are terminating.

The paper [10] simulates a graph transformation system by a Petri-net. Thus, the presence of edges with certain labels and the causal relationships between them are modeled, but no other structural properties of the graph. The paper uses typed graph transformation systems; thus, a type graph is used but, unlike in our weighted type graph method, it is fixed by the graph transformation system.

Finally, [3] was one of the inspirations for this paper. As shown in Sect. 3.3, its termination argument is subsumed by the weighted type graph technique.

7 Conclusion and Further Research

We introduced the weighted type graph technique for proving termination of graph transformation systems in the double pushout approach. The technique uses type graphs to assign weights to graphs that strictly decrease in each graph transformation step. It is simple and elegant and supports both uniform and non-uniform termination. Two simpler techniques, weighted edge and node counting (Sect. 3.2) and match bounds (Sect. 3.3) are subsumed by the technique.

Secondly, we showed that uniform termination of graph transformation systems of a specific form can be reduced to uniform termination of cycle rewriting, a form of rewriting related to string rewriting. This makes it possible to use the

stronger termination algorithms of cycle rewriting for graph transformation systems. As a bonus, it provides an alternative proof of the undecidability of the termination problem of graph transformation systems.

Although all theorems have been stated and proved for (binary) multigraphs, a generalization to hypergraphs would be trivial. On the other hand, transferring the results to other graph transformation formalisms is harder. For example, in the single pushout approach, the graph transformation system corresponding to the one of Ex. 1 is non-terminating, so the result of Ex. 2 (in which it is proved that this system is terminating) shows that the weighted type graph technique cannot be transferred one-to-one to single pushout graph transformation. It is left as future research to find similar arguments for the single pushout approach and other formalisms.

Another direction for further research is to allow for graph transformation systems with negative application conditions or more general application conditions. Note, however, that the implicit negative application condition of double pushout graph transformation, the dangling edge condition, can in some cases already be handled (see Ex. 2).

Finally, for interesting real-world applications, it would be interesting to generalize the technique to more expressive methods of specifying the initial graph languages, so that we can, for example, restrict to trees or rings of arbitrary size (both graph languages cannot be expressed by a type graph).

References

1. Aßmann, U.: Graph rewrite systems for program optimization. ACM Transactions on Programming Languages and Systems 22(4), 583–637 (2000)
2. Bottoni, P., Hoffman, K., Presicce, F.P., Taentzer, G.: High-level replacement units and their termination properties. Journal of Visual Languages and Computing (2005)
3. Bruggink, H.J.S.: Towards a systematic method for proving termination of graph transformation systems. In: Proceedings of GT-VC 2007 (2007)
4. Corradini, A., Montanari, U., Rossi, F.: Graph processes. Fundamenta Informaticae 26(3/4), 241–265 (1996)
5. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. 1, World Scientific (1997)
6. Ehrig, H., Ehrig, K., de Lara, J., Taentzer, G., Varró, D., Varró-Gyapay, S.: Termination criteria for model transformation. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 49–63. Springer, Heidelberg (2005)
7. Geser, A., Hofbauer, D., Waldmann, J.: Match-bounded string rewriting. Applicable Algebra in Engineering, Communication and Computing 15(3-4), 149–171 (2004)
8. Koprowski, A., Waldmann, J.: Arctic termination ... Below zero. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 202–216. Springer, Heidelberg (2008)
9. Plump, D.: Termination of graph rewriting is undecidable. Fundamenta Informaticae 33(2), 201–209 (1998)

10. Varró, D., Varró–Gyapay, S., Ehrig, H., Prange, U., Taentzer, G.: Termination analysis of model transformations by petri nets. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 260–274. Springer, Heidelberg (2006)
11. Zantema, H.: Termination of term rewriting by semantic labelling. Fundementa Informaticae 24, 89–105 (1995)
12. Zantema, H., König, B., Bruggink, H.J.S.: Termination of cycle rewriting. In: Dowek, G. (ed.) RTA-TLCA 2014. LNCS, vol. 8560, pp. 476–490. Springer, Heidelberg (2014)

Tropical Two-Way Automata

Vincent Carnino[1] and Sylvain Lombardy[2]

[1] LIGM - Laboratoire d'informatique Gaspard-Monge
Université Paris-Est Marne-la-Vallée, France
Vincent.Carnino@univ-mlv.fr
[2] LaBRI - Laboratoire Bordelais de Recherche en Informatique
Institut Polytechnique de Bordeaux, France
Sylvain.Lombardy@labri.fr

Abstract. In this paper we study two-way min-plus automata. We prove that two-way distance automata are equivalent to one-way distance automata. In the second part of the paper we show that, with general min-plus semirings, it is decidable whether every accepted word has a weight different from $-\infty$ and that, in contrast, it is undecidable whether there exists a word accepted with a weight different from $-\infty$.

1 Introduction

Min-plus automata have attracted much attention for three decades. $\mathbb{N}$-min-plus is one of the simplest extensions of the Boolean semiring and min-plus automata are therefore a very natural extension of automata with various applications in natural language processing or optimization. They are indeed very powerful tools and take part in some very important results like star height.

We study here two-way min-plus automata. When the weights are non negative, we extend the classical result [8,7] that states that a two-way finite automaton is equivalent to a one-way finite automaton.

In the second part, we show that in general some words may be accepted in some two-way automata by a family of runs whose the infimum over weights is $-\infty$. In this case, the behaviour of the automata may be not rational. We prove that the existence of such accepted words is decidable. In contrast, we prove that given a $\mathbb{Z}$-min-plus automaton, it is undecidable whether there exists a word accepted with a finite weight.

2 Tropical Two-Way Automata

2.1 Automata and Runs

An alphabet is a finite set of letters; for every alphabet A, we assume that there exist two fresh symbols $\vdash$ and $\dashv$ that are marks at the beginning and the end of the tapes of automata. We denote $A_{\vdash}$ the alphabet $A \cup \{\vdash, \dashv\}$. For every word w in A, $w_{\vdash}$ is the word in $A_{\vdash}$ equal to $\vdash w \dashv$.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 195–206, 2014.

Tropical automata are instances of automata weighted by a semiring. For every additive submonoid $\mathbb{K}$ of $\mathbb{R}$, we can define the min-plus semiring $\mathcal{K} = (\mathbb{K} \cup \{\infty\}, \min, +)$. For instance, from $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}_+$, we can respectively define the min-plus semirings $\mathcal{N}$, $\mathcal{Z}$ and $\mathcal{R}_+$. Notice that in a $\mathcal{K}$-automaton, only weights in $\mathbb{K}$ appear (∞ is an algebraic way to specify the absence of transition).

In the sequel, we call *distance automaton* every min-plus automaton with non negative weights. Hence, $\mathcal{N}$-automata and $\mathcal{R}_+$-automata are distance automata.

In some applications, the semiring $\mathbb{P} = ([0;1], \max, \cdot)$ is used (the weight of a path is in this case the product of the probabilities of the path and the weight of a word is the weight of the run on this word with the highest probability). The application $x \longmapsto -\log x$ is actually an isomorphism from $\mathbb{P}$ onto $\mathcal{R}_+$. Every result on distance automata is therefore valid for $\mathbb{P}$-automata.

One-way and two-way $\mathcal{K}$-automata share a part of their definition. A $\mathcal{K}$-automaton is a tuple $\mathcal{A} = (Q, A, E, I, T)$ where Q is a finite set of state, A is a finite alphabet, and I and T are partial functions from Q to $\mathbb{K}$. The support of I, $\underline{I}$, is the set of initial states of $\mathcal{A}$, and the support of T, $\underline{T}$, is the set of final states of $\mathcal{A}$.

The definition of transitions differ. In a two-way $\mathbb{K}$-automaton, E is a partial function from $Q \times (A_{\vdash} \times \{-1, +1\}) \times Q$ into $\mathbb{K}$ and the support of E, $\underline{E}$, is the set of transitions of $\mathcal{A}$. Moreover, the intersection of $\underline{E}$ and $Q \times (\{\vdash\} \times \{-1\} \cup \{\dashv\} \times \{1\}) \times Q$ must be empty.

Let t be a transition in $\underline{E}$; if $t = (p, a, d, q)$, we denote $\sigma(t) = p$, $\tau(t) = q$, $\lambda(t) = a$, $\delta(t) = d$. On figures, the value of δ is represented by a left (-1) or right (+1) arrow. For instance, if $t = (p, a, -1, q)$ and $E_t = k$, we draw $p \xrightarrow{a, \leftarrow | k} q$.

In a one-way $\mathbb{K}$-automaton, E is a partial function from $Q \times A \times Q$ into $\mathbb{K}$, and the support of E, $\underline{E}$, is the set of transitions of $\mathcal{A}$.

Let t be a transition in $\underline{E}$; if $t = (p, a, q)$, we denote $\sigma(t) = p$, $\tau(t) = q$, $\lambda(t) = a$.

Definition 1. *Let $w = w_1 \dots w_n$ be a word of A^*, we set $w_0 = \vdash$ and $w_{n+1} = \dashv$. A* configuration *of $\mathcal{A}$ on w is a pair (t, i) where i is in $[0; n+1]$ and t is a transition of $\mathcal{A}$ with $\lambda(t) = w_i$. A computation (or run) ρ of $\mathcal{A}$ on w is a finite sequence of configurations $((t_1, i_1), \dots, (t_k, i_k))$ such that :*

- *$i_1 = 1$, $i_k = n$, $\delta(t_1) = \delta(t_k) = 1$, $\sigma(t_1)$ is in $\underline{I}$ and $\tau(t_k)$ is in $\underline{T}$;*
- *for every j in $[0; k-1]$, $\sigma(t_{j+1}) = \tau(t_j)$ and $i_{j+1} = i_j + \delta(t_j)$.*

Example 1. Let $\mathcal{A}_1$ be the two-way $\mathcal{N}$-automaton of Figure 1. The automaton checks by a left-right reading the parity of the length of each subsequence of repetitions of $'a's$; if it is odd, a right-left reading computes the length of the block; otherwise the automaton goes to the next block of $'a's$.

A run of the $\mathcal{N}$-automaton $\mathcal{A}_1$ over the word *abaaba* is represented on Figure 2. The weight of this run is equal to 2, since there are two odd subsequences of $'a's$ in the string, and each id of length 1.

The weight of such a computation, denoted by $|\rho|$, is $I(\sigma(t_1)) + \sum_{j=1}^{k} E(t_j) + T(\tau(t_k))$. The weight of w in $\mathcal{A}$, denoted by $\langle |\mathcal{A}|, w \rangle$, is the infimum of the weights

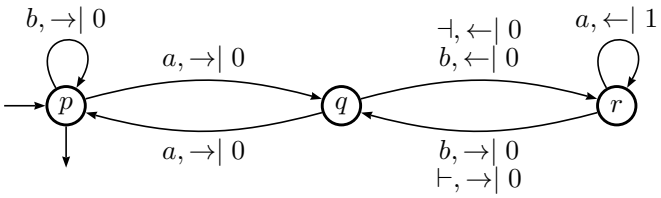


Fig. 1. The two-way $\mathcal{N}$-automaton $\mathcal{A}_1$

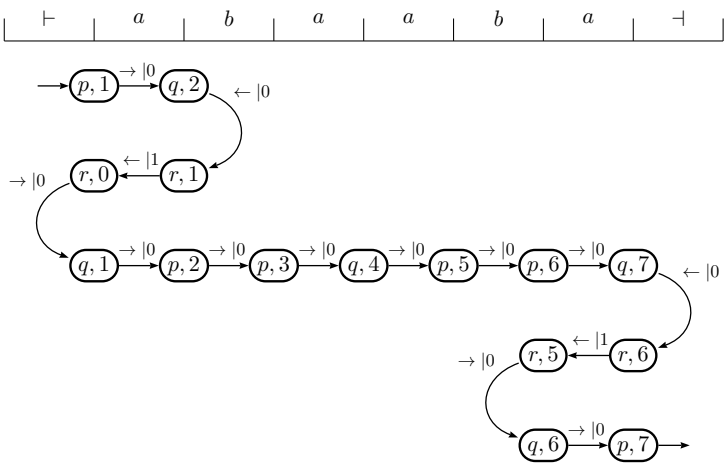


Fig. 2. A run of $\mathcal{A}_1$ over the word *abaaba*

of all the runs with label w in $\mathcal{A}$. Notice that there may be an infinite number of computations with the same label w; in this case the infimum may not belong to $\mathcal{K}$; actually, $\mathcal{K}$ can always be embedded into the semiring $(\mathbb{R} \cup \{-\infty, \infty\}, \min, +)$ (with $-\infty + \infty = \infty$) where the infimum of every family is always defined.

2.2 δ-normalization

Definition 2. *Let $\mathcal{A}$ be a two-way $\mathbb{K}$-automaton.*
If, for each state p of $\mathcal{A}$, every outgoing transition from p has the same direction, then $\mathcal{A}$ is δ-local.
If $\mathcal{A}$ is δ-local and, for each state p of $\mathcal{A}$, every transition arriving at p has the same direction, then $\mathcal{A}$ is δ-normalized.

If $\mathcal{A}$ is a δ-local automaton, for every state p in Q, we set $\delta_{\mathbf{O}}(p) = \delta(t)$, where t is any transition outgoing from p; if it is normalized, we also set $\delta_{\mathbf{I}}(p) = \delta(t)$, where t is any transition incoming to p.

The following proposition is proved in [2].

Proposition 1. *For every two-way $\mathbb{K}$-automaton, there exists an equivalent δ-local two-way $\mathbb{K}$-automaton.*

To make a two-way automaton $\mathcal{A}$ δ-local, a *covering* of $\mathcal{A}$ is built: every state p with outgoing transitions with different directions is split into two states p_+ and p_- that have the same incoming transitions as p, transitions outgoing from p_+ (*resp.* p_-) are the transitions outgoing from p with direction $+1$ (*resp.* -1).

The dual construction consists in splitting the states to separate incoming transitions with different directions. Applied to a δ-local two-way automaton, it results in a δ-normalized automaton.

Example 2. Figure 3 shows the conversion of a two-way automaton into a δ-local automaton, and then into a δ-normalized automaton.

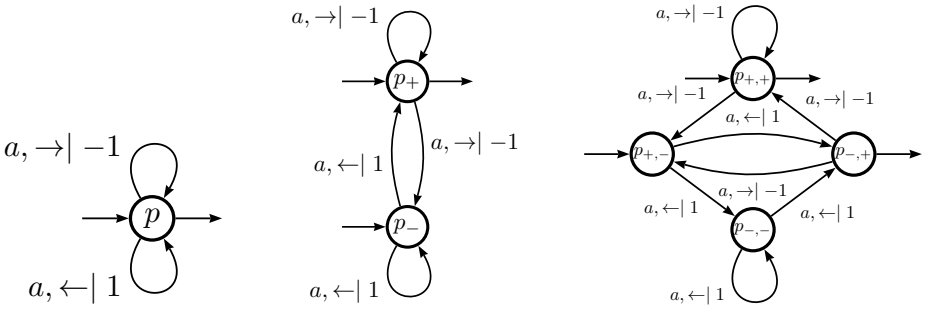


Fig. 3. The two steps of the δ-normalization

Example 3. The automaton $\mathcal{A}_1$ of Figure 1 is not δ-normalized: in states q and r, there are outgoing transitions with direction -1 and others with direction $+1$. The automaton $\mathcal{A}'_1$ of Figure 4 is a δ-normalized equivalent automaton.

2.3 The Slice Automaton

The slice automaton is a one-way automaton that (non-deterministicly) emulates the runs of a two-way automaton. On a given run, for each position of the input, the *slice* of the run is the vector made of the states visited at this position. Every state of the slice automaton is such a vector and there is a transition between two states if the corresponding slices can successively appear in a run of the two-way automaton.

We give here the formal definition of the slices. A more complete description of the slice automaton is given in [2].

Definition 3. *Let $\mathcal{A} = (Q, A, E, I, T)$ be a two-way $\mathbb{K}$-automaton and let $w = w_1 \dots w_k$ be a word. Let $\rho = ((p_0, i_0), \dots (t_n, i_n))$ be a run over w, and j in $[1; k+1]$. Let h be the subsequence of all pairs (p_k, i_k) such that $(i_k, i_{k+1}) =$*

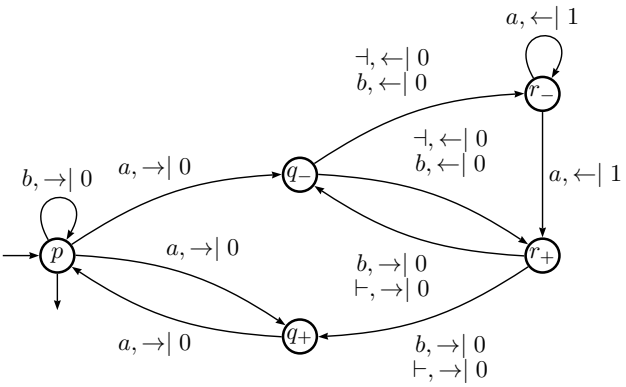


Fig. 4. The δ-normalized two-way $\mathcal{N}$-automaton $\mathcal{A}'_1$

$(j, j+1)$ *or* $(i_{k-1}, i_k) = (j, j-1)$. *The* j*-th* slice *of* ρ *is the vector* $s^{(j)}$ *of states obtained by the projection of the first component of each pair of* h.
The signature $S(\rho)$ *of* ρ *is the sequence of its slices.*

The slices we define here are not exactly the *crossing sequences* defined in [8].

Example 4. The vector $\begin{bmatrix} q \\ r \\ p \end{bmatrix}$ is the second (and the seventh) slice of the run of Figure 2. The signature of this run is:

$$\begin{pmatrix} p & q & & p & q \\ r, & r, & p, q, p, & r, & r \\ q & p & & q & p \end{pmatrix}. \tag{1}$$

Let $\mathcal{A} = (Q, A, E, I, T)$ be a δ-local two-way $\mathbb{K}$-automaton. In order to define a one-way $\mathbb{K}$-automaton from slices we consider the set X of subvectors of slices, that are vectors in Q^* with an odd length; let Y be the vectors in Q^* with an even length.

We define inductively two partial functions $\theta : X \times A \times X \to \mathbb{K}$ and $\eta : Y \times A \times Y \to \mathbb{K}$ by:

$$\begin{aligned} \eta(\varepsilon, a, \varepsilon) &= 0_{\mathbb{K}}, \\ \forall p, q \in Q, \quad \delta_{\mathbf{O}}(p) = 1 \Rightarrow \forall u, v \in Y, \ \theta(pu, a, qv) &= E(p, a, 1, q) + \eta(u, a, v), \\ \eta(u, a, pqv) &= E(p, a, 1, q) + \eta(u, a, v), \\ \delta_{\mathbf{O}}(p) = -1 \Rightarrow \forall u, v \in X, \ \theta(pqu, a, v) &= E(p, a, -1, q) + \theta(u, a, v), \\ \eta(qu, a, pv) &= E(p, a, -1, q) + \theta(u, a, yv). \end{aligned} \tag{2}$$

Since $\mathcal{A}$ is δ-local, if θ is defined on a triple (u, a, v), it is uniquely defined.

For every vector pu in X, pu is initial if p is in $\underline{I}$ and $(\varepsilon, \vdash, u)$ is in $\underline{\eta}$; in this case, we set $\mathcal{I}(pu) = I(p) + \eta(\varepsilon, \vdash, u)$. Likewise, every vector up in X is final if p is in $\underline{T}$ and $(u, \dashv, \varepsilon)$ is in $\underline{\eta}$; in this case, we set $\mathcal{T}(up) = \eta(u, \dashv, \varepsilon) + T(p)$.

Definition 4. *With the above notations, the* slice automaton *of the two-way* $\mathbb{K}$*-automaton* $\mathcal{A} = (Q, A, E, I, T)$ *is the infinite one-way* $\mathbb{K}$*-automaton* $\mathcal{C} = (X, A, \theta, \mathcal{I}, \mathcal{T})$.

3 Two-Way Distance Automata

In two-way automata, in the same computation, there may be two steps where the automaton is in the same state and reads the same letter of the input. In this case we say that the computation contains an *unmoving circuit*.

Definition 5. *Let* $\rho = ((t_1, i_1), \dots, (t_k, i_k))$ *be a run. If there exists* m, n *in* $[1, k]$*, with* $m < n$ *such that* $i_m = i_n$ *and* $\sigma(t_m) = \sigma(t_n)$*, then we say that* $((t_m, i_m), \dots, (t_{n-1}, i_{n-1}))$ *is an* unmoving circuit *of* ρ*. If* ρ *does not contain any unmoving circuit, it is* reduced.

If a run contains unmoving circuits, they can all be removed with a finite number of iterations since the removing of such a circuit leads to a shorter run.

Lemma 1. *If a two-way* $\mathbb{K}$*-automaton admits a run* ρ *which is not reduced, it admits a reduced run with the same label.*

Lemma 2. *Let* $\mathcal{A}$ *be a two-way distance automaton on an alphabet* A*. For each* w *in* A^**,* $\langle |\mathcal{A}|, w\rangle$ *is the weight of a reduced run of* w*.*

Proof. By contradiction, let us suppose that, for a word w, there is no reduced run in $\mathcal{A}$ labeled by w with a minimal weight. Then let $\rho = ((t_1, i_1), \dots, (t_l, i_l))$ be one of the shortest non reduced run labeled by w with a minimal weight. Since it is not reduced, then there exist j and k, with $j < k$, such that $i_j = i_k$ and $\sigma(t_j) = \sigma(t_k)$. Then there exists a run $\rho' = ((t_1, i_1), \dots, (t_{j-1}, i_{j-1}), (t_k, i_k), \dots, (t_l, i_l))$ labeled by w with $|\rho'| \leq |\rho|$ which is a contradiction. □

By Lemma 2 to simulate a two-way distance automaton by a one-way automaton, we only need to simulate reduced runs.

Actually, if a run of a two-way automaton contains an unmoving circuit, the signature of this run contains a vector where two entries with an index with the same parity are equals. In [2], we prove that the restriction of the slice automaton of $\mathcal{A}$ to states labelled by vectors that do not contain this kind of entry results in a finite one-way automaton where every computation corresponds to a reduced computation of $\mathcal{A}$ with the same weight and that every reduced computation of $\mathcal{A}$ has a representative in this finite one-way automaton.

Example 5. From the δ-normalization of $\mathcal{A}_1$, we can build an equivalent one-way $\mathcal{N}$-automaton.

Finally, by Lemma 2,

Proposition 2. *Every two-way distance automaton is equivalent to a one-way distance automaton.*

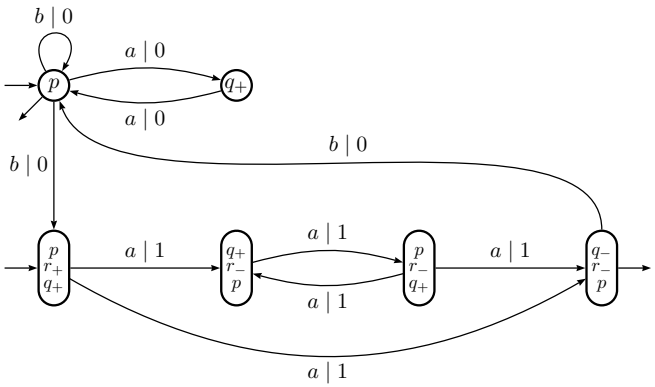


Fig. 5. A one-way $\mathcal{N}$-automaton equivalent to $\mathcal{A}_1$

4 Two-Way Min-plus Automata

In this part, we study two-way automata on min-plus semirings based on non postive submonoids of $\mathbb{R}$. In this case, a word may label an infinite number of paths with an increasingly smaller weight.

We say that a two-way min-plus automaton is *valid* if the weight of every accepted word is finite. We address the problem of deciding whether a two-way min-plus automaton is valid.

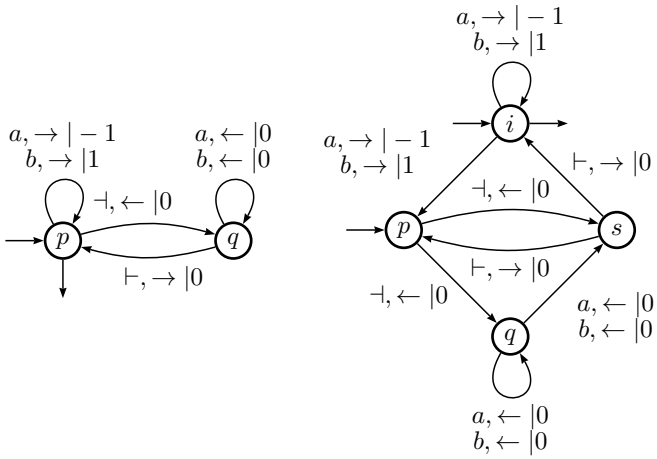


Fig. 6. The two-way $\mathcal{Z}$-automata $\mathcal{A}_2$ and $\mathcal{A}'_2$

Example 6. Let $\mathcal{A}_2$ be the two-way $\mathcal{Z}$-automaton of Figure 6 (left). Every time this automaton reads a word from left to right it computes the difference between

the number of 'b's and the number of 'a's. Since for each accepted word, there can be an unbounded number of left-right reading, if there are more 'a's than 'b's, the weight of runs is not lowerly bounded. Thus, the behaviour of this automaton is only defined for words where the number of 'a's is at most equal to the number of 'b's.

The automaton $\mathcal{A}'_2$ of Figure 6 (right) is the δ-normalization of $\mathcal{A}_2$.

This example shows the following fact.

Proposition 3. *There exist two-way min-plus automata such that the language of words accepted with a finite weight is not rational (or regular).*

Theorem 1. *It is decidable whether a two-way min-plus automaton is valid.*

To prove this theorem, we need to consider another restriction of the slice automaton. Unlike the case of distance automata where we want that unmoving circuits do not appear at all, we want here to detect when unmoving circuits appear, but we want to deal with a finite automaton. So, we allow that each unmoving circuit appears at most once.

To this purpose, we consider the slices that belong to $W = \bigcup_k W_k$ with W_k defined for all k in $\mathbb{N}$ as follows :

$$W_k = \{v \in Q^{2k+1} | \forall p \in Q, \forall s \in [0;1], |\{i \mid v_i = p \text{ and } i \mod 2 = s\}| \leqslant 2\}. \quad (3)$$

We consider the restriction of the slice automaton to W.

Proposition 4. *Let $\mathcal{A}$ be a two-way $\mathbb{K}$-automaton and let $\mathcal{C}$ be the restriction of the slice automaton of $\mathcal{A}$ to W. If $\mathcal{A}$ accepts a run that contains an unmoving circuit with a negative weight, then there exists a run in $\mathcal{A}$ that contains an unmoving circuit with a negative weight and that is mapped into $\mathcal{C}$.*

Proof. Assume that there exist runs of $\mathcal{A}$ that contain at least one unmoving circuit with a negative weight. We chose ρ among these runs with a minimal number of transitions. Let q be the end of the unmoving circuit with negative weight and let q_1 and q_2 be the both occurences of q. If ρ is not mapped into $\mathcal{C}$, there exists a state p that appears (at least) three times in a slice v_1 of ρ (let p_1, p_2 and p_3 be these three occurences); p is the end of two consecutive unmoving circuits.
Different cases occur. If one of the two consecutive unmoving circuits has a negative weight, the other one can be removed to simplify the run. (This case may occur if $p = q$.)
Likewise, if one of the two consecutive unmoving circuits does not intersect the unmoving circuit with negative weight, it can be removed.
The only case that remains is when the run ρ can be decomposed as:

$$\to i \xrightarrow{w_1|k_1} p_1 \xrightarrow{w_2|k_2} q_1 \xrightarrow{w_3|k_3} p_2 \xrightarrow{w_4|k_4} q_2 \xrightarrow{w_5|k_5} p_3 \xrightarrow{w_6|k_6} t \to . \quad (4)$$

In this case we have $k_3 + k_4 < 0$ and the shorter run

$$\to i \xrightarrow{w_1|k_1} p_1 = p_2 \xrightarrow{w_4|k_4} q_2 = q_1 \xrightarrow{w_3|k_3} p_2 = p_3 \xrightarrow{w_6|k_6} t \to \quad (5)$$

contains an unmoving circuit with a negative weight. □

In the automaton $\mathcal{C}$, every run that meets a state in $W \setminus V$ does contain an unmoving circuit. The problem is to detect whether such an unmoving circuit has a negative weight. The solution consists in comparing the weight of this run with the weight of the run without the unmoving circuit. To this purpose, we define an automaton which is a kind of *square* of the automaton $\mathcal{C}$ (*cf.* [1]): it compares paths of $\mathcal{C}$ that differ by unmoving circuits.

We consider first the set $X = \{(x, y, z) \in (Q^*)^3 \mid xz, xyz \in W\}$. An element (x, y, z) in X is *special* if $y_1 = z_1$. From the function θ, we define the (partial) function $\tilde{\theta} : X \times A \times X \longrightarrow \mathbb{K}$ as

$$\tilde{\theta}((x, y, z), a, (t, u, v)) = \theta(xyz, a, tuv) - \theta(xz, a, tv), \tag{6}$$

for every triple $((x, y, z), a, (t, u, v))$ that fulfils one of the three following conditions:

$$\begin{aligned} &(x, a, t), (y, a, u) \text{ and } (z, a, v) \in \underline{\theta}; \\ &(xy_1, a, t), (y_1^{-1} y z_1, a, u) \text{ and } (z_1^{-1} z, a, v) \in \underline{\theta}, y_1 = z_1 \text{ and } \delta_{\mathbf{I}}(y_1) = -1; \\ &(x, a, tu_1), (y, a, u_1^{-1} u v_1) \text{ and } (z, a, v_1^{-1} v) \in \underline{\theta}, u_1 = v_1 \text{ and } \delta_{\mathbf{I}}(u_1) = 1. \end{aligned} \tag{7}$$

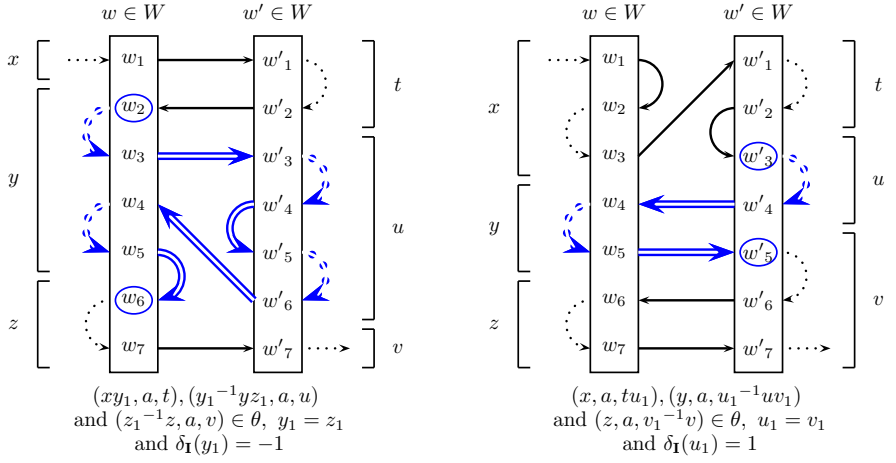


Fig. 7. A valid run over the word $w = w_1 \dots w_k$

Let $X_0 = \{(x, y, z) \in X \mid y = 1\}$. We define on X the relation $(x, y, z) \equiv (x', y', z')$ if and only if $y = y' = 1$ and $xz = x'z'$. In the quotient $\tilde{X}$ of X by $\equiv$, every element which is not in X_0 is the only element of its class, while the quotient of X_0 is isomorphic to W. Moreover, this equivalence is compliant with the definition of $\tilde{\theta}$. Let $\mathcal{P} = (\tilde{X}, A, \tilde{\theta}, J, U)$ be a one-way automaton defined as follows. The transition function is $\tilde{\theta}$; every transition that corresponds to one of

the two last lines of (7) is called a special transition. We set $J(x, y, z) = \theta(0, \vdash, xyz) - \theta(0, \vdash, xz)$ if x is non empty or if x is empty and $y_1 = z_1$ (special initial state). Likewise, $U(x, y, z) = \theta(xyz, \dashv, 0) - \theta(xz, \dashv, 0)$ if z is non empty or if z is empty and $x_1 = y_1$ (special final state).

Every computation in the automaton $\mathcal{P}$ that meets one (and only one) special transition (or special initial or final state) corresponds to two computations in the slice automaton. Each state of the first computation is obtained from every state (x, y, z) of the computation in $\mathcal{P}$ by concatenating x, y and z, while each state of the second computation is given by the concatenation of x and z. This two computations correspond to two runs in $\mathcal{A}$, one with an unmoving circuit, the second one where the unmoving circuit has been removed.

Example 7. From the automaton $\mathcal{A}'_2$ of Figure 6 (right), we can build the automaton $\mathcal{P}_2$ of Figure 8. The states in X_0 are labelled by one vector, and the other ones by three vectors. The special transitions and special initial states are red (there is no special final state). Each run in this automaton that contains a special transition (or initial state) corresponds two runs in $\mathcal{A}'_2$, one with an unmoving circuit, the other one without this circuit. Such a pair of paths in $\mathcal{A}'_2$ may correspond to several paths in $\mathcal{P}_2$, depending where the path with the unmoving circuit is cut. For instance, consider the path $\left(i, \begin{smallmatrix} s \\ i \end{smallmatrix}, \text{-}\right) \xrightarrow{a,-1} \left(\text{-}, \begin{smallmatrix} i \\ q \end{smallmatrix}, i\right) \xrightarrow{a,-1} \left(\text{-}, \begin{smallmatrix} p \\ q \end{smallmatrix}, i\right)$; it corresponds to the path of Figure 9 and to the cut between the two red states. The weight of the path with the unmoving circuit is -4, without the unmoving circuit, it is -2; the difference is -2 which is equal to the weight of the path in $\mathcal{P}_2$.

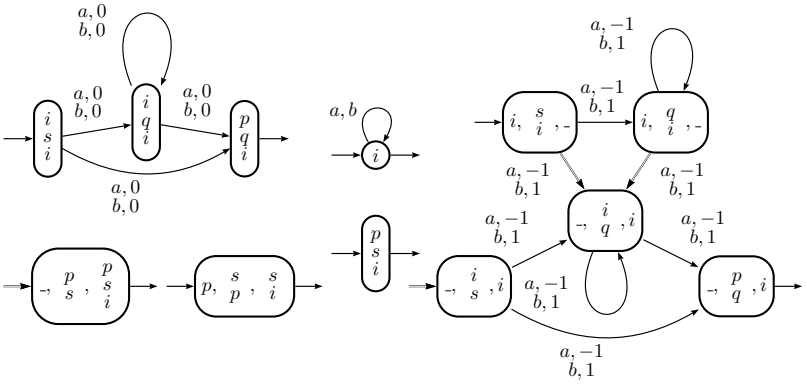


Fig. 8. The one-way automaton $\mathcal{P}_2$

Proposition 5. *Let $\mathcal{A}$ be a two-way min-plus automaton and let $\mathcal{P}$ be the automaton built above. If $\mathcal{A}$ accepts a run that contains an unmoving circuit with a negative weight, then there is a run in $\mathcal{P}$ that meets one special transition (or special initial/final state) with a negative weight.*

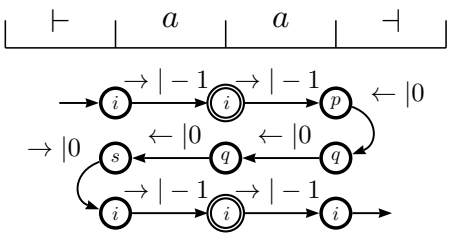


Fig. 9. A path in the automaton $\mathcal{A}_2$ with a negative unmoving circuit

This property can be checked on the automaton $\mathcal{P}$ in polynomial time (see for instance [5]), and this implies Theorem 1.

If the two-way automaton is not valid, it could be interesting to compute an effective description of the language on which the behaviour is defined. By Proposition 3 this language must be non rational; worst, it is undecidable to know whether it is empty.

Theorem 2. *Let $\mathcal{A}$ be a two-way $\mathcal{Z}$-automaton. It is undecidable whether there exists a word w accepted by $\mathcal{A}$ with a finite weight.*

Proof. In [4], it is prove that it is undecidable, given a one-way $\mathcal{Z}$-automaton $\mathcal{B} = (Q, A, E, I, T)$, to know whether there exists a word w whose weight in $\mathcal{B}$ is non negative. Let r be an element which is not in Q and let $\mathcal{A} = (Q \cup \{r\}, A, F, I, T)$ be the two-way $\mathcal{Z}$-automaton defined as follow:

$$\begin{aligned} F =& \{p \xrightarrow{a,\rightarrow|k} q \mid p \xrightarrow{a|k} q \in E\} \\ =& \{p \xrightarrow{\dashv,\leftarrow|k} r \mid p \in \underline{T}, T(p) = k\} \\ =& \{r \xrightarrow{\vdash,\rightarrow|k} p \mid p \in \underline{I}, I(p) = k\} \\ =& \{r \xrightarrow{a,\leftarrow|0} r \mid a \in A\} \end{aligned} \tag{8}$$

For every word w in $\mathcal{A}$, every computation on w is any sequence of computations on w in $\mathcal{B}$. Therefore, the weight of a word w in $\mathcal{A}$ is defined if and only if it has no computation with a negative weight in $\mathcal{B}$, that is if its weight in $\mathcal{B}$ is non negative. □

5 Conclusion

The problem tackled in this paper raises a more general problem on two-way automata. Actually, since the number of computations for a given input can be infinite, a proper definition of the behaviour of a weighted automaton must be forged. It meets some works on the behaviour of one-way weighted automata with ε-transitions (*cf.* [3,6]).

The last part of the paper also introduces some open questions. Despite the fact that the emptiness of the domain of a tropical two-way automaton is undecidable, is it possible to give a usable characterization of this domain?

References

1. Béal, M.-P., Carton, O., Prieur, C., Sakarovitch, J.: Squaring transducers: an efficient procedure for deciding functionality and sequentiality. Theor. Comput. Sci. 292(1), 45–63 (2003)
2. Carnino, V., Lombardy, S.: On determinism and unambiguity of weighted two-way automata. In: AFL 2014 (accepted, 2014)
3. Ésik, Z., Kuich, W.: Finite automata. In: Droste, M., et al. (eds.) Handbook of Weighted Automata, pp. 69–104. Springer (2009)
4. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. Internat. J. Algebra Comput. 4(3), 405–425 (1994)
5. Lombardy, S., Mairesse, J.: Series which are both max-plus and min-plus rational are unambiguous. RAIRO - Theor. Inf. and Appl. 40(1), 1–14 (2006)
6. Lombardy, S., Sakarovitch, J.: The validity of weighted automata. Internat. J. Algebra Comput. 23, 863–913 (2013)
7. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 114–125 (1959)
8. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. 3(2), 198–200 (1959)

Parametric LTL on Markov Chains

Souymodip Chakraborty and Joost-Pieter Katoen*

RWTH Aachen University, Ahornstraße 55, D-52074 Aachen, Germany

Abstract. This paper is concerned with the verification of finite Markov chains against parametrized LTL (pLTL) formulas. In pLTL, the until-modality is equipped with a bound that contains variables; e.g., $\Diamond_{\leqslant x}\ \varphi$ asserts that φ holds within x time steps, where x is a variable on natural numbers. The central problem studied in this paper is to determine the set of parameter valuations $V_{\prec p}(\varphi)$ for which the probability to satisfy pLTL-formula φ in a Markov chain meets a given threshold $\prec p$, where $\prec$ is a comparison on reals and p a probability. As for pLTL determining the emptiness of $V_{>0}(\varphi)$ is undecidable, we consider several logic fragments. We consider parametric reachability properties, a sub-logic of pLTL restricted to next and $\Diamond_{\leqslant x}$, parametric Büchi properties and finally, a maximal subclass of pLTL for which emptiness of $V_{>0}(\varphi)$ is decidable.

1 Introduction

Verifying a finite Markov chain (MC, for short) M against an LTL-formula φ amounts to determining the probability that M satisfies φ, i.e., the likelihood of the set of infinite paths of M satisfying φ. Vardi [1] considered the qualitative version of this problem, that is, does M almost surely satisfy φ, or with positive probability. Together with Wolper, he showed that the qualitative LTL model-checking problem for MCs is PSPACE-complete. The quantitative verification problem – what is the probability of satisfying φ? – has been treated by Courcoubetis and Yannakakis [2]. An alternative algorithm that has a time complexity which is polynomial in the size of the MC and exponential in $|\varphi|$ is by Couvreur *et al.* [3]. Recently, practical improvements have been obtained by Chatterjee *et al.* for verifying the LTL(F,G)-fragment on MCs using generalized deterministic Rabin automata [4].

This paper considers the verification of MCs against *parametric* LTL formulas. In parametric LTL [5] (pLTL, for short), temporal operators can be subscripted by a variable ranging over the natural numbers. The formula $\Diamond_{\leqslant x}\, a$ means that in at most x steps a occurs, and $\Box\Diamond_{\leqslant y}\, a$ means that at every index a occurs within y steps. Note that x and y are variables whose value is not fixed in advance. The central question is now to determine the values of x and y such that the probability of a given MC satisfying the pLTL-formula φ meets a certain

* Currently on sabbatical leave at the University of Oxford, United Kingdom.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 207–221, 2014.

threshold p. This is referred to as the valuation set $V_{\prec p}(\varphi)$ for comparison operator $\prec$. This problem has both a qualitative (threshold > 0 and $= 1$) and a quantitative variant ($0 < p < 1$).

The main results of this paper are as follows. Just as for the setting with Kripke structures [5], it is shown that checking the emptiness of $V_{>0}(\varphi)$ in general is undecidable. We therefore resort to fragments of pLTL. We show that determining $V_{\geqslant p}(\Diamond_{\leqslant x}\, a)$ can be done by searching in a range defined by the *precision* of the input, whereas polynomial time graph algorithms suffice for its qualitative variant. The same applies to formulas of the form $\Box\Diamond_{\leqslant x}\, a$. We provide necessary and sufficient criteria for checking the emptiness of $V_{>0}(\varphi)$ (and $V_{=1}(\varphi)$) for the fragments pLTL(F,X) and pLTL$_{\Diamond}$, and prove that checking these criteria are NP-complete and PSPACE-complete, respectively. We also define a representation of these sets and provide algorithms to construct them.

Related work. The verification of parametric probabilistic models in which certain transition probabilities are given as parameters (or functions thereof) has recently received considerable attention. Most of these works are focused on parameter synthesis: for which parameter instances does a given (LTL or PCTL) formula hold? To mention a few, Han *et al.* [6] considered this problem for timed reachability in continuous-time MCs, Hahn *et al.* [7] and Pugelli *et al.* [8] for Markov decision processes (MDPs), and Benedikt *et al.* [9] for ω-regular properties of interval MCs. Hahn *et al.* [10] provide an algorithm for computing the rational function expressing the probability of reaching a given set of states in a parametric (reward) MDP based on exploiting regular expressions as initially proposed by Daws [11]. Other related work includes the synthesis of loop invariants for parametric probabilistic programs [12]. To the best of our knowledge, verifying parametric properties on MCs has not been considered so far. The closest related works are on combining two-variable FO with LTL for MDPs by Benedikt *et al.* [13] and the computation of quantiles by Ummels and Baier [14].

Organization of the paper. Section 2 presents pLTL and MCs and a first undecidability result. Section 3 considers parametric reachability. Section 4 treats the fragment pLTL(F,X) and Section 5 parametric Büchi properties. Section 6 treats the bounded always-free fragment of pLTL. Section 7 concludes the paper. Full version of the paper can be found in the archive.

2 Preliminaries

Parametric LTL. Parametric LTL extends propositional LTL with bounded temporal modalities, for which the bound is either a constant or a variable. Let *Var* be a finite set of variables ranged over by x, y, and AP be a finite set of propositions ranged over by a and b. Let $c \in \mathbb{N}$. Parametric LTL formulas adhere to the following syntax:

$$\varphi \;::=\; a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \Diamond_{\prec x}\, \varphi \mid \Diamond_{\prec c}\, \varphi$$

where $\prec \in \{=, \leqslant, <, >, \geqslant\}$. A pLTL structure is a triple $(w, i, \boldsymbol{v})$ where $w \in \Sigma^\omega$ with $\Sigma = 2^{AP}$ is an infinite word over sets of propositions, $i \in \mathbb{N}$ is an index, and $\boldsymbol{v} : \mathit{Var} \to \mathbb{N}$ is a variable valuation. Analogously, we consider a valuation $\boldsymbol{v}$ as a vector in $\mathbb{N}^d$, where d for pLTL formula φ is the number of variables occurring in φ. E.g. for $d = 1$, the valuation is just a number v. We compare valuations $\boldsymbol{v}$ and $\boldsymbol{v}'$ as $\boldsymbol{v} \leqslant \boldsymbol{v}'$ iff $\boldsymbol{v}(x) \leqslant \boldsymbol{v}'(x)$ for all x. Let $w[i]$ denote the i-th element of w. The satisfaction relation $\models$ is defined by structural induction over φ as follows:

$$\begin{array}{ll}
(w, i, \boldsymbol{v}) \models a & \text{iff } a \in w[i] \\
(w, i, \boldsymbol{v}) \models \neg\, \varphi & \text{iff } (w, i, \boldsymbol{v}) \not\models \varphi \\
(w, i, \boldsymbol{v}) \models \varphi_1 \wedge \varphi_2 & \text{iff } (w, i, \boldsymbol{v}) \models \varphi_1 \text{ and } (w, i, \boldsymbol{v}) \models \varphi_2 \\
(w, i, \boldsymbol{v}) \models \Diamond_{\prec x}\, \varphi & \text{iff } (w, j, \boldsymbol{v}) \models \varphi \text{ for some } j \prec \boldsymbol{v}(x){+}i.
\end{array}$$

For the sake of brevity, we have omitted the semantics of the standard LTL modalities. As usual, $\varphi_1 \,\mathsf{R}\, \varphi_2 \equiv \neg(\neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2)$, $\Diamond\varphi \equiv \mathrm{true}\,\mathsf{U}\,\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. The language of φ is defined by $\mathcal{L}(\varphi) = \{(w, \boldsymbol{v}) \mid (w, 0, \boldsymbol{v}) \models \varphi\}$. Alur et al. [5] have shown that other modalities such as $\mathsf{U}_{\leqslant x}$, $\Diamond_{>x}$, $\Box_{>x}$, $\mathsf{U}_{>x}$, $\mathsf{R}_{\leqslant x}$ and $\mathsf{R}_{>x}$, can all be encoded in our syntax. For instance, the following equivalences hold:

$$\begin{array}{rcl@{\qquad}rcl}
\Diamond_{>x}\,\varphi & \equiv & \Box_{\leqslant x}\Diamond \bigcirc \varphi, & \Box_{>x}\,\varphi & \equiv & \Diamond_{\leqslant x}\Box \bigcirc \varphi, \\
\varphi\,\mathsf{U}_{\leqslant x}\,\psi & \equiv & (\varphi\,\mathsf{U}\,\psi) \wedge \Diamond_{\leqslant x}\,\psi, & \varphi\,\mathsf{U}_{>x}\,\psi & \equiv & \Box_{\leqslant x}\,(\varphi \wedge \bigcirc(\varphi\,\mathsf{U}\,\psi))
\end{array} \tag{1}$$

In the remainder of this paper, we focus on bounded always and eventualities where all bounds are upper bounds. We abbreviate $\Diamond_{\leqslant x}$ by $\Diamond_x$ and do similar for the other modalities. For valuation $\boldsymbol{v}$ and pLTL-formula φ, let $\boldsymbol{v}(\varphi)$ denote the LTL formula obtained from φ by replacing variable x by its valuation $\boldsymbol{v}(x)$; e.g., $\boldsymbol{v}(\Diamond_x\,\varphi)$ equals $\Diamond_{\boldsymbol{v}(x)}\,\boldsymbol{v}(\varphi)$.

Markov chains. A discrete-time Markov chain M is a quadruple $(S, \mathbf{P}, s_0, L)$ where S is a finite set of states with $m = |S|$, $\mathbf{P} : S \times S \to [0, 1]$ is a stochastic matrix, $s_0 \in S$ an initial state, and $L : S \to 2^{AP}$ a state-labeling function. $\mathbf{P}(u, v)$ denotes the one-step probability of moving from state u to v. A trajectory (or path) of a Markov chain (MC, for short) M is a sequence $\{\, s_i \,\}_{i \geqslant 0}$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geqslant 0$. A trajectory $\pi = s_0 s_1 s_2 \ldots$ induces the trace $\mathit{trace}(\pi) = L(s_0)L(s_1)L(s_2)\ldots$. Let $\mathit{Paths}(M)$ denote the set of paths of MC M. A path π satisfies the pLTL-formula φ under the valuation $\boldsymbol{v}$, denoted $\pi \models \boldsymbol{v}(\varphi)$, whenever $(\mathit{trace}(\pi), 0, \boldsymbol{v}) \models \varphi$ (or equivalently, $(\mathit{trace}(\pi), \boldsymbol{v}) \in \mathcal{L}(\varphi)$). A finite path (or path fragment) satisfies a formula under a valuation if any infinite extension of it also satisfies the formula. Let Pr be the probability measure on sets of paths, defined by a standard cylinder construction [1]. The probability of satisfying φ by M under valuation $\boldsymbol{v}$ is given by $Pr\{\, \pi \in \mathit{Paths}(M) \mid \pi \models \boldsymbol{v}(\varphi) \,\}$, generally abbreviated as $Pr(M \models \boldsymbol{v}(\varphi))$.

Valuation set. The central problem addressed in this paper is to determine the valuation set of a pLTL formula φ. Let M be an MC, $p \in [0, 1]$ a probability bound, and $\prec \in \{=, \leqslant, <, >, \geqslant\}$. Then we are interested in determining:

$$V_{\prec p}(\varphi) \;=\; \{\, \boldsymbol{v} \mid Pr(M \models \boldsymbol{v}(\varphi)) \prec p \,\},$$

i.e., the set of valuations under which the probability of satisfying φ meets the bound $\prec p$. In particular, we will focus on the decidability and complexity of the emptiness problem for $V_{\prec p}(\varphi)$, i.e., the decision problem whether $V_{\prec p}(\varphi) = \varnothing$ or not, on algorithms (if any) determining the set $V_{\prec p}(\varphi)$, and on the size of the minimal representation of $V_{\prec p}(\varphi)$. In the qualitative setting, the bound $\prec p$ is either > 0, or $= 1$.

Proposition 1. *For $\varphi \in$ pLTL, the problem if $V_{>0}(\varphi) = \varnothing$ is undecidable.*

Proof. The proof is based on [5, Th. 4.1], see the archived version for details. ■

It follows that deciding whether $V_{=1}(\varphi) = \varnothing$ is undecidable, as $V_{>0}(\varphi) = \varnothing$ iff $V_{=1}(\neg\varphi) \neq \varnothing$. As a combination of $\Diamond_{\leqslant x}$ and $\Box_{\leqslant x}$ modalities can encode $\mathsf{U}_{=x}$, e.g.,

$$\neg a \wedge \bigcirc(\neg a \,\mathsf{U}_{=x}\, a) \;\equiv\; \bigcirc(\neg a \,\mathsf{U}_{\leqslant x}\, a) \wedge (\neg a \,\mathsf{U}_{>x}\, a),$$

we will restrict ourselves to fragments of pLTL where each formula is in negative normal form and the only parametrized operator is $\Diamond_{\leqslant x}\,\varphi$. We refer to this fragment as pLTL$_\Diamond$:

$$\varphi \;::=\; a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{R}\, \varphi \mid \Box\varphi \mid \Diamond_{\leqslant x}\,\varphi \mid \Diamond_{\leqslant c}\,\varphi \mid \Box_{\leqslant c}\,\varphi. \quad (2)$$

We show it is a sub-logic of pLTL for which the emptiness problem for $V_{>0}(\varphi)$ is decidable. The logic has a favourable *monotonicity* property, i.e.,

Remark 1. For every pLTL$_\Diamond$-formula φ, infinite word w and valuations $\boldsymbol{v}, \boldsymbol{v}'$, $\boldsymbol{v} \leqslant \boldsymbol{v}'$ implies $(w, \boldsymbol{v}) \models \varphi \implies (w, \boldsymbol{v}') \models \varphi$.

Here $(w, \boldsymbol{v}) \models \varphi$ is s shorthand for $(w, 0, \boldsymbol{v}) \models \varphi$. We start off with briefly considering (only) parametric eventualities and then consider the sub-logic pLTL(F,X) restricted to next and $\Diamond_x$. Later on, we also consider parametric Büchi formulas, and finally, pLTL$_\Diamond$.

3 Parametric Reachability

In this section, we consider pLTL-formulas of the form $\Diamond_x\, a$ for proposition a, or equivalently, $\Diamond_x\, T$ for the set of target states $T = \{\, s \in S \mid a \in L(s) \,\}$. We consider bounds of the form $\geqslant p$ with $0 < p < 1$. The valuation set of interest is thus $V_{\geqslant p}(\Diamond_x\, a)$. Let μ_i be the probability of reaching T within i steps; the sequence $\{\mu_i\}$ is ascending. There can be two cases: (a) the sequence reaches a constant value in m steps (m being the size of Markov chain) or (b) the sequence monotonically increases and converges to μ_∞. This makes the emptiness problem for $V_{\geqslant p}(\Diamond_x\, a)$ decidable. In the first case, we check $\mu_m \geqslant p$. In the second case, emptiness is decidable in time polynomial in m, by determining $\mu_\infty = Pr(\Diamond a)$ which can be done by solving a system of linear equations with at most m variables. Then, $V_{\geqslant p}(\Diamond_x\, a) \neq \varnothing$ iff $p < \mu_\infty$.

Assume in the sequel that T is non-empty. Let $\min V_{\geqslant p}(\Diamond_x\, a) = n_0$. The valuation set can thus be represented by n_0 (this gives a minimal representation of the set). Membership queries, i.e., does $n \in V_{\geqslant p}(\Diamond_x\, a)$, then simply boil down to checking whether $n_0 \leqslant n$, which can be done in constant time (modulo the size of n_0). The only catch is that n_0 can be very large if p is close to μ_∞. A simple example elucidates this fact.

Example 1. Consider the MC M with $S = \{\, s_0, t \,\}$, $L(t) = \{\, a \,\}$, $L(s_0) = \varnothing$, $\mathbf{P}(s_0, s_0) = \frac{1}{2} = \mathbf{P}(s_0, t)$ and $\mathbf{P}(t, t) = 1$. Then $Pr(M \models \Diamond_n\, a) = 1 - \left(\frac{1}{2}\right)^n$. It follows that $\min V_{\geqslant p}(\Diamond_x\, a)$ goes to infinity when p approaches one.

The following bound on n_0 can nonetheless be provided. This bound allows for obtaining the minimum value n_0 by a binary search.

Proposition 2. *For MC M, $\min V_{\geqslant p}(\Diamond_x a) \leqslant \log_\gamma(1-(1-\gamma)\frac{p}{b})$, where $0 < \gamma < 1$ and $b > 0$.*

Proof. Collapse all a-states into a single state t and make it absorbing (i.e., replace all outgoing transitions by a self-loop with probability one). Let t be the only *bottom strongly connected component* (BSCC) of M (other BSCCs can be safely ignored). Let $\{1, \ldots, m\}$ be the states of the modified MC M, with the initial state s_0 and the target state t represented by 1 and m, respectively. Let $\mathbf{Q}$ be the $(m-1) \times (m-1)$ transition matrix of the modified MC without the state t. That is, $\mathbf{Q}(i, j) = \mathbf{P}(i, j)$ iff $j \neq m$ where $\mathbf{P}$ is the transition probability matrix of M. We have the following observation:

1. Let the coefficient of ergodicity $\tau(\mathbf{Q})$ of $\mathbf{Q}$ defined as

$$\tau(\mathbf{Q}) \;=\; 1 - \min_{i,j} \left(\sum_k \min\{\mathbf{Q}(i,k), \mathbf{Q}(j,k)\} \right).$$

   As $\mathbf{Q}$ is sub-stochastic and no row of $\mathbf{Q}$ is zero, it follows $0 < \tau(\mathbf{Q}) < 1$.
2. Let vector $\mathbf{r}^T = (r_1, \ldots, r_{m-1})$ with $r_i = \mathbf{P}(i, m)$, $r_{\max}$ be the maximum element in $\mathbf{r}$ and $\mathbf{i}^T$ be $(1, 0, \ldots, 0)$. The probability of reaching the state m from the state 1 in at most $n+1$ steps is the probability of being in some state $i < m$ within n steps and taking the next transition to m:

$$\mu_{n+1} = \sum_{j=0}^{n+1} \mathbf{i}^T \mathbf{Q}^j \mathbf{r} \;\leqslant\; \sum_{j=0}^{n+1} \tau(\mathbf{Q})^j\, r_{\max}.$$

Let $\tau(\mathbf{Q}) = \gamma$ and $r_{\max} = b$. The integer n_0 is the smallest integer such that $\mu_{n_0} \geqslant p$, which implies that $b \cdot \frac{1-\gamma^{n_0}}{1-\gamma} \geqslant p$. This yields $n_0 \leqslant \log_\gamma(1-(1-\gamma)\frac{p}{b})$. ■

As in the non-parametric setting, it follows that (for finite MCs) the valuation sets $V_{>0}(\Diamond_x\, a)$ and $V_{=1}(\Diamond_x\, a)$ can be determined by a graph analysis, i.e. no inspection of the transition probabilities is necessary for qualitative parametric reachability properties.

Proposition 3. *The problem $V_{>0}(\Diamond_x\, a) = \varnothing$ is* NL*-complete.*

Proof. The problem is the same as reachability in directed graphs. ■

Proposition 4. *The sets $V_{>0}(\Diamond_x\, a)$ and $V_{=1}(\Diamond_x\, a)$ can be determined in polynomial time by a graph analysis of MC M.*

Proof. Collapse all the a-states into a target state t and make t absorbing. If $V_{>0}(\Diamond_x\, a)$ is non-empty, it suffices to determine $\min V_{>0}(\Diamond_x\, a)$ which equals the length of a shortest path from s_0 to t. To determine whether $V_{=1}(\Diamond_x\, a)$ is empty or not, we proceed as follows. If a cycle without t is reachable from s_0, then no finite n exists for which the probability of reaching t within n steps equals one. Thus, $V_{=1}(\Diamond_x\, a) = \varnothing$. If this is not the case, then the graph of M is a DAG (apart from the self-loop at t), and $\min V_{=1}(\Diamond_x\, a)$ equals the length of a longest path from s_0 to t. ■

4 The Fragment pLTL(F,X)

This section considers the fragment pLTL(F,X) which is defined by:

$$\varphi \;::=\; a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \Diamond \varphi \mid \Diamond_{\leqslant x}\, \varphi \mid \Diamond_{\leqslant c}\, \varphi$$

Our first result is a necessary and sufficient condition for the emptiness of $V_{>0}(\varphi)$.

Theorem 1. *For $\varphi \in$ pLTL(F,X) and MC M with m states, $V_{>0}(\varphi) \neq \varnothing$ iff $\bar{\boldsymbol{v}} \in V_{>0}(\varphi)$ with $\bar{\boldsymbol{v}}(x) = m{\cdot}|\varphi|$.*

Proof. Let φ be a pLTL(F,X)-formula and assume $V_{>0}(\varphi) \neq \varnothing$. By monotonicity, it suffices to prove that $\boldsymbol{v} \in V_{>0}(\varphi)$ with $\boldsymbol{v} \not\leqslant \bar{\boldsymbol{v}}$ implies $\bar{\boldsymbol{v}} \in V_{>0}(\varphi)$. The proof proceeds in a number of steps. (1) We show that it suffices to consider formulas without disjunction. (2) We show that if path fragment $\pi[0..l] \models \bar{\varphi}$, (where LTL(F,X)-formula $\bar{\varphi}$ is obtained from φ by omitting all parameters from φ) then $\pi[0..l] \models \boldsymbol{v}_l(\varphi)$ with $\boldsymbol{v}_l(x) = l$ for every x. (3) We construct a deterministic Büchi automaton (DBA) $A_{\bar{\varphi}}$ for $\bar{\varphi}$ such that its initial and final state are at most $|\bar{\varphi}|$ transitions apart. (4) We show that reachability of a final state in the product of MC M and DBA $A_{\bar{\varphi}}$ implies the existence of a finite path in M of length at most $m{\cdot}|\varphi|$ satisfying $\bar{\varphi}$.

1. As disjunction distributes over $\wedge, \bigcirc, \Diamond$, and $\Diamond_x$, each formula can be written in disjunctive normal form. Let $\varphi \equiv \varphi_1 \vee \ldots \vee \varphi_k$, where each φ_i is disjunction-free. Evidently, $|\varphi_i| \leqslant |\varphi|$. Assume $\boldsymbol{v} \in V_{>0}(\varphi)$. Then, $\boldsymbol{v} \in V_{>0}(\varphi_i)$ for some $0 < i \leqslant k$. Assuming the theorem holds for φ_i (this will be proven below), $\bar{\boldsymbol{v}}_i \in V_{>0}(\varphi_i)$ with $\bar{\boldsymbol{v}}_i(x) = |\varphi_i|{\cdot}m$. Since $\bar{\boldsymbol{v}} \geqslant \bar{\boldsymbol{v}}_i$, it follows by monotonicity that $\bar{\boldsymbol{v}} \in V_{>0}(\varphi_i)$, and hence, $\bar{\boldsymbol{v}} \in V_{>0}(\varphi)$. It thus suffices in the remainder of the proof to consider disjunction-free formulas.
2. For pLTL(F,X)-formula φ, let $\bar{\varphi}$ be the LTL(F,X)-formula obtained from φ by replacing all occurrences of $\Diamond_x$ by $\Diamond$, e.g., for $\varphi = \Diamond_x(a \wedge \Diamond_y b)$, $\bar{\varphi} = \Diamond(a \wedge \Diamond b)$. We claim that $\pi[0...l] \models \bar{\varphi}$ implies $\pi[0...l] \models \boldsymbol{v}_l(\varphi)$ with $\boldsymbol{v}_l(x) = l$ for all x. This is proven by induction on the structure of φ. The base cases

a and $\neg a$ are obvious. For the induction step, conjunctions, $\bigcirc\varphi$ and $\Diamond\varphi$ are straightforward. It remains to consider $\Diamond_x\varphi$. Assume $\pi[0...l] \models \Diamond\bar{\varphi}$. Thus, for some $i \leqslant l$, $\pi[i...l] \models \bar{\varphi}$. By induction hypothesis, $\pi[i...] \models \boldsymbol{v}_{il}(\varphi)$ with $\boldsymbol{v}_{il}(y) = l-i$ for each variable y in φ. Thus, $\pi[0..l] \models \boldsymbol{v}_l(\Diamond_x\varphi)$ with $\boldsymbol{v}_l(x) = l$ and for all y in φ, $\boldsymbol{v}_l(y) = l$.

3. We provide a DBA $A_{\bar{\varphi}} = \langle Q, \Sigma, \delta, q_0, F\rangle$ with $\Sigma = 2^{AP}$ for each LTL(F,X)-formula $\bar{\varphi}$ using the construction from [15]. We first treat $\bar{\varphi} = a$ and $\bar{\varphi} = \Diamond a$. As every LTL(F,X)-formula can be obtained from $\Diamond(a\wedge\varphi)$, $\varphi_1\wedge\varphi_2$ and $\bigcirc\varphi$, we then treat these inductive cases. (Negations are treated similarly.) For $\bar{\varphi} = a$, $A_a = \langle\{\, q_0, q_1\,\}, \Sigma, \delta, q_0, \{\, q_1\,\}\rangle$ with $\delta(q_0, a) = q_1$ and $\delta(q_1, \text{true}) = q_1$. For $\bar{\varphi} = \Diamond a$, the DBA $A_{\Diamond a} = \langle\{\, q_0, q_1\,\}, \Sigma, \delta, q_0, \{\, q_1\,\}\rangle$, where $\delta(q_0, a) = q_1$, $\delta(q_0, \neg a) = q_0$ and $\delta(q_1, \text{true}) = q_1$. This completes the base cases. For the three inductive cases, the DBA is constructed as follows.
   (a) Let $A_{\bar{\varphi}} = \langle Q, \Sigma, \delta, q_0, F\rangle$. $A_{\Diamond(a\wedge\bar{\varphi})} = \langle Q \cup \{\, q_0'\,\}, \Sigma, \delta', q_0', F\rangle$ where q_0' is fresh, $\delta'(q, \cdot) = \delta(q, \cdot)$ if $q \in Q$, $\delta'(q_0', a) = \delta(q_0, a)$, and $\delta'(q_0', \neg a) = q_0'$.
   (b) For $\bar{\varphi}_1 \wedge \bar{\varphi}_2$, the DBA is a standard synchronous product of the DBA for $\bar{\varphi}_1$ and $\bar{\varphi}_2$.
   (c) Let $A_{\bar{\varphi}} = \langle Q, \Sigma, \delta, q_0, F\rangle$. $A_{\bigcirc\bar{\varphi}} = \langle Q \cup \{\, q_0'\,\}, \Sigma, \delta', q_0', F\rangle$ where q_0' is fresh, $\delta'(q_0', a) = q_0$ for all $a \in \Sigma$ and $\delta'(q, a) = \delta(q, a)$ for every $q \in Q$.

   A few remarks are in order. The resulting DBA have a single final state. In addition, the DBA enjoy the property that the reflexive and transitive closure of the transition relation is a partial order [15]. Formally, $q \preceq q'$ iff $q' \in \delta^*(q, w)$ for some $w \in \Sigma^\omega$. The diameter of $A_{\bar{\varphi}}$ is the length of a longest simple path from the initial to the final state. This implies that the diameter of $A_{\Diamond(a\wedge\bar{\varphi})}$ and $A_{\bigcirc\bar{\varphi}}$ is $n{+}1$ where n is this diameter of $A_{\bar{\varphi}}$, and the diameter of $A_{\bar{\varphi}_1\wedge\bar{\varphi}_2}$ is $n_1 + n_2$ where n_i is the diameter of $A_{\bar{\varphi}_i}$, $i \in \{\, 1, 2\,\}$.
4. Let $\varphi \equiv \varphi_1 \vee \ldots \vee \varphi_k$, where each φ_i is disjunction-free, with DBA $A_{\bar{\varphi}_i}$. Evidently, $V_{>0}(\varphi) \neq \varnothing$ iff $V_{>0}(\varphi_i) \neq \varnothing$ for some disjunct φ_i. Consider the product of MC M and DBA $A_{\bar{\varphi}_i}$, denoted $M \otimes A_{\bar{\varphi}_i}$; see, e.g., [16, Def. 10.50]. By construction, $M \otimes A_{\bar{\varphi}_i}$ is partially ordered and has diameter at most $m{\cdot}|\varphi_i|$. We have that $Pr(M \models \bar{\varphi}_i) > 0$ iff an accepting state in $M \otimes A_{\bar{\varphi}_i}$ is reachable. Thus, there exists a finite path $\pi[0..m{\cdot}|\varphi_i|]$ in M with $\pi[0..m{\cdot}|\varphi_i|] \models \bar{\varphi}$, or, $\pi[0..m{\cdot}|\varphi|] \models \bar{\boldsymbol{v}}(\varphi)$. This concludes the proof.

$M \otimes A_{\bar{\varphi}_i}$ can also be used to show that, if we have a valuation $\boldsymbol{v}$ such that $\boldsymbol{v}(x) > m{\cdot}|\varphi|$ and for all other variables $y \neq x$, $\boldsymbol{v}(x) \leqslant m{\cdot}|\varphi|$ and $\boldsymbol{v} \in V_{>0}(\varphi)$ then $\boldsymbol{v}' \in V_{>0}(\varphi)$, where $\boldsymbol{v}'(x) = m{\cdot}|\varphi|$ and for $y \neq x$, $\boldsymbol{v}'(y) = \boldsymbol{v}(y)$. The argument proceeds as induction on $\bar{\varphi}_i$. ■

The above Theorem 1 leads to the following proposition.

Proposition 5. *For $\varphi \in$ pLTL(F,X), deciding if $V_{>0}(\varphi) = \varnothing$ is NP-complete.*

For almost sure properties, a similar approach as for $V_{>0}(\varphi)$ suffices.

Theorem 2. *For $\varphi \in$ pLTL(F,X) and MC M with m states, $V_{=1}(\varphi) \neq \varnothing$ iff $\bar{\boldsymbol{v}} \in V_{=1}(\bar{\varphi})$ with $\bar{\boldsymbol{v}}(x) = m{\cdot}|\varphi|$.*

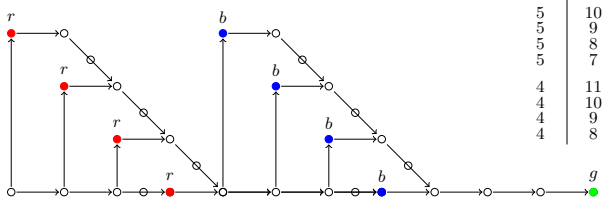
Proof. Consider the direction from left to right. The argument goes along similar lines as the proof of Theorem 1. We build the DBA $A_{\bar{\varphi}}$ for $\bar{\varphi}$ and take the cross product with Markov chain M. There are $m{\cdot}|\varphi|$ state in the cross product. If $Pr(M \models \bar{\boldsymbol{v}}(\varphi)) < 1$ then there is some cycle in the cross product that does not contain the final state. Thus, $V_{=1}(\varphi)$ is empty. ■

Theorem 1 suggests that $\min V_{>0}(\varphi)$ lies in the hyper-cube $H = \{0, \ldots, N\}^d$, where $N = m{\cdot}|\varphi|$. A possible way to find $\min V_{>0}(\varphi)$ is to apply the bisection method in d-dimensions. We recursively choose a middle point of the cube, say $\boldsymbol{v} \in H$ —in the first iteration $\boldsymbol{v}(x) = N/2$— and divide H in 2^d equally sized hypercubes. If $\boldsymbol{v} \in V_{>0}(\varphi)$, then the hypercube whose points exceed $\boldsymbol{v}$ is discarded, else the cube whose points are below $\boldsymbol{v}$ is discarded. The asymptotic time-complexity of this procedure is given by the recurrence relation:

$$T(k) = (2^d - 1) \cdot T(k{\cdot}2^{-d}) + F \tag{3}$$

where k is the number of points in the hypercube and F is the complexity of checking $\boldsymbol{v} \in V_{>0}(\varphi)$ where $|\boldsymbol{v}| \leqslant N$. Section 6 presents an algorithm working in $\mathcal{O}(m{\cdot}N^d{\cdot}2^{|\varphi|})$ for a somewhat more expressive logic. From (3), this yields a complexity of $\mathcal{O}(m{\cdot}N^d{\cdot}2^{|\varphi|}{\cdot}\log N)$. The size of a set of minimal points can be exponential in the number of variables, as shown below.

Proposition 6. $|\min V_{>0}(\varphi)| \leqslant (N{\cdot}d)^{d-1}$.



x_1	x_2	x_3
5	10	14
5	9	15
5	8	16
5	7	17
4	11	15
4	10	16
4	9	17
4	8	18

x_1	x_2	x_3
3	10	16
3	11	17
3	10	18
3	9	19
2	13	17
2	12	18
2	11	19
2	10	20

Fig. 1. MC and $\min V_{>0}(\varphi)$ for pLTL(F,X)-formula $\varphi = \Diamond_{x_1} r \wedge \Diamond_{x_2} b \wedge \Diamond_{x_3} g$

Example 2. There exist MCs for which $|\min V_{>0}(\varphi)|$ grows exponentially in d, the number of parameters in φ, whereas the number m of states in the MC grows linearly in d. For instance, consider the MC M in Fig. 1 and $\varphi = \Diamond_{x_1} r \wedge \Diamond_{x_2} b \wedge \Diamond_{x_3} g$, i.e., d=3. We have $|\min V_{>0}(\varphi)| = 4^2$ as indicated in the table.

We conclude this section by briefly considering the membership query: does $\boldsymbol{v} \in V_{>0}(\varphi)$ for pLTL(F,X)-formula φ with d parameters? Checking membership

of a valuation $\boldsymbol{v} \in V_{>0}(\varphi)$ boils down to deciding whether there exists a $\boldsymbol{v}' \in \min V_{>0}(\varphi)$ such that $\boldsymbol{v} \geqslant \boldsymbol{v}'$. A representation of $\min V_{>0}(\varphi)$ facilitating an efficient membership test can be obtained by putting all elements in this set in lexicographical order. This involves sorting over all d coordinates. A membership query then amounts to a recursive binary search over d dimensions. This yields:

Proposition 7. *For pLTL(F,X)-formula φ, $\boldsymbol{v} \in V_{>0}(\varphi)$? takes $\mathcal{O}(d \cdot \log N \cdot d)$ time, provided a representation of $\min V_{>0}(\varphi)$ is given.*

5 Qualitative Parametric Büchi

In this section, we consider pLTL-formulas of the form $\varphi = \Box\Diamond_x\, a$, for proposition a. We are interested in $V_{>0}(\varphi)$, i.e., does the set of infinite paths visiting a-states that are maximally x apart infinitely often, have a positive measure? Let MC $M = (S, \mathbf{P}, s_0, L)$. A *bottom strongly-connected component* (BSCC) $B \subseteq S$ of M is a set of mutually reachable states with no edge leaving B. For BSCC B, let $n_{a,B} = \max\{\, |\pi| \mid \forall i \leqslant |\pi|, \pi[i] \in B \wedge a \notin L(\pi[i]) \,\}$.

Proposition 8. *Let B be a BSCC and $s \in B$. Then, $\forall n \in \mathbb{N}, n > n_{a,B} \Leftrightarrow Pr(s \models \Box\Diamond_n\ a) = 1$ and $n \leqslant n_{a,B} \Leftrightarrow Pr(s \models \Box\Diamond_n\ a) = 0$.*

Proof. If $n > n_{a,B}$, then each path π from any state $s \in B$ will have at least one a-state in finite path fragment $\pi[i, \ldots, i{+}n]$ for all i. Hence, $Pr(s \models \Box\Diamond_n\ a) = 1$. If $n \leqslant n_{a,B}$, then there exists a finite path fragment ρ of B, such that, for all $i \leqslant n$, $a \notin L(\rho[i])$. Consider an infinite path π starting from any arbitrary $s \in B$. As $s \in B$, π will almost surely infinitely often visit the initial state of ρ. Therefore, by [16, Th.10.25], π will almost surely visit every finite path fragment starting in that state, in particular ρ. Path π thus almost surely refutes $\Box\Diamond_n\ a$, i.e. $Pr(s \models \Box\Diamond_n\ a) = 0$. ■

For any BSCC B and $\Box\Diamond_x\ a$, $n_{a,B} < \infty$ iff every cycle in B has at least one a-state. Hence, $n_{a,B}$ can be obtained by analysing the digraph of B (in $\mathcal{O}(m^2)$, the number of edges). BSCC B is called *accepting* for $\Box\Diamond_x\, a$ if $n_{a,B} < \infty$ and B is reachable from the initial state s_0. Note that this may differ from being an accepting BSCC for $\Box\Diamond a$. Evidently, $V_{>0}(\Box\Diamond_x\, a) \neq \varnothing$ iff $n_{a,B} < \infty$. This result can be extended to *generalized Büchi* formula $\varphi = \Box\Diamond_{x_1}\, a_1 \wedge \ldots \wedge \Box\Diamond_{x_d}\, a_d$, by checking $n_{a_i,B} < \infty$ for each a_i.

As a next problem, we determine $\min V_{>0}(\Box\Diamond_x\, a)$. For the sake of simplicity, let MS M have a single accepting BSCC B. For states s and t in MC M, let $d(s,t)$ be the distance from s to t in the graph of M. (Recall, the distance between state s and t is the length of the shortest path from s to t.) For BSCC B, let $d_{a,B}(s) = \min_{t \in B, a \in L(t)} d(s,t)$, i.e., the minimal distance from s to an a-state in B. Let the proposition a_B hold in state s iff $s \in B$ and $a \in L(s)$. Let $G_a = (V, E)$ be the digraph defined as follows: V contains all a-states of M and the initial state s_0 and $(s, s') \in E$ iff there is path from s to s' in M. Let c be a cost function defined on a finite path $s_0 \ldots s_n$ in graph G_a as: $c(s_0 \ldots s_n) = \max_i d(s_i, s_{i+1})$, ($d$ is defined on the graph of M). Using these auxiliary notions we obtain the following characterization for $\min V_{>0}(\Box\Diamond_x\, a)$:

Theorem 3. $\min V_{>0}(\Box\Diamond_x\, a) = n_0$ *where* $n_0 = \max\left(n_{a,B}, \min_{\pi=s_0\ldots s_n, s_n\models a_B} c(\pi)\right)$ *if* $n_{a,B} < d_{a,B}(s_0)$ *and* $n_0 = n_{a,B}$ *otherwise.*

Proof. We show for $n \geqslant n_0$, $Pr(\Box\Diamond_n\, a) > 0$, and for $n < n_0$, $Pr(\Box\Diamond_n\, a) = 0$. Distinguish:

1. $n_{a,B} \geqslant d_{a,B}(s_0)$. Then, from s_0 an a-state in B can be reached within $n_{a,B}$ steps, i.e., $Pr(s_0 \models \Diamond_{n_{a,B}}\, a_B) > 0$. For this a_B-state, s, say, by Proposition 8 it follows $Pr(s \models \Box\Diamond_{n_{a,B}}\, a) = 1$. Together this yields $Pr(s_0 \models \Box\Diamond_n\, a) > 0$ for each $n \geqslant n_{a,B} = n_0$. For $n < n_0 = n_{a,B}$, it follows by Proposition 8 that $Pr(s \models \Box\Diamond_n\, a) = 0$ for every a_B-state s. Thus, $Pr(s_0 \models \Box\Diamond_n\, a) = 0$.
2. $n_{a,B} < d_{a,B}(s_0)$. As B is accepting, $d_{a,B}(s_0) \neq \infty$. Consider a simple path π from s_0 to an a-state in B. Let $c(\pi)$ be the maximal distance between two consecutive a-states along this path. Then it follows $Pr(s_0 \models \Box\Diamond_k\, a) > 0$ where $k = \max(c(\pi), n_{a,B})$. By taking the minimum c_{min} over all simple paths between s_0 and B, it follows $Pr(s_0 \models \Box\Diamond_n\, a) > 0$ for each $n \geqslant n_0 = \max(n_{a,B}, c_{min})$ with $c_{min} = \min_{\pi\in Paths(s_0,\Diamond a_B)} c(\pi)$. For $n < n_0$, distinguish between $n_0 = n_{a,B}$ and $n_0 = c_{min}$. In the former case, it follows (as in the first case) by Proposition 8 that $Pr(s_0 \models \Box\Diamond_n\, a) = 0$ for all $n \geqslant n_0$. Consider now $n_0 = c_{min} \geqslant n_{a,B}$. Let $n < n_0$. By contra-position. Assume $Pr(s_0 \models \Box\Diamond_n\, a) > 0$. Let $\pi = s_0 \ldots s_{1,a} \ldots s_{2,a} \ldots\ldots s_{k,a}$ be a finite path fragment in M where $s_{i,a} \models a$ and $s_{k,a}$ is the first a-state along π which belongs to B. Then, by definition of the digraph G_a, the sequence $\pi = s_0 s_{1,a} s_{2,a} \ldots s_{k,a}$ is a path in G_a satisfying $c(s_{i,a}, s_{i+1,a}) \leqslant n$ for all $0 \leqslant k < n$. But then $c_{min} \leqslant n$. Contradiction. ■

If MC M has more than one accepting BSCC, say $\{B_1, \ldots, B_k\}$ with $k > 1$, then $n_0 = \min_i n_{0,B_i}$, where n_{0,B_i} for $0 < i \leqslant k$ is obtained as in Theorem 3.

Proposition 9. *The sets* $V_{>0}(\Box\Diamond_x\, a)$ *and* $V_{=1}(\Box\Diamond_x\, a)$ *can be determined in polynomial time by a graph analysis of MC* M*.*

Determining $\min V_{\geqslant p}(\Box\Diamond_x\, a)$ for arbitrary p reduces to reachability of *accepting* BSCCs. In a similar way as for parametric reachability (cf. Section 3), this can be done searching. For generalized Büchi formula $\varphi = \Box\Diamond_{x_i}\, a_i \wedge \ldots \wedge \Box\Diamond_{x_d}\, a_d$ and BSCC B, $n_{a_i B}$ is at most m. Thus, $\min V_{>0}(\varphi) \in \{0, \ldots, m{\cdot}d\}^d$ and can be found by the bisection method, similar to the procedure described in Section 4.

6 The Fragment pLTL$_\Diamond$

This section is concerned with the logical fragment pLTL$_\Diamond$, as defined in (2):

$$\varphi \;::=\; a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{R}\, \varphi \mid \Box\varphi \mid \Diamond_{\leqslant x}\, \varphi.^1$$

We will focus on the emptiness problem: is $V_{>0}(\varphi) = \varnothing$. The decision problem whether $V_{=1}(\varphi)$ is very similar. Similar as for pLTL(F,X), we obtain necessary and sufficient criteria for both cases. The proofs for these criteria depend on an algorithm that checks whether $\boldsymbol{v} \in V_{>0}(\varphi)$. This algorithm is presented first.

[1] The modalities $\Diamond_{\leqslant c}$ and $\Box_{\leqslant c}$ can be removed with only quadratic blow up.

Automata constructions. Let φ be a pLTL$_\Diamond$-formula, and $\boldsymbol{v}$ a variable valuation. W.l.o.g. we assume that each variable occurs once in φ. We will extend the classical automaton-based approach for LTL by constructing a nondeterministic Büchi automaton for φ that is amenable to treat the variables occurring in φ. To that end, inspired by [17], we proceed in a number of steps:

1. Construct an automaton G_φ for φ, independent from the valuation $\boldsymbol{v}$, with two types of acceptance sets, one for treating until and release-modalities (as standard for LTL [18]), and one for treating the parameter constraints.
2. Establish how for a given valuation $\boldsymbol{v}$, a Büchi automaton $B_\varphi(\boldsymbol{v})$ can be obtained from G_φ such that for infinite word w, $(w, \boldsymbol{v}) \in \mathcal{L}(\varphi)$ iff w is an accepting run of $B_\varphi(\boldsymbol{v})$.
3. Exploit the technique advocated by Couvreur *et al.* [3] to verify MC M versus $B_\varphi(\boldsymbol{v})$.

We start with constructing G_φ. Like for the LTL-approach, the first step is to consider consistent sets of sub-formulas of φ. Let $cl(\varphi)$ be the set of all sub-formulas of φ. Set $H \subseteq cl(\varphi)$ is *consistent*, when:

- $a \in H$ iff $\neg a \notin H$,
- $\varphi_1 \wedge \varphi_2 \in H$ iff $\varphi_1 \in H$ and $\varphi_2 \in H$,
- $\varphi_1 \vee \varphi_2 \in H$ iff $\varphi_1 \in H$ or $\varphi_2 \in H$,
- $\varphi_2 \in H$ implies $\varphi_1 \,\mathsf{U}\, \varphi_2 \in H$,
- $\varphi_1, \varphi_2 \in H$ implies $\varphi_1 \,\mathsf{R}\, \varphi_2 \in H$,
- $\varphi_1 \in H$ implies $\Diamond_x \varphi_1 \in H$.

We are now in a position to define G_φ, an automaton with two acceptance sets. For $\varphi \in \text{pLTL}_\Diamond$, let $G_\varphi = (Q, 2^{AP}, Q_0, \delta, Acc_B, Acc_P)$ where

- Q is the set of all consistent sub-sets of $cl(\varphi)$ and $Q_0 = \{\, H \in Q \mid \varphi \in H \,\}$.
- $(H, a, H') \in \delta$, where $a \in 2^{AP}$ whenever:
  - $H \cap AP = \{\, a \,\}$,
  - $\bigcirc \varphi_1 \in H \iff \varphi_1 \in H'$,
  - $\varphi_1 \,\mathsf{U}\, \varphi_2 \in H \iff \varphi_2 \in H$ or $(\varphi_1 \in H$ and $\varphi_1 \,\mathsf{U}\, \varphi_2 \in H')$,
  - $\varphi_1 \,\mathsf{R}\, \varphi_2 \in H \iff \varphi_2 \in H$ and $(\varphi_1 \in H$ or $\varphi_1 \,\mathsf{R}\, \varphi_2 \in H')$,
  - $\Diamond_x \varphi_1 \in H \iff \varphi_1 \in H$ or $\Diamond_x \varphi_1 \in H'$,
- (generalized) Büchi acceptance Acc_B and parametric acceptance Acc_P:
  - $Acc_B = \{\, F_{\varphi'} \mid \varphi' \in cl(\varphi) \wedge (\varphi' = \varphi_1 \,\mathsf{U}\, \varphi_2 \vee \varphi' = \varphi_1 \,\mathsf{R}\, \varphi_2) \,\}$ where
    - $F_{\varphi'} = \{\, H \mid \varphi' \in H \Rightarrow \varphi_2 \in H \,\}$ if $\varphi' = \varphi_1 \,\mathsf{U}\, \varphi_2$, and
    - $F_{\varphi'} = \{\, H \mid \varphi_2 \in H \Rightarrow \varphi' \in H \,\}$ if $\varphi' = \varphi_1 \,\mathsf{R}\, \varphi_2$,
  - $Acc_P = \{\, F_{x_i} \mid \Diamond_{x_i} \varphi_i \in cl(\varphi) \,\}$ with $F_{x_i} = \{\, H \mid \Diamond_{x_i} \varphi_i \in H \Rightarrow \varphi_i \in H \,\}$.

A run $\rho \in Q^\omega$ of G_φ is *accepting* under valuation $\boldsymbol{v}$ if it visits each set in Acc_B infinitely often and each $F_{x_i} \in Acc_P$ in every infix of length $\boldsymbol{v}(x_i)$. $\mathcal{L}(G_\varphi)$ contains all pairs $(w, \boldsymbol{v})$ such that there is an accepting run of w under the valuation $\boldsymbol{v}$. G_φ is *unambiguous* if $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ implies $\mathcal{L}(q') \cap \mathcal{L}(q'') = \varnothing$, where $\mathcal{L}(q)$ is the language starting from the state q.

Proposition 10 ([17]). *For $\varphi \in pLTL_\Diamond$, the automaton G_φ is unambiguous and $\mathcal{L}(G_\varphi) = \mathcal{L}(\varphi)$.*

The automaton G_φ can be constructed in $\mathcal{O}(2^{|\varphi|})$. Apart from the parametric acceptance condition, G_φ behaves as a generalized Büchi automaton (GNBA) with accepting set $Acc_B = \{ F_1, \ldots, F_k \}$. In order to obtain a non-deterministic automaton, we first apply a similar transformation as for GNBA to NBA [16]. We convert G_φ to $U_\varphi = (Q', 2^{AP}, Q'_0, \delta', Acc'_B, Acc'_P)$ where $Q' = Q \times \{ 1, \ldots, k \}$, $Q'_0 = Q_0 \times \{ 1 \}$. If $(q, a, q') \in \delta$, then $((q, i), a, (q', i')) \in \delta'$ with $i{=}i'$ if $q \notin F_i$ else $i' = (i \mod k){+}1$. $Acc_B = F_1 \times \{ 1 \}$ and $Acc'_P = \{ F'_{x_i} \mid F_{x_i} \in Acc_P \}$, where $F'_{x_i} = F_{x_i} \times \{ 1, \ldots, k \}$. Note that the construction preserves unambiguity and the size of U_φ is in $\mathcal{O}(|\varphi| \cdot 2^{|\varphi|})$.

For a given valuation $\boldsymbol{v}$, U_φ can be converted into an NBA $B_\varphi(\boldsymbol{v})$. This is done as follows. Let $U_\varphi = (Q', 2^{AP}, Q'_0, \delta', Acc'_B, Acc'_P)$ and $\boldsymbol{v}$ a valuation of φ with d parameters. Then $B_\varphi(\boldsymbol{v}) = (Q'', 2^{AP}, Q''_0, \delta'', Acc)$ with:

- $Q'' \subseteq Q' \times \{0, \ldots, \boldsymbol{v}(x_1)\} \times \ldots \times \{0, \ldots, \boldsymbol{v}(x_d)\}$,
- $((q, \mathbf{n}), a, (q', \mathbf{n}')) \in \delta''$ if $(q, a, q') \in \delta'$ and for all x_i:
  - if $q' \in F'_{x_i}$ and $\mathbf{n}(x_i) < \boldsymbol{v}(x_i)$ then $\mathbf{n}'(x_i) = 0$,
  - if $q' \notin F'_{x_i}$ and $\mathbf{n}(x_i) < \boldsymbol{v}(x_i)$ then $\mathbf{n}'(x_i) = \mathbf{n}(x_i) + 1$.
- $Q''_0 = Q'_0 \times 0^d$ and $Acc = Acc'_B \times \{0, \ldots, \boldsymbol{v}(x_1)\} \times \ldots \times \{0, \ldots, \boldsymbol{v}(x_d)\}$.

It follows that $B_\varphi(\boldsymbol{v})$ is unambiguous for any valuation $\boldsymbol{v}$. Furthermore, every run of $B_\varphi(\boldsymbol{v})$ is either finite or satisfies the parametric acceptance condition for valuation $\boldsymbol{v}$. Thus we have:

Proposition 11. *An infinite word $w \in \mathcal{L}(B_\varphi(\boldsymbol{v}))$ if and only if $(w, \boldsymbol{v}) \in \mathcal{L}(\varphi)$.*

The size of $B_\varphi(\boldsymbol{v})$ is in $\mathcal{O}(c_{\boldsymbol{v}} \cdot |\varphi| \cdot 2^{|\varphi|})$ where $c_{\boldsymbol{v}} = \prod_{x_i} (\boldsymbol{v}(x_i) + 1)$.

As a next step, we exploit the fact that $B_\varphi(\boldsymbol{v})$ is unambiguous, and apply the technique by Couvreur *et al.* [3] for verifying MC M against $B_\varphi(\boldsymbol{v})$. Let $M \otimes B_\varphi(\boldsymbol{v})$ be the synchronous product of M and $B_\varphi(\boldsymbol{v})$ [16], Π_1 the projection to M and Π_2 the projection to $B_\varphi(\boldsymbol{v})$. Let $\mathcal{L}(s, q) = \{ \pi \in Paths(s) \mid trace(\pi) \in \mathcal{L}(q) \}$ and $Pr(s, q) = Pr(\mathcal{L}(s, q))$. Let $Pr(M \otimes B_\varphi(\boldsymbol{v})) = \sum_{q_0 \in Q_0} Pr(s_0, q_0)$. As $B_\varphi(\boldsymbol{v})$ is unambiguous, we have for any (s, q):

$$Pr(s, q) \;=\; \sum_{(t, q') \in \delta(s, q)} \mathbf{P}(s, t) \cdot Pr(t, q'),$$

where δ is the transition relation of $M \otimes B_\varphi(\boldsymbol{v})$ and $\mathbf{P}(s, t)$ is the one-step transition probability from s to t in MC M. A (maximal) strongly connected component (SCC, for short) $C \subseteq S$ is *complete* if for any $s \in \Pi_1(C)$:

$$Paths(s) = \bigcup_{(s, q) \in C} \mathcal{L}_C(s, q)$$

where $\mathcal{L}_C(s, q)$ *restricts* runs to C (runs only visits states from C). The SCC C is *accepting* if $Acc \cap \Pi_2(C) \neq \varnothing$ (where Acc is the set of accepting states in $B_\varphi(\boldsymbol{v})$).

Proposition 12 ([3]). *Let C be a complete and accepting SCC in $M \otimes B_\varphi(\boldsymbol{v})$. Then for all $s \in \Pi_1(C)$:*

$$Pr\Big(\bigcup_{(s,q)\in C} \mathcal{L}_C(s,q) \Big) = 1.$$

Moreover, since $B_\varphi(\boldsymbol{v})$ is unambiguous, $Pr(M \otimes B_\varphi(\boldsymbol{v})) > 0$ implies there exists a reachable, complete and accepting SCC.

Finding complete and accepting SCC in $M \otimes B_\varphi(\boldsymbol{v})$ is done by standard graph analysis. Altogether, $\boldsymbol{v} \in V_{>0}(\varphi)$ is decided in $\mathcal{O}(m{\cdot}c_{\boldsymbol{v}}{\cdot}|\varphi|{\cdot}2^{|\varphi|})$. The space complexity is polynomial in the size of the input (including the valuation), as $M \otimes B_\varphi(\boldsymbol{v})$ can be stored in $\mathcal{O}(\log m + |\varphi| + \log c_{\boldsymbol{v}})$ bits. In the sequel, we exploit these results to obtain a necessary and sufficient criterion for the emptiness of $V_{>0}(\varphi)$ for φ in pLTL$_\Diamond$.

Theorem 4. *For $\varphi \in pLTL_\Diamond$, $V_{>0}(\varphi) \neq \varnothing$ iff $\bar{\boldsymbol{v}} \in V_{>0}(\varphi)$ s.t. $\bar{\boldsymbol{v}}(x) = m{\cdot}|\varphi|{\cdot}2^{|\varphi|}$.*

Proof. Consider the direction from left to right. The only non-trivial case is when there exists a valuation $\boldsymbol{v} \not\leqslant \bar{\boldsymbol{v}}$ such that $\boldsymbol{v} \in V_{>0}(\varphi)$ implies $\bar{\boldsymbol{v}} \in V_{>0}(\varphi)$. In the model checking algorithm described above, we first construct G_φ, and then U_φ with a single Büchi accepting set Acc'_B and d parametric accepting sets F'_{x_i}, one for each variable x_i in φ. For the sake of clarity, assume $d = 1$, i.e., we consider valuation v. The explanation extends to the general case in a straightforward manner. For valuation v, consider $M \otimes B_\varphi(v)$. We show that, for $r < v$, $Pr(M \otimes B_\varphi(v)) > 0$ implies $Pr(M \otimes B_\varphi(r)) > 0$, where $r = m{\cdot}|U_\varphi|$, which is in $\mathcal{O}(m{\cdot}|\varphi|{\cdot}2^{|\varphi|})$.

Note that every cycle in $M \otimes B_\varphi(r)$ contains a state (s,q,i) with $i = 0$. Moreover, the graph of $M \otimes B_\varphi(r)$ is a sub-graph of $M \otimes B_\varphi(v)$. We now prove that, if a (maximal) SCC C of $M \otimes B_\varphi(r)$ is not complete (or accepting) then any SCC C' of $M \otimes B_\varphi(v)$ containing C is also not complete (or accepting, respectively).

(a) Suppose C is not complete. Then there exists a finite path $\sigma = s\, s_1 \dots s_k$ of M, such that from any q, with $(s,q,0) \in C$, the run $\rho = (s,q,0)(s_1,q_1,1)\dots(s_j, q_j, j)$ leads to a deadlock state. This can have two causes: either (s_j, q_j, j) has no successor for any j. Then, C' is not complete. Or, the path ρ terminates in (s_j, q_j, j) where $j = r$. This means, for all $(s', q', j{+}1) \in \delta(s_j, q_j, j)$ in C', $q' \notin F_x$. As the length of ρ exceeds r, there are states in the run whose first and second component appear multiple times. Thus, we can find another path σ' (possibly longer than σ) for C' which goes through states where the first and the second component of some of its states are repeated sufficiently many times to have a run $(s,q,0)(s_1,q_1,1)\dots(s_j,q_j,v)$ which is a deadlock state. Thus, C' is not complete.

(b) Suppose C' is accepting. Then there exists (s', q', i') with $q' \in Acc$. Since C' is an SCC and $C \subseteq C'$, there is a path from $(s,q,0) \in C$ to (s', q', i'). If the length of the path is less than r, then we are done. If $i' > r$, then some (s'', q'') pair in the path must be repeated. Thus, we can find another path of length less

than r to a state (s', q', i), where $i \leqslant r$. Therefore, C is accepting. The rest of the proof follows from Proposition 12. ■

For almost sure properties, a similar approach as for $V_{>0}(\varphi)$ suffices.

Theorem 5. *For $\varphi \in pLTL_{\Diamond}$, $V_{=1}(\varphi) \neq \varnothing$ iff $\bar{\boldsymbol{v}} \in V_{=1}(\bar{\varphi})$ with $\bar{\boldsymbol{v}}(x) = m{\cdot}|\varphi|{\cdot}2^{|\varphi|}$.*

Let $N_{\varphi M} = m{\cdot}|\varphi|{\cdot}2^{|\varphi|}$. Note that $c_{\bar{\boldsymbol{v}}}$ equals $(N_{\varphi M})^d$. Thus, we have:

Proposition 13. *For $\varphi \in pLTL_{\Diamond}$, deciding if $V_{>0}(\varphi) = \varnothing$ is PSPACE-complete.*

Proof. Theorem 4 gives an algorithm in PSPACE, as $M \otimes B_{\varphi}(\bar{\boldsymbol{v}})$ can be stored in $O(\log m + |\varphi| + d \log N_{\varphi M})$ bits. PSPACE hardness follows trivially, as for LTL formula φ and MC M, deciding $Pr(M \models \varphi) > 0$ (which is known to be a PSPACE complete problem) is the same as checking the emptiness of $V_{>0}(\varphi)$. ■

Just as for pLTL(F,X), we can use the bisection method to find $\min V_{>0}(\varphi)$. The search procedure invokes the model checking algorithm multiple times. We can reuse the space each time we check $Pr(M \models \boldsymbol{v}(\varphi)) > 0$. Hence, $\min V_{>0}(\varphi)$ can be found in polynomial space. The time complexity of finding $\min V_{>0}(\varphi)$ is $\mathcal{O}(m{\cdot}(N_{\varphi M})^d{\cdot}2^{|\varphi|}{\cdot}\log N_{\varphi M})$. Membership can also be similarly solved.

Proposition 14. *For $pLTL_{\Diamond}$-formula φ, $\boldsymbol{v} \in V_{>0}(\varphi)$? takes $\mathcal{O}(d{\cdot}\log \frac{N_{\varphi M}}{d})$ time, provided a representation of $V_{>0}(\varphi)$ is given.*

7 Concluding Remarks

This paper considered the verification of finite MCs against parametric LTL. We obtained several results on the emptiness problem for qualitative verification problems, including necessary and sufficient conditions as well as some complexity results. Future work consists of devising more efficient algorithms for the quantitative verification problems, and lifting the results to extended temporal logics [19] and stochastic games, possibly exploiting [17].

Acknowledgement. This work was partially supported by the EU FP7 projects MoVeS and Sensation, the EU Marie Curie project MEALS and the Excellence initiative of the German federal government.

References

1. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338. IEEE Computer Society (1985)
2. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
3. Couvreur, J.M., Saheb, N., Sutre, G.: An optimal automata approach to LTL model checking of probabilistic systems. In: Vardi, M.Y., Voronkov, A. (eds.) LPAR 2003. LNCS, vol. 2850, pp. 361–375. Springer, Heidelberg (2003)

4. Chatterjee, K., Gaiser, A., Kretínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013)
5. Alur, R., Etessami, K., La Torre, S., Peled, D.: Parametric temporal logic for "model measuring". ACM Trans. Comput. Log. 2(3), 388–407 (2001)
6. Han, T., Katoen, J.P., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: IEEE Real-Time Systems Symposium (RTSS), pp. 173–182. IEEE Computer Society (2008)
7. Hahn, E.M., Han, T., Zhang, L.: Synthesis for PCTL in parametric markov decision processes. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 146–161. Springer, Heidelberg (2011)
8. Puggelli, A., Li, W., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Polynomial-time verification of PCTL properties of mDPs with convex uncertainties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 527–542. Springer, Heidelberg (2013)
9. Benedikt, M., Lenhardt, R., Worrell, J.: LTL model checking of interval markov chains. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 32–46. Springer, Heidelberg (2013)
10. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. STTT 13(1), 3–19 (2011)
11. Daws, C.: Symbolic and parametric model checking of discrete-time markov chains. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 280–294. Springer, Heidelberg (2005)
12. Katoen, J.P., McIver, A., Meinicke, L., Morgan, C.C.: Linear-invariant generation for probabilistic programs: In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 390–406. Springer, Heidelberg (2010)
13. Benedikt, M., Lenhardt, R., Worrell, J.: Two variable vs. linear temporal logic in model checking and games. Logical Methods in Computer Science 9(2) (2013)
14. Ummels, M., Baier, C.: Computing quantiles in Markov reward models. In: Pfenning, F. (ed.) FOSSACS 2013. LNCS, vol. 7794, pp. 353–368. Springer, Heidelberg (2013)
15. Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Log. 5(1), 1–25 (2004)
16. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
17. Zimmermann, M.: Optimal bounds in parametric LTL games. Theor. Comput. Sci. 493, 30–45 (2013)
18. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
19. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115, 1–37 (1994)

Specifying and Verifying Properties of Space*

Vincenzo Ciancia[1], Diego Latella[1], Michele Loreti[2], and Mieke Massink[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', CNR, Italy
[2] Università di Firenze, Italy

Abstract. The interplay between process behaviour and spatial aspects of computation has become more and more relevant in Computer Science, especially in the field of *collective adaptive systems*, but also, more generally, when dealing with systems distributed in physical space. Traditional verification techniques are well suited to analyse the temporal evolution of programs; properties of space are typically not explicitly taken into account. We propose a methodology to verify properties depending upon physical space. We define an appropriate logic, stemming from the tradition of topological interpretations of modal logics, dating back to earlier logicians such as Tarski, where modalities describe neighbourhood. We lift the topological definitions to a more general setting, also encompassing discrete, graph-based structures. We further extend the framework with a spatial *until* operator, and define an efficient model checking procedure, implemented in a proof-of-concept tool.

1 Introduction

Much attention has been devoted in Computer Science to formal verification of process behaviour. Several techniques, such as *run-time monitoring* and *model-checking*, are based on a formal understanding of system requirements through *modal* logics. Such logics typically have a *temporal* flavour, describing the flow of events along time, and are interpreted in various kinds of transition structures.

Recently, aspects of computation related to the distribution of systems in physical space have become more relevant. An example is provided by so called *collective adaptive systems*[1], typically composed of a large number of interacting objects. Their global behaviour critically depends on interactions which are often local in nature. Locality immediately poses issues of spatial distribution of objects. Abstraction from spatial distribution may sometimes provide insights in the system behaviour, but this is not always the case. For example, consider a bike (or car) sharing system having several parking stations, and featuring twice as many parking slots as there are vehicles in the system. Ignoring the spatial dimension, on average, the probability to find completely full or empty parking stations at an arbitrary station is very low; however, this kind of analysis may

* Research partially funded by EU ASCENS (nr. 257414), EU QUANTICOL (nr. 600708), IT MIUR CINA and PAR FAS 2007-2013 Regione Toscana TRACE-IT.

[1] See e.g. the web site of the QUANTICOL project: `http://www.quanticol.eu`

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 222–235, 2014.

be misleading, as in practice some stations are much more popular than others, often depending on nearby points of interest. This leads to quite different probabilities to find stations completely full or empty, depending on the spatial properties of the examined location. In such situations, it is important to be able to predicate over spatial aspects, and eventually find methods to certify that a given formal model of space satisfies specific requirements in this respect. In Logics, there is quite an amount of literature focused on so called *spatial* logics, that is, a spatial interpretation of modal logics. Dating back to early logicians such as Tarski, modalities may be interpreted using the concept of *neighbourhood* in a topological space. The field of spatial logics is well developed in terms of descriptive languages and computability/complexity aspects. However, the frontier of current research does not yet address verification problems, and in particular, discrete models are still a relatively unexplored field.

In this paper, we extend the topological semantics of modal logics to *closure spaces*. As we shall discuss in the paper, this choice is motivated by the need to use non-idempotent *closure operators*. A closure space (also called *Čech closure space* or *preclosure space* in the literature), is a generalisation of a standard topological space, where idempotence of closure is not required. By this, graphs and topological spaces are treated uniformly, letting the topological and graph-theoretical notions of neighbourhood coincide. We also provide a spatial interpretation of the *until* operator, which is fundamental in the classical temporal setting, arriving at the definition of a logic which is able to describe unbounded areas of space. Intuitively, the spatial until operator describes a situation in which it is not possible to "escape" an area of points satisfying a certain property, unless by passing through at least one point that satisfies another given formula. To formalise this intuition, we provide a characterising theorem that relates infinite paths in a closure space and until formulas. We introduce a model-checking procedure that is *linear* in the size of the considered space. A prototype implementation of a spatial model-checker has been made available; the tool is able to interpret spatial logics on digital images, providing graphical understanding of the meaning of formulas, and an immediate form of counterexample visualisation.[2]

Related work. We use the terminology *spatial logics* in the "topological" sense; the reader should be warned that in Computer Science literature, spatial logics typically describe situations in which modal operators are interpreted syntactically, against the structure of agents in a process calculus (see [8,6] for some classical examples). The object of discussion in this research line are operators that quantify e.g., over the parallel sub-components of a system, or the hidden resources of an agent. Furthermore, logics for graphs have been studied in the context of databases and process calculi (see [7,15], and references), even though the relationship with physical space is often not made explicit, if considered at all. The influence of space on agents interaction is also considered in the literature on process calculi using *named locations* [11].

[2] Due to lack of space all the proofs are omitted and can be found in [10].

Variants of spatial logics have also been proposed for the symbolic representation of the contents of images, and, combined with temporal logics, for sequences of images [12]. The approach is based on a discretisation of the space of the images in rectangular regions and the orthogonal projection of objects and regions onto Cartesian coordinate axes such that their possible intersections can be analysed from different perspectives. It involves two spatial until operators defined on such projections considering spatial shifts of regions along the positive, respectively negative, direction of the coordinate axes and it is very different from the topological spatial logic approach.

A successful attempt to bring topology and digital imaging together is represented by the field of *digital topology* [22,25]. In spite of its name, this area studies digital images using models inspired by topological spaces, but neither generalising nor specialising these structures. Rather recently, closure spaces have been proposed as an alternative foundation of digital imaging by various authors, especially Smyth and Webster [23] and Galton [17]; we continue that research line, enhancing it with a logical perspective. Kovalevsky [19] studied alternative axioms for topological spaces in order to recover well-behaved notions of neighbourhood. In the terminology of closure spaces, the outcome is that one may impose closure operators on top of a topology, that do not coincide with topological closure. The idea of interpreting the until operator in a topological space is briefly discussed in the work by Aiello and van Benthem [1,24]. We start from their definition, discuss its limitations, and provide a more fine-grained operator, which is interpreted in closure spaces, and has therefore also an interpretation in topological spaces. In the specific setting of complex and collective adaptive systems, techniques for efficient approximation have been developed in the form of mean-field / fluid-flow analysis (see [5] for a tutorial introduction). Recently (see e.g., [9]), the importance of spatial aspects has been recognised and studied in this context. In this work, we aim at paving the way for the inclusion of spatial logics, and their verification procedures, in the framework of mean-field and fluid-flow analysis of collective adaptive systems.

2 Closure Spaces

In this work, we use *closure spaces* to define basic concepts of *space*. Below, we recall several definitions, most of which are explained in [17].

Definition 1. *A* closure space *is a pair* $(X, \mathcal{C})$ *where* X *is a set, and the* closure operator $\mathcal{C} : 2^X \to 2^X$ *assigns to each subset of* X *its* closure*, obeying to the following laws, for all* $A, B \subseteq X$*:*

1. $\mathcal{C}(\emptyset) = \emptyset$*;*
2. $A \subseteq \mathcal{C}(A)$*;*
3. $\mathcal{C}(A \cup B) = \mathcal{C}(A) \cup \mathcal{C}(B)$*.*

As a matter of notation, in the following, for $(X, \mathcal{C})$ a closure space, and $A \subseteq X$, we let $\overline{A} = X \setminus A$ be the complement of A in X.

Definition 2. *Let $(X, \mathcal{C})$ be a closure space, for each $A \subseteq X$:*

1. *the* interior *$\mathcal{I}(A)$ of A is the set $\overline{\mathcal{C}(\overline{A})}$;*
2. *A is a* neighbourhood *of $x \in X$ if and only if $x \in \mathcal{I}(A)$;*
3. *A is* closed *if $A = \mathcal{C}(A)$ while it is* open *if $A = \mathcal{I}(A)$.*

Lemma 1. *Let $(X, \mathcal{C})$ be a closure space, the following properties hold:*

1. *$A \subseteq X$ is open if and only if $\overline{A}$ is closed;*
2. *closure and interior are monotone operators over the inclusion order, that is: $A \subseteq B \implies \mathcal{C}(A) \subseteq \mathcal{C}(B)$ and $\mathcal{I}(A) \subseteq \mathcal{I}(B)$*
3. *Finite intersections and arbitrary unions of open sets are open.*

Closure spaces are a generalisation of *topological spaces*. The axioms defining a closure space are also part of the definition of a *Kuratowski closure space*, which is one of the possible alternative definitions of a topological space. More precisely, a closure space is Kuratowski, therefore a topological space, whenever closure is *idempotent*, that is, $\mathcal{C}(\mathcal{C}(A)) = \mathcal{C}(A)$. We omit the details for space reasons (see e.g., [17] for more information).

Next, we introduce the topological notion of *boundary*, which also applies to closure spaces, and two of its variants, namely the *interior* and *closure* boundary (the latter is sometimes called *frontier*).

Definition 3. *In a closure space $(X, \mathcal{C})$, the* boundary *of $A \subseteq X$ is defined as $\mathcal{B}(A) = \mathcal{C}(A) \setminus \mathcal{I}(A)$. The* interior boundary *is $\mathcal{B}^-(A) = A \setminus \mathcal{I}(A)$, and the* closure boundary *is $\mathcal{B}^+(A) = \mathcal{C}(A) \setminus A$.*

Proposition 1. *The following equations hold in a closure space:*

$$\mathcal{B}(A) = \mathcal{B}^+(A) \cup \mathcal{B}^-(A) \tag{1}$$

$$\mathcal{B}^+(A) \cap \mathcal{B}^-(A) = \emptyset \tag{2}$$

$$\mathcal{B}(A) = \mathcal{B}(\overline{A}) \tag{3}$$

$$\mathcal{B}^+(A) = \mathcal{B}^-(\overline{A}) \tag{4}$$

$$\mathcal{B}^+(A) = \mathcal{B}(A) \cap \overline{A} \tag{5}$$

$$\mathcal{B}^-(A) = \mathcal{B}(A) \cap A \tag{6}$$

$$\mathcal{B}(A) = \mathcal{C}(A) \cap \mathcal{C}(\overline{A}) \tag{7}$$

3 Quasi-Discrete Closure Spaces

In this section we see how a closure space may be derived starting from a *binary relation*, that is, a *graph*. The following comes from [17].

Definition 4. *Consider a set X and a relation $R \subseteq X \times X$. A closure operator is obtained from R as $\mathcal{C}_R(A) = A \cup \{x \in X \mid \exists a \in A.(a, x) \in R\}$.*

Remark 1. One could also change Definition 4 so that $\mathcal{C}_R(A) = A \cup \{x \in X \mid \exists a \in A.(x,a) \in R\}$, which actually is the definition of [17]. This does not affect the theory presented in the paper. Indeed, one obtains the same results by replacing R with R^{-1} in statements of theorems that explicitly use R, and are not invariant under such change. By our choice, closure represents the "least possible enlargement" of a set of nodes.

Proposition 2. *The pair $(X, \mathcal{C}_R)$ is a closure space.*

Closure operators obtained by Definition 4 are not necessarily idempotent. Lemma 11 in [17] provides a necessary and sufficient condition, that we rephrase below. We let $R^{=}$ denote the reflexive closure of R (that is, the least relation that includes R and is reflexive).

Lemma 2. *$\mathcal{C}_R$ is idempotent if and only if $R^{=}$ is transitive.*

Note that, when R is transitive, so is $R^{=}$, thus $\mathcal{C}_R$ is idempotent. The vice-versa is not true, e.g., when $(x,y) \in R$, $(y,x) \in R$, but $(x,x) \notin R$.

Remark 2. In topology, open sets play a fundamental role. However, the situation is different in closure spaces derived from a relation R. For example, in the case of a closure space derived from a connected symmetric relation, the only open sets are the whole space, and the empty set.

Proposition 3. *Given $R \subseteq X \times X$, in the space $(X, \mathcal{C}_R)$, we have:*

$$\mathcal{I}(A) = \{x \in A \mid \neg\exists a \in \overline{A}.(a,x) \in R\} \tag{8}$$

$$\mathcal{B}^{-}(A) = \{x \in A \mid \exists a \in \overline{A}.(a,x) \in R\} \tag{9}$$

$$\mathcal{B}^{+}(A) = \{x \in \overline{A} \mid \exists a \in A.(a,x) \in R\} \tag{10}$$

We note in passing that [16] provides an alternative definition of boundaries for closure spaces obtained from Definition 4, and proves that it coincides with the topological definition (our Definition 3). Closure spaces derived from a relation can be characterised as *quasi-discrete* spaces (see also Lemma 9 of [17] and the subsequent statements).

Definition 5. *A closure space is* quasi-discrete *if and only if one of the following equivalent conditions holds: i) each $x \in X$ has a* minimal neighbourhood[3] *N_x; ii) for each $A \subseteq X$, $\mathcal{C}(A) = \bigcup_{a \in A} \mathcal{C}(\{a\})$.*

Theorem 1. *(Theorem 1 in [17]) A closure space $(X, \mathcal{C})$ is quasi-discrete if and only if there is a relation $R \subseteq X \times X$ such that $\mathcal{C} = \mathcal{C}_R$.*

[3] A *minimal neighbourhood* of x is a set that is a neighbourhood of x (Definition 2 (2)) and is included in all other neighbourhoods of x.

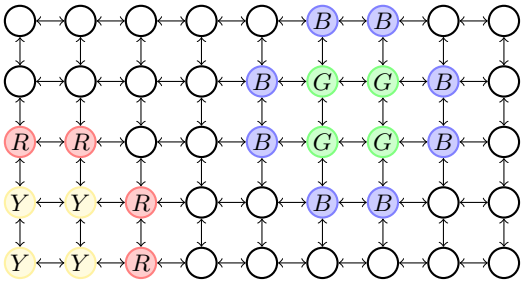


Fig. 1. A graph inducing a *quasi-discrete* closure space

Example 1. Every graph induces a *quasi-discrete* closure space. For instance, we can consider the (undirected) graph depicted in Figure 1. Let R be the (symmetric) binary relation induced by the graph edges, and let Y and G denote the set of *yellow* and *green* nodes, respectively. The closure $\mathcal{C}_R(Y)$ consists of all *yellow* and *red* nodes, while the closure $\mathcal{C}_R(G)$ contains all *green* and *blue* nodes. The interior $\mathcal{I}(Y)$ of Y contains a single node, i.e. the one located at the bottom-left in Figure 1. On the contrary, the *interior* $\mathcal{I}(G)$ of G is empty. Indeed, we have that $\mathcal{B}(G) = \mathcal{C}(G)$, while $\mathcal{B}^-(G) = G$ and $\mathcal{B}^+(G)$ consists of the *blue* nodes.

4 A Spatial Logic for Closure Spaces

In this section we present a spatial logic that can be used to express properties of closure spaces. The logic features two *spatial operators*: a "one step" modality, turning closure into a logical operator, and a binary *until* operator, which is interpreted spatially. Before introducing the complete framework, we first discuss the design of an *until operator* $\phi \mathcal{U} \psi$.

The spatial logical operator $\mathcal{U}$ is interpreted on points of a closure space. The basic idea is that point x satisfies $\phi \mathcal{U} \psi$ whenever it is included in an area A satisfying ϕ, and there is "no way out" from A unless passing through an area B that satisfies ψ. For instance, if we consider the model of Figure 1, *yellow* nodes satisfy *yellow* $\mathcal{U}$ *red* while *green* nodes satisfy *green* $\mathcal{U}$ *blue*. To turn this intuition into a mathematical definition, one should clarify the meaning of the words *area*, *included*, *passing*, in the context of closure spaces.

In order to formally define our logic, and the *until* operator in particular, we first need to introduce the notion of *model*, providing a context of evaluation for the satisfaction relation, as in $\mathcal{M}, x \models \phi \mathcal{U} \psi$. From now on, fix a (finite or countable) set P of *proposition letters.*

Definition 6. *A* closure model *is a pair* $\mathcal{M} = ((X, \mathcal{C}), \mathcal{V})$ *consisting of a closure space* $(X, \mathcal{C})$ *and a valuation* $\mathcal{V} : P \to 2^X$*, assigning to each proposition letter the set of points where the proposition holds.*

When $(X, \mathcal{C})$ is a topological space (that is, $\mathcal{C}$ is idempotent), we call $\mathcal{M}$ a *topological model*, in line with [24], and [1], where the *topological until* operator is presented. We recall it below.

Definition 7. *The* topological until *operator $\mathcal{U}_T$ is interpreted in a topological model $\mathcal{M}$ as $\mathcal{M}, x \models \phi \mathcal{U}_T \psi \iff \exists A$ open $.x \in A \wedge \forall y \in A. \mathcal{M}, y \models \phi \wedge \forall z \in \mathcal{B}(A). \mathcal{M}, z \models \psi$.*

The intuition behind this definition is that one seeks for an area A (which, topologically speaking, could sensibly be an open set) where ϕ holds, and that is completely surrounded by points where ψ holds. Unfortunately, Definition 7 cannot be translated directly to closure spaces, even if all the used topological notions have a counterpart in the more general setting of closure spaces. Open sets in closure spaces are often too coarse (see Remark 2). For this reason, we can modify Definition 7 by not requiring A to be an *open* set. However, the usage of $\mathcal{B}$ in Definition 7 is not satisfactory either. By Proposition 1 we have $\mathcal{B}(A) = \mathcal{B}^+(A) \cup \mathcal{B}^-(A)$, where $\mathcal{B}^-(A)$ is included in A while $\mathcal{B}^+(A)$ is in $\overline{A}$. For instance, when $\mathcal{B}$ is used in Definition 7, we have that the *green* nodes in Figure 1 do not satisfy *green* $\mathcal{U}_T$ *blue*. Indeed, as we remarked in Example 1, the *boundary* of the set G of green nodes coincide with the closure of G that contains both *green* and *blue* nodes.

A more satisfactory definition can be obtained by letting $\mathcal{B}^+$ play the same role as $\mathcal{B}$ in Definition 7 and not requiring A to be an open set. We shall in fact require that ϕ is satisfied by *all* the points of A, and that in $\mathcal{B}^+(A)$, ψ holds. This allows us to ensure that there are no "gaps" between the region satisfying ϕ and that satisfying ψ.

4.1 Syntax and Semantics of *SLCS*

We can now define *SLCS*: a *Spatial Logic for Closure Spaces*. The logic features boolean operators, a "one step" modality, turning closure into a logical operator, and a spatially interpreted *until* operator. More precisely, as we shall see, the *SLCS* formula $\phi \mathcal{U} \psi$ requires ϕ to hold at least on one point. The operator is similar to a *weak until* in temporal logics terminology, as there may be no point satisfying ψ, if ϕ holds everywhere.

Definition 8. *The syntax of SLCS is defined by the following grammar, where p ranges over P:*

$$\Phi ::= p \mid \top \mid \neg \Phi \mid \Phi \wedge \Phi \mid \Diamond \Phi \mid \Phi \mathcal{U} \Phi$$

Here, $\top$ denotes *true*, $\neg$ is negation, $\wedge$ is conjunction, $\Diamond$ is the *closure* operator, and $\mathcal{U}$ is the *until* operator. Closure (and interior, see Figure 2) operators come from the tradition of topological spatial logics [24].

$$\begin{array}{lll} \bot \triangleq \neg\top & \phi \vee \psi \triangleq \neg(\neg\phi \wedge \neg\psi) & \Box\phi \triangleq \neg(\Diamond\neg\phi) \\ \partial\phi \triangleq (\Diamond\phi) \wedge (\neg\Box\phi) & \partial^{-}\phi \triangleq \phi \wedge (\neg\Box\phi) & \partial^{+}\phi \triangleq (\Diamond\phi) \wedge (\neg\phi) \\ \phi\,\mathcal{R}\psi \triangleq \neg((\neg\psi)\,\mathcal{U}(\neg\phi)) & \mathcal{G}\phi \triangleq \phi\,\mathcal{U}\bot & \mathcal{F}\phi \triangleq \neg\mathcal{G}(\neg\phi) \end{array}$$

Fig. 2. *SLCS* derivable operators

Definition 9. *Satisfaction $\mathcal{M}, x \models \phi$ of formula ϕ at point x in model $\mathcal{M} = ((X, \mathcal{C}), \mathcal{V})$ is defined, by induction on terms, as follows:*

$$\begin{array}{lll} \mathcal{M}, x \models p & \iff & x \in \mathcal{V}(p) \\ \mathcal{M}, x \models \top & \iff & \mathit{true} \\ \mathcal{M}, x \models \neg\phi & \iff & \mathcal{M}, x \not\models \phi \\ \mathcal{M}, x \models \phi \wedge \psi & \iff & \mathcal{M}, x \models \phi \text{ and } \mathcal{M}, x \models \psi \\ \mathcal{M}, x \models \Diamond\phi & \iff & x \in \mathcal{C}(\{y \in X | \mathcal{M}, y \models \phi\}) \\ \mathcal{M}, x \models \phi\,\mathcal{U}\psi & \iff & \exists A \subseteq X. x \in A \wedge \forall y \in A. \mathcal{M}, y \models \phi \wedge \\ & & \quad \wedge \forall z \in \mathcal{B}^{+}(A). \mathcal{M}, z \models \psi \end{array}$$

In Figure 2, we present some derived operators. Besides standard logical connectives, the logic can express the *interior* ($\Box\phi$), the *boundary* ($\partial\phi$), the *interior boundary* ($\partial^{-}\phi$) and the *closure boundary* ($\partial^{+}\phi$) of the set of points satisfying formula ϕ. Moreover, by appropriately using the *until* operator, operators concerning *reachability* ($\phi\,\mathcal{R}\psi$), *global satisfaction* ($\mathcal{G}\phi$) and *possible satisfaction* ($\mathcal{F}\phi$) can be derived.

To clarify the expressive power of $\mathcal{U}$ and operators derived from it we provide Theorem 2 and Theorem 3, giving a formal meaning to the idea of "way out" of ϕ, and providing an interpretation of $\mathcal{U}$ in terms of *paths*.

Definition 10. *A* closure-continuous function *$f : (X_1, \mathcal{C}_1) \to (X_2, \mathcal{C}_2)$ is a function $f : X_1 \to X_2$ such that, for all $A \subseteq X_1$, $f(\mathcal{C}_1(A)) \subseteq \mathcal{C}_2(f(A))$.*

Definition 11. *Consider a closure space $(X, \mathcal{C})$, and the quasi-discrete space $(\mathbb{N}, \mathcal{C}_{Succ})$, where $(n, m) \in Succ \iff m = n + 1$. A (countable)* path *in $(X, \mathcal{C})$ is a closure-continuous function $p : (\mathbb{N}, \mathcal{C}_{Succ}) \to (X, \mathcal{C})$. We call p a path* from *x, and write $p : x \rightsquigarrow \infty$, when $p(0) = x$. We write $y \in p$ whenever there is $l \in \mathbb{N}$ such that $p(l) = y$. We write $p : x \overset{A}{\underset{y}{\rightsquigarrow}} \infty$ when p is a path from x, and there is l with $p(l) = y$ and for all $l' \leq l. p(l') \in A$.*

Theorem 2. *If $\mathcal{M}, x \models \phi\,\mathcal{U}\psi$, then for each $p : x \rightsquigarrow \infty$ and l, if $\mathcal{M}, p(l) \models \neg\phi$, there is $k \in \{1, \ldots, l\}$ such that $\mathcal{M}, p(k) \models \psi$.*

Theorem 2 can be strengthened to a necessary and sufficient condition in the case of models based on quasi-discrete spaces. First, we establish that paths in a quasi-discrete space are also paths in its underlying graph.

Lemma 3. *Given path p in a quasi-discrete space $(X, \mathcal{C}_R)$, for all $i \in \mathbb{N}$ with $p(i) \neq p(i+1)$, we have $(p(i), p(i+1)) \in R$, i.e., the image of p is a (graph theoretical, infinite) path in the graph of R. Conversely, each path in the graph of R uniquely determines a path in the sense of Definition 11.*

Function $\texttt{Sat}(\mathcal{M}, \phi)$
  Input: Quasi-discrete closure model $\mathcal{M} = ((X, \mathcal{C}), \mathcal{V})$, $SLCS$ formula ϕ
  Output: Set of points $\{x \in X \mid \mathcal{M}, x \models \phi\}$
  Match ϕ
    case $\top$: **return** X
    case p : **return** $\mathcal{V}(p)$
    case $\neg\psi$:
      let $P = \texttt{Sat}(\mathcal{M}, \psi)$ **in**
      return $X \setminus P$
    case $\psi \wedge \xi$:
      let $P = \texttt{Sat}(\mathcal{M}, \psi)$ **in**
      let $Q = \texttt{Sat}(\mathcal{M}, \xi)$ **in**
      return $P \cap Q$
    case $\Diamond\psi$:
      let $P = \texttt{Sat}(\mathcal{M}, \psi)$ **in**
      return $\mathcal{C}(P)$
    case $\psi \mathcal{U} \xi$: **return** `CheckUntil` $(\mathcal{M}, \psi, \xi)$

Algorithm 1. Decision procedure for the model checking problem

Theorem 3. *In a quasi-discrete closure model* $\mathcal{M}$, $\mathcal{M}, x \models \phi \mathcal{U} \psi$ *if and only if* $\mathcal{M}, x \models \phi$, *and for each path* $p : x \rightsquigarrow \infty$ *and* $l \in \mathbb{N}$, *if* $\mathcal{M}, p(l) \models \neg\phi$, *there is* $k \in \{1, \ldots, l\}$ *such that* $\mathcal{M}, p(k) \models \psi$.

Remark 3. Directly from Theorem 3 and from the definitions in Figure 2 we have also that in a quasi-discrete closure model $\mathcal{M}$:

1. $\mathcal{M}, x \models \phi \mathcal{R} \psi$ iff. there is $p : x \rightsquigarrow \infty$ and $k \in \mathbb{N}$ such that $\mathcal{M}, p(k) \models \psi$ and for each $j \in \{1, \ldots, k\}$ $\mathcal{M}, p(j) \models \phi$;
2. $\mathcal{M}, x \models \mathcal{G}\phi$ iff. for each $p : x \rightsquigarrow \infty$ and $i \in \mathbb{N}$, $\mathcal{M}, p(i) \models \phi$;
3. $\mathcal{M}, x \models \mathcal{F}\phi$ iff. there is $p : x \rightsquigarrow \infty$ and $i \in \mathbb{N}$ such that $\mathcal{M}, p(i) \models \phi$.

Note that, a point x satisfies $\phi \mathcal{R} \psi$ if and only if either ψ is satisfied by x or there exists a sequence of points after x, all satisfying ϕ, leading to a point satisfying both ψ and ϕ. In the second case, it is not required that x satisfies ϕ.

5 Model Checking *SLCS* Formulas

In this section we present a model checking algorithm for $SLCS$, which is a variant of standard CTL model checking [3]. Function `Sat`, presented in Algorithm 1, takes as input a finite quasi-discrete model $\mathcal{M} = ((X, \mathcal{C}_R), \mathcal{V})$ and an $SLCS$ formula ϕ, and returns the set of all points in X satisfying ϕ. The function is inductively defined on the structure of ϕ and, following a bottom-up approach, computes the resulting set via an appropriate combination of the recursive invocations of `Sat` on the sub-formulas of ϕ. When ϕ is $\top$, p, $\neg\psi$ or $\psi \wedge \xi$,

```
Function CheckUntil (M,ψ, ξ)
    let V = Sat(M, ψ) in
    let Q = Sat(M, ξ) in
    var T := B+(V ∪ Q)
    while T ≠ ∅ do
        T' := ∅
        for x ∈ T do
            N := pre(x) ∩ V
            V := V \ N
            T' := T' ∪ (N \ Q)
        T := T';
    return V
```

Algorithm 2. Checking until formulas in a quasi-discrete closure space

definition of $\texttt{Sat}(\mathcal{M}, \phi)$ is as expected. To compute the set of points satisfying $\Diamond\psi$, the closure operator $\mathcal{C}$ of the space is applied to the set of points satisfying ψ.

When ϕ is of the form $\psi \mathcal{U} \xi$, function `Sat` relies on the function `CheckUntil` defined in Algorithm 2. This function takes as parameters a model $\mathcal{M}$ and two $SLCS$ formulas ψ and ξ and computes the set of points in $\mathcal{M}$ satisfying $\psi \mathcal{U} \xi$ by removing from $V = \texttt{Sat}(\mathcal{M}, \psi)$ all the *bad* points. A point is *bad* if there exists a path passing through it, that leads to a point satisfying $\neg\psi$ without passing through a point satisfying ξ. Let $Q = \texttt{Sat}(\mathcal{M}, \xi)$ be the set of points in $\mathcal{M}$ satisfying ξ. To identify the *bad* points in V the function `CheckUntil` performs a *backward search* from $T = \mathcal{B}^+(V \cup Q)$. Note that any *path* exiting from $V \cup Q$ has to pass through points in T. Moreover, the latter only contains points that satisfy neither ψ nor ξ. Until T is empty, function `CheckUntil` first picks an element x in T and then removes from V the set of (bad) points N that can reach x in *one step*. To compute the set N we use the function $pre(x) = \{y \in X \mid (y, x) \in R\}$.[4] At the end of each iteration the set T is updated by considering the set of new discovered *bad points*.

Lemma 4. *Let X a finite set and $R \subseteq X \times X$. For any finite quasi-discrete model $\mathcal{M} = ((X, \mathcal{C}_R), \mathcal{V})$ and SLCS formula ϕ with k operators,* `Sat` *terminates in $\mathcal{O}(k \cdot (|X| + |R|))$ steps.*

Theorem 4. *For any finite quasi-discrete closure model $\mathcal{M} = ((X, \mathcal{C}), \mathcal{V})$ and SLCS formula ϕ, $x \in \texttt{Sat}(\mathcal{M}, \phi)$ if and only if $\mathcal{M}, x \models \phi$.*

6 A Model Checker for Spatial Logics

The algorithm described in Section 5 is available as a proof-of-concept tool[5]. The tool, implemented using the functional language OCaml, contains a generic

[4] Function *pre* can be *pre-computed* when the relation R is loaded from the input.
[5] Web site: `http://www.github.com/vincenzoml/slcs`.

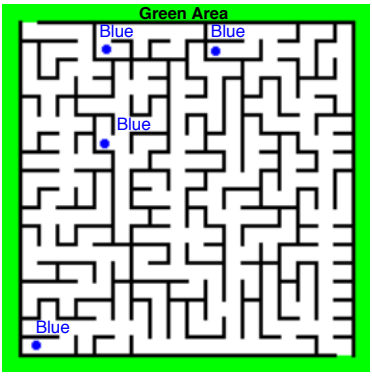


Fig. 3. A maze

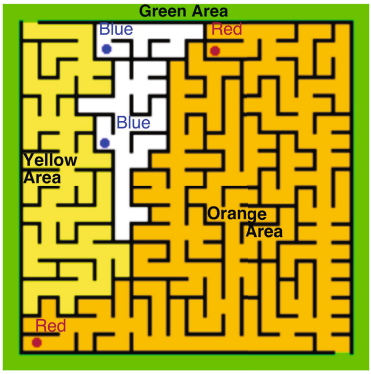


Fig. 4. Model checker output

```
Let reach(a,b) = !( (!b) U (!a) );
Let reachThrough(a,b) = a & reach((a|b),b);

Let toExit = reachThrough(["white"],["green"]);
Let fromStartToExit = toExit & reachThrough(["white"],["blue"]);
Let startCanExit = reachThrough(["blue"],fromStartToExit);

Paint "yellow" toExit;
Paint "orange" fromStartToExit;
Paint "red" startCanExit;
```

Fig. 5. Input to the model checker

implementation of a global model-checker using closure spaces, parametrised by the type of models.

An example of the tool usage is to approximately identify regions of interest on a digital picture (e.g., a map, or a medical image), using spatial formulas. In this case, digital pictures are treated as quasi-discrete models in the plane $\mathbb{Z} \times \mathbb{Z}$. The language of propositions is extended to simple formulas dealing with colour ranges, in order to cope with images where there are different shades of certain colours.

In Figure 3 we show a digital picture of a maze. The green area is the exit. The blue areas are start points. The input of the tool is shown in Figure 5, where the `Paint` command is used to invoke the global model checker and colour points satisfying a given formula. Three formulas, making use of the until operator, are used to identify interesting areas. The output of the tool is in Figure 4. The colour red denotes start points from which the exit can be reached. Orange and yellow indicate the two regions through which the exit can be reached, including and excluding a start point, respectively.

Fig. 6. Input: the map of a town

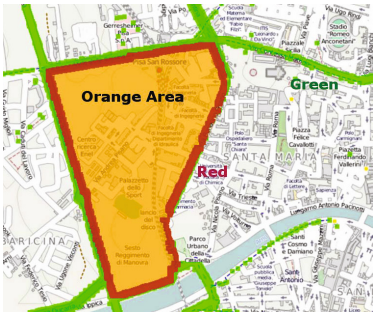


Fig. 7. Output of the tool

In Figure 6 we show a digital image[6] depicting a portion of the map of Pisa, featuring a red circle which denotes a train station. Streets of different importance are painted with different colors in the map. The model checker is used to identify the area surrounding the station which is delimited by main streets, and the delimiting main streets. The output of the tool is shown in Figure 7, where the station area is coloured in orange, the surrounding main streets are red, and other main streets are in green. We omit the source code of the model checking session for space reasons (see the source code of the tool). As a mere hint on how practical it is to use a model checker for image analysis, the execution time on our test image, consisting of about 250000 pixels, is in the order of ten seconds on a standard laptop equipped with a 2Ghz processor.

7 Conclusions and Future Work

In this paper, we have presented a methodology to verify properties that depend upon space. We have defined an appropriate logic, stemming from the tradition of topological interpretations of modal logics, dating back to earlier logicians such as Tarski, where modalities describe neighbourhood. The topological definitions have been lifted to a more general setting, also encompassing discrete, graph-based structures. The proposed framework has been extended with a spatial variant of the *until* operator, and we have also defined an efficient model checking procedure, which is implemented in a proof-of-concept tool.

As future work, we first of all plan to merge the results presented in this paper with temporal reasoning. This integration can be done in more than one way. It is not difficult to consider "snapshot" models consisting of a temporal model (e.g., a Kripke frame) where each state is in turn a closure model, and atomic formulas of the temporal fragment are replaced by spatial formulas. The various possible combinations of temporal and spatial operators, in linear and branching time, are examined for the case of topological models and basic modal

[6] © *OpenStreetMap contributors* – `http://www.openstreetmap.org/copyright`.

formulas in [18]. Snapshot models may be susceptible to state-space explosion problems as spatial formulas could need to be recomputed at every state. On the other hand, one might be able to exploit the fact that changes of space over time are incremental and local in nature. Promising ideas are presented both in [17], where principles of "continuous change" are proposed in the setting of closure spaces, and in [20] where spatio-temporal models are generated by locally-scoped update functions, in order to describe dynamic systems. In the setting of collective adaptive systems, it will be certainly needed to extend the basic framework we presented with metric aspects (e.g., distance-bounded variants of the until operator), and probabilistic aspects, using atomic formulas that are probability distributions. A thorough investigation of these issues will be the object of future research.

A challenge in spatial and spatio-temporal reasoning is posed by recursive spatial formulas, *a la* μ-calculus, especially on infinite structures with relatively straightforward generating functions (think of fractals, or fluid flow analysis of continuous structures). Such infinite structures could be described by topologically enhanced variants of ω-automata. Classes of automata exist living in specific topological structures; an example is given by *nominal automata* (see e.g., [4,14,21]), that can be defined using presheaf toposes [13]. This standpoint could be enhanced with notions of neighbourhood coming from closure spaces, with the aim of developing a unifying theory of languages and automata describing space, graphs, and process calculi with resources.

References

1. Aiello, M.: Spatial Reasoning: Theory and Practice. PhD thesis, Institute of Logic, Language and Computation, University of Amsterdam (2002)
2. Aiello, M., Pratt-Hartmann, I., van Benthem, J. (eds.): Handbook of Spatial Logics. Springer (2007)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
4. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: LICS, pp. 355–364. IEEE Computer Society Press (2011)
5. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective system behaviour: A tutorial. Perform. Eval. 70(5), 317–349 (2013)
6. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). Information and Computation 186(2), 194–235 (2003)
7. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 597–610. Springer, Heidelberg (2002)
8. Cardelli, L., Gordon, A.D.: Anytime, anywhere: Modal logics for mobile ambients. In: POPL, pp. 365–377. ACM (2000)
9. Chaintreau, A., Le Boudec, J., Ristanovic, N.: The age of gossip: Spatial mean field regime. In: SIGMETRICS, pp. 109–120. ACM, New York (2009)
10. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space - extended version. CoRR, abs/1406.6393 (2014)
11. De Nicola, R., Ferrari, G.L., Pugliese, R.: Klaim: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. 24(5), 315–330 (1998)

12. Del Bimbo, A., Vicario, E., Zingoni, D.: Symbolic description and visual querying of image sequences using spatio-temporal logic. IEEE Trans. Knowl. Data Eng. 7(4), 609–622 (1995)
13. Fiore, M.P., Staton, S.: Comparing operational models of name-passing process calculi. Information and Computation 204(4), 524–560 (2006)
14. Gabbay, M.J., Ciancia, V.: Freshness and name-restriction in sets of traces with names. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 365–380. Springer, Heidelberg (2011)
15. Gadducci, F., Lluch Lafuente, A.: Graphical encoding of a spatial logic for the π-calculus. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 209–225. Springer, Heidelberg (2007)
16. Galton, A.: The mereotopology of discrete space. In: Freksa, C., Mark, D.M. (eds.) COSIT 1999. LNCS, vol. 1661, pp. 251–266. Springer, Heidelberg (1999)
17. Galton, A.: A generalized topological view of motion in discrete space. Theoretical Computer Science 305(1-3), 111 (2003)
18. Kontchakov, R., Kurucz, A., Wolter, F., Zakharyaschev, M.: Spatial logic + temporal logic =? In: Aiello, et al. (eds.) [2], pp. 497–564
19. Kovalevsky, V.A.: Geometry of Locally Finite Spaces: Computer Agreeable Topology and Algorithms for Computer Imagery. House Dr. Baerbel Kovalevski (2008)
20. Kremer, P., Mints, G.: Dynamic topological logic. In: Aiello, et al. (eds.) [2], pp. 565–606
21. Kurz, A., Suzuki, T., Tuosto, E.: On nominal regular languages with binders. In: Birkedal, L. (ed.) FOSSACS 2012. LNCS, vol. 7213, pp. 255–269. Springer, Heidelberg (2012)
22. Rosenfeld, A.: Digital topology. The American Mathematical Monthly 86(8), 621–630 (1979)
23. Smyth, M.B., Webster, J.: Discrete spatial models. In: Aiello, et al. (eds.) [2], pp. 713–798
24. van Benthem, J., Bezhanishvili, G.: Modal logics of space. In: Handbook of Spatial Logics, pp. 217–298 (2007)
25. Yung Kong, T., Rosenfeld, A.: Digital topology: Introduction and survey. Computer Vision, Graphics, and Image Processing 48(3), 357–393 (1989)

Primal Infon Logic with Conjunctions as Sets

Carlos Cotrini[1], Yuri Gurevich[2], Ori Lahav[3], and Artem Melentyev[4]

[1] Swiss Federal Institute of Technology, Switzerland
[2] Microsoft Research, Redmond, WA, USA
[3] Tel Aviv University, Israel
[4] Ural Federal University, Russia

Abstract. Primal infon logic was proposed by Gurevich and Neeman as an efficient yet expressive logic for policy and trust management. It is a propositional multimodal subintuitionistic logic decidable in linear time. However in that logic the principle of the replacement of equivalents fails. For example, $(x \wedge y) \to z$ does not entail $(y \wedge x) \to z$, and similarly $w \to ((x \wedge y) \wedge z)$ does not entail $w \to (x \wedge (y \wedge z))$. Imposing the full principle of the replacement of equivalents leads to an NP-hard logic according to a recent result of Beklemishev and Prokhorov. In this paper we suggest a way to regain the part of this principle restricted to conjunction: We introduce a version of propositional primal logic that treats conjunctions as sets, and show that the derivation problem for this logic can be decided in linear expected time and quadratic worst-case time.

1 Introduction

Propositional infon logic is a version of propositional multimodal intuitionistic logic [7]. It is applicable for policy and trust management but the derivability problem for propositional infon logic is PSPACE-complete. Nevertheless, an expressive fragment of this logic, called *propositional primal infon logic* (**PIL**, in short), is decidable in linear time [7]. **PIL** is far below propositional infon logic in the time-complexity hierarchy. A natural problem arises how to extend the expressive power (and usefulness) of **PIL** keeping the logic feasible. In this paper, we present substantial progress toward this goal.

One of the main limitations of **PIL** is that it does not satisfy the principle of replacement of equivalents, that allows us to substitute a formula with an equivalent one in any context. For example, the formulas $x \wedge y$ and $y \wedge x$ are equivalent in **PIL** (i.e., each one is derivable from the other). However, $(x \wedge y) \to z$ and $(y \wedge x) \to z$ are not. In general, replacing a variable occurring in some formula with $x \wedge y$ is not the same as replacing it with $y \wedge x$. A similar situation occurs, e.g., with formulas of the form $(x \wedge y) \wedge w$ and $x \wedge (y \wedge w)$.

Imposing the full principle of replacement of equivalents on **PIL** makes it NP-hard [2]. Nevertheless, in this paper, we present an extension of **PIL**, called **SPIL**, that overcomes this limitation for conjunction. The idea behind **SPIL** is to treat conjunctions as sets of conjuncts (the 'S' in **SPIL** alludes to the word "set"). In other words any two conjunctions are viewed equivalent if the sets

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 236–249, 2014.

(not multisets!) of their conjuncts are the same, and the reasoning is done modulo this equivalence. For example, this equivalence relation identifies formulas $(x \wedge y) \to z$ and $(y \wedge x) \to z$.

This paper is organized as follows. First, we recall the syntax of **PIL** (Section 2). Then we define **SPIL** (Section 3), and we prove the local formula property for its Hilbertian calculus: any derivation of a formula X from a set Ω of formulas can be modified so that it uses only a small set of "local formulas" computable from $\Omega \cup \{X\}$. In Section 3.1 we present a Kripke-style semantics for **SPIL**. Finally, in Section 4, we present an efficient algorithm for the multi-derivability problem for **SPIL**. An implementation of the algorithm is available at `http://dkal.codeplex.com/` (in the context of Distributed Knowledge Authorization Language [3]).

Related work. We refer the reader to detailed related work sections: subsection 1.1 in the article [1] on propositional primal logic with disjunction, and section 6 in the article [5] on extensions of **PIL** with transitivity of implication. In addition, we note that proof systems in which derivations are performed modulo an equational theory between propositions were studied earlier in different contexts (see, e.g., [6]).

2 Preliminaries

We start with describing **PIL** (propositional primal infon logic), originally presented in [7]. We presume a set of propositional variables $\{v_1, v_2, \ldots\}$, a set of principal constants $\{p_1, p_2, \ldots\}$ and a constant $\top$ (used to denote an item of information that is known to all principals). The formulas of **PIL** are built from the propositional variables and $\top$ using the binary connectives $\wedge, \vee, \to$, and unary connectives of the form "q:" (called: *quotations*) where q ranges over principal constants. The intended meaning of a formula $q{:}x$ is that: the principal q said x. The size $sz(x)$ of a formula x is taken to be the number of connectives occurring in x. For any sequence of principal constants $q_1, q_2, \ldots, q_k$, we call the string $\boldsymbol{q} = q_1 : q_2 : \ldots q_k :$ a *quotation prefix* (where ϵ, the empty sequence, is a quotation prefix as well).

$$(\top)\ \frac{}{\boldsymbol{q}\,\top} \qquad (\wedge\mathrm{i})\ \frac{\boldsymbol{q}\,x \qquad \boldsymbol{q}\,y}{\boldsymbol{q}\,(x \wedge y)} \qquad (\wedge\mathrm{e})\ \frac{\boldsymbol{q}\,(x \wedge y)}{\boldsymbol{q}\,x} \qquad \frac{\boldsymbol{q}\,(x \wedge y)}{\boldsymbol{q}\,y}$$

$$(\vee\mathrm{i})\ \frac{\boldsymbol{q}\,x}{\boldsymbol{q}\,(x \vee y)} \qquad \frac{\boldsymbol{q}\,y}{\boldsymbol{q}\,(x \vee y)} \qquad (\to\mathrm{i})\ \frac{\boldsymbol{q}\,y}{\boldsymbol{q}\,(x \to y)} \qquad (\to\mathrm{e})\ \frac{\boldsymbol{q}\,x \qquad \boldsymbol{q}\,(x \to y)}{\boldsymbol{q}\,y}$$

Fig. 1. Calculus for **PIL**. $\boldsymbol{q}$ ranges over quotation prefixes and x, y over formulas.

Figure 1 provides a Hilbertian calculus defining **PIL**. For a set of formulas Γ (called *hypotheses*), a *derivation* D of a formula x from Γ in **PIL** is a finite tree

such that each node u is labeled with a formula $D(u)$. The root is labeled with x and leaves are labeled with (instances of) axioms or formulas in Γ. If a node u has children $u_1, u_2, \ldots, u_n$, then $D(u_1), \ldots, D(u_n)$ / $D(u)$ is an instance of an inference rule. The *size* of the derivation is the number of nodes in this tree.

Given two sets of formulas Γ and Δ, the problem of deciding which formulas in Δ are derivable from Γ in **PIL** is called *the multi-derivability problem for* **PIL**. This problem is decidable in linear time [4,7].

3 The Logic SPIL

We present an extension of **PIL** that we call **SPIL**. The letter 'S' alludes to the word "set" and reflects our intention to treat conjunctions as sets of conjuncts. To define **SPIL** we use an auxiliary notion of *abstract formulas*.

Definition 1. An equivalence relation $\sim$ between formulas is defined as follows: $x \sim y$ if x and y are related according to the reflexive transitive symmetric closure of the rewriting relation induced by the following term rewriting system:[1]

- $(x_1 \wedge x_2) \longrightarrow (x_2 \wedge x_1)$
- $((x_1 \wedge x_2) \wedge x_3) \longrightarrow (x_1 \wedge (x_2 \wedge x_3))$
- $(x_1 \wedge x_1) \longrightarrow x_1$
- $(x_1 \wedge \top) \longrightarrow x_1$
- $q{:}(x_1 \wedge x_2) \longrightarrow (q{:}x_1) \wedge (q{:}x_2)$
- $q{:}\top \longrightarrow \top$

Roughly speaking, we have $x \sim y$ if x and y are the same formulas modulo the following properties of $\wedge$: commutativity, associativity, idempotence, contraction of the identity element $\top$, as well as the distributivity of quotations over $\wedge$.

Example 1. The formula $(v_1 \rightarrow p_1{:}((p_1{:}v_1) \wedge v_2))$ is equivalent to $(((v_1 \wedge v_1) \wedge \top) \rightarrow (p_1{:}v_2 \wedge (p_1{:}p_1{:}v_1)))$.

Definition 2. *Abstract formulas* are equivalence classes of formulas under $\sim$. The size $sz(X)$ of an abstract formula X is defined as $\min\{sz(x) \mid x \in X\}$.

We use $X, Y, \ldots$ as metavariables for abstract formulas. The equivalence class of a formula x under $\sim$ is denoted by $[x]$. Since abstract formulas play a dominant role in **SPIL**, we will refer to them simply as *formulas*, where true (non-abstract) formulas will be called *concrete formulas*. We define several operations on formulas.

Definition 3. For two formulas X, Y and connective $* \in \{\rightarrow, \vee\}$, $X * Y := [x * y]$ for some $x \in X$ and $y \in Y$. Similarly, for a formula X and a quotation prefix $\boldsymbol{q}$, $\boldsymbol{q}\,X := [\boldsymbol{q}\,x]$ for some $x \in X$.

Definition 4.

- A formula X is called *conjunctive* if $X = [\top]$ or $X = [x \wedge y]$ for concrete formulas x, y satisfying $x \not\sim y$, $x \not\sim \top$ and $y \not\sim \top$.

[1] Recall that rewriting rules of a term rewriting system may be applied under any context, and not necessarily on the topmost level.

- A finite set of non-conjunctive formulas with at least two elements is called a *conjunction set*.
- For a conjunction set S, $\bigwedge S := [(\cdots((x_1 \wedge x_2) \wedge x_3)\ldots) \wedge x_n]$ for some concrete formulas $x_1,\ldots,x_n$ such that $S = \{[x_1],\ldots,[x_n]\}$.

It is easy to see that these operations are well-defined. In particular, the choices of concrete formulas is immaterial. Note that we use conjunction *sets* rather than *multisets*, and that, by definition, conjunction sets contain at least two members.

Proposition 1. *$\bigwedge S$ is conjunctive for every conjunction set S.*

The next proposition allows us to use inductive definitions and prove claims by induction on size of formulas.

Proposition 2. *Every formula X is either non-conjunctive and exactly one of the following holds:*

- *$sz(X) = 0$ and $X = [v]$ for a unique propositional variable v.*
- *$X = Y * Z$ for unique formulas Y and Z and $* \in \{\rightarrow, \vee\}$. In this case $sz(X) = sz(Y) + sz(Z) + 1$.*
- *$X = q{:}Y$ for unique principal constant q and formula Y. In this case $sz(X) = sz(Y) + 1$, and Y is non-conjunctive.*

or else X is conjunctive and either $sz(X) = 0$ and $X = [\top]$, or $X = \bigwedge S$ for a unique conjunction set S. In the latter case, $sz(Y) < sz(X)$ for every $Y \in S$.

$$(\tilde{\top})\ \frac{}{[\top]} \qquad (\tilde{\wedge}\mathrm{i})\ \frac{X_1 \quad X_2 \quad \ldots \quad X_n}{\bigwedge S} \text{ where } S = \{X_1,\ldots,X_n\} \text{ and } n \geq 2$$

$$(\tilde{\wedge}\mathrm{e})\ \frac{\bigwedge S}{X} \text{ where } X \in S \qquad (\tilde{\vee}\mathrm{i})\ \frac{\boldsymbol{q}\,X}{\boldsymbol{q}\ (X \vee Y)} \qquad \frac{\boldsymbol{q}\,Y}{\boldsymbol{q}\ (X \vee Y)}$$

$$(\tilde{\rightarrow}\mathrm{i})\ \frac{\boldsymbol{q}\,Y}{\boldsymbol{q}\ (X \rightarrow Y)} \qquad (\tilde{\rightarrow}\mathrm{e})\ \frac{\boldsymbol{q}\,X \qquad \boldsymbol{q}\ (X \rightarrow Y)}{\boldsymbol{q}\,Y}$$

Fig. 2. Calculus for **SPIL**. $\boldsymbol{q}$ ranges over quotation prefixes, X, Y over formulas, and S over conjunction sets.

SPIL is defined via the Hilbertian calculus given in Figure 2. The definitions of a derivation and its size are naturally adopted to this Hilbertian calculus. Note that derivations now consist of *abstract* formulas. We write $\Omega \vdash X$ to denote that the abstract formula X has a derivation from the set Ω of abstract formulas in **SPIL**.

Definition 5. The consequence relation $\vdash$ between concrete formulas in **SPIL** is given by: $\Gamma \vdash x$ if $\{[y] \mid y \in \Gamma\} \vdash [x]$.

Note that the language of the concrete formulas is that of **PIL**. Abstract formulas are used only for defining this consequence relation.

Theorem 1. *If Γ entails x in* **PIL**, *then it does so in* **SPIL** *as well.*

Next, we show that **SPIL** enjoys a locality property similar to that of **PIL**, which allows one to confine derivations of X from Ω to those built from a certain small set of formulas computable from X and Ω. This property is essential for the correctness of the decision algorithm for **SPIL**.

Definition 6. The set of formulas that are *local* to a formula X is inductively defined by: (*a*) X is local to X; (*b*) If $\boldsymbol{q}\,(Y * Z)$ is local to X (for $* \in \{\rightarrow, \vee\}$ and quotation prefix $\boldsymbol{q}$) then so are $\boldsymbol{q}\,Y$ and $\boldsymbol{q}\,Z$; and (*c*) If $\bigwedge S$ is local to X (for conjunction set S) then so is every $Y \in S$. A formula is local *to a set* Ω of formulas if it is local to some $X \in \Omega$.

Definition 7. A derivation of a formula X from a set Ω of formulas is called *local* if all node formulas of the derivation are local to $\Omega \cup \{X\}$.

Theorem 2. *Any shortest derivation of X from Ω in* **SPIL** *is local.*

The following definition will be useful in the sequel.

Definition 8. A quotation prefix $\boldsymbol{q}$ is *local* to a formula X if some formula of the form $\boldsymbol{q}\,Y$ is local to X. A quotation prefix is local *to a set* Ω of formulas if it is local to some $X \in \Omega$.

3.1 Semantics

We adapt the semantics for **PIL** presented in [4,7] to **SPIL**.

Definition 9. A *Kripke model* is any structure M whose vocabulary comprises of (i) binary relations S_q where q ranges over the principal constants and (ii) unary relations V_X where X ranges over non-conjunctive formulas. The elements of (the universe of) M are called *worlds*.

Definition 10. Given a Kripke model M, we define when a world w *satisfies* a formula X, symbolically $w \vDash X$, by induction on $sz(X)$, distinguishing the cases according to Proposition 2:

1. $X = [\top]$: $w \vDash X$ for every w.
2. $X = [v]$ (where v is a propositional variable): $w \vDash X$ if $w \in V_{[v]}$.
3. $X = Y \rightarrow Z$: $w \vDash X$ if $w \vDash Z$ or ($w \nvDash Y$ and $w \in V_X$).
4. $X = Y \vee Z$: $w \vDash X$ if $w \vDash Y$ or $w \vDash Z$ or $w \in V_X$.
5. $X = q{:}Y$ (for non-conjunctive formula Y): $w \vDash X$ if $w' \vDash Y$ for all w' with wS_qw'.
6. $X = \bigwedge S$: $w \vDash X$ if $w \vDash Y$ for all $Y \in S$.

A world w satisfies *a set* Ω of formulas if it satisfies every $X \in \Omega$.

Theorem 3 (Soundness and Completeness). *Let Γ be a set of concrete formulas and x a concrete formula. $\Gamma \vdash x$ if and only if, for every Kripke model and world w, $w \vDash [x]$ whenever w satisfies $\{[y] \mid y \in \Gamma\}$.*

Remark 1. One of our referees wondered whether the full principle of replacement of equivalents holds in **SPIL**. It does not. Intuitively the reason is that, while **SPIL** generously enriches the algebra of conjunction, it imposes only mild restrictions on implication. Here is a example showing that the full principle of replacement of equivalents fails: $(x \wedge y) \to z \nvdash (x \wedge (x \to y)) \to z$ while $x \wedge y$ and $x \wedge (x \to y)$ are interderivable. This can be easily verified using our Kripke semantics.

4 A Decision Algorithm

In this section we present an efficient decision algorithm for the the multi-derivability problem for **SPIL**.

Definition 11. The *multi-derivability problem for* **SPIL** is defined as follows. Given two sequences of concrete formulas, called *concrete hypotheses* and *concrete queries* respectively, decide which concrete queries are derivable from the concrete hypotheses in **SPIL**, and print them.

Theorem 4. *There is a randomized algorithm that solves the multi-derivability problem for* **SPIL** *in expected linear time and worst-case quadratic time.*

Note that in "expected linear time" the average is taken for internal random choices during the execution, while assuming any input. We employ the same standard computation model of analysis of algorithms used in [4], according to which the registers are of size $O(\log n)$ where n is the size of the input, and the basic register operations are constant time. We also presume a function Random that generates $\lceil \log(n) \rceil$ random bits in constant time.

The rest of this paper is devoted to prove Theorem 4. The algorithm has two main stages. First, we construct a data structure that succinctly represents the input (Sections 4.1 and 4.2). Then, we use this data structure to compute the derivable concrete queries (Section 4.3).

4.1 Input Parse Dag and Local Prefixes Dictionary

We refer to the abstract formulas that correspond to the concrete hypotheses simply as *hypotheses*, and similarly to these of the concrete queries as *queries*. A formula (quotation prefix) is called *a local input formula (local prefix)* if it is local to the set of hypotheses or the set of queries (see Definitions 6 and 8). The input is represented in a directed acyclic graph (dag, for short) data structure.[2] We assume that each node u is uniquely identified by a constant-size key, denoted by $\mathrm{Key}(u)$ (e.g., its memory address), stores the keys of its children in a list $\mathrm{Ch}(u)$, and of its parents in a corresponding list $\mathrm{Pa}(u)$. To handle quotation prefixes, we will use of the following auxiliary data structure:

[2] By graph we actually mean multigraph, where parallel edges are allowed.

Definition 12. A *local prefixes dictionary* for a given input is a data structure that assigns a unique constant-size key $\mathrm{Key}(\boldsymbol{q})$ to every local input quotation prefix $\boldsymbol{q}$. Given such a key k, we will denote by $\mathrm{Prf}(k)$ the quotation prefix $\boldsymbol{q}$ such that $\mathrm{Key}(\boldsymbol{q}) = k$.

Note that the *trie of local prefixes* as defined in [4] is a particular implementation of a local prefixes dictionary, where $\mathrm{Key}(\boldsymbol{q})$ is taken to be the memory address of the trie node that corresponds to $\boldsymbol{q}$. Given a local prefixes dictionary, our dag data structures are defined as follows.

Definition 13. A *parse dag* is a rooted dag in which every node u is decorated with two additional (constant-size) fields: $\mathrm{Label}(u)$ and $\mathrm{PrfKey}(u)$. Its root r has two children denoted by r_h and r_q, where $\mathrm{Label}(r) = \mathrm{Label}(r_h) = \mathrm{Label}(r_q) =$ `nil` and $\mathrm{PrfKey}(r) = \mathrm{PrfKey}(r_h) = \mathrm{PrfKey}(r_q) = \mathrm{Key}(\epsilon)$. All other nodes are called *regular nodes*. For each regular node u, $\mathrm{Label}(u)$ is $\top$,$\rightarrow$,$\vee$,$\wedge$ or a propositional variable, and $\mathrm{PrfKey}(u)$ holds a key of some local input quotation prefix, such that:

1. If $\mathrm{Label}(u)$ is $\top$ or a propositional variable, then $\mathrm{Ch}(u)$ is empty.
2. If $\mathrm{Label}(u)$ is $\rightarrow$ or $\vee$, then $\mathrm{Ch}(u)$ contains exactly two keys.
3. If $\mathrm{Label}(u)$ is $\wedge$, then $\mathrm{Ch}(u)$ contains at least one key.
4. If u is a child of v, then $\mathrm{Prf}(\mathrm{PrfKey}(v))$ is a prefix of $\mathrm{Prf}(\mathrm{PrfKey}(u))$.

Each node in a parse dag naturally represents a (concrete and abstract) formula. Formally, this relation is defined as follows.

Notation 5. *For a regular node u, we denote $Prf(PrfKey(u))$ by $Prf(u)$.*

Notation 6. *Given two quotation prefixes $\boldsymbol{q}$ and $\boldsymbol{p}$, we denote by $\boldsymbol{p}\backslash\boldsymbol{q}$ the quotation prefix $\boldsymbol{r}$, such that $\boldsymbol{pr} = \boldsymbol{q}$, or ϵ if such $\boldsymbol{r}$ does not exist.*

Definition 14. The *complete concrete formula of a regular node u with respect to a quotation prefix $\boldsymbol{q}$* is denoted by $\mathrm{F}(u, \boldsymbol{q})$, and defined by:

1. If u has no children, then $\mathrm{F}(u, \boldsymbol{q}) = (\boldsymbol{q}\backslash\mathrm{Prf}(u))\mathrm{Label}(u)$.
2. If $\mathrm{Label}(u) = *$ for $* \in \{\rightarrow, \vee\}$, then $\mathrm{F}(u, \boldsymbol{q})$ is $(\boldsymbol{q}\backslash\mathrm{Prf}(u))$ $(\mathrm{F}(u_1, \mathrm{Prf}(u)) * \mathrm{F}(u_2, \mathrm{Prf}(u)))$ where u_1 and u_2 are the first and second children of u (respectively).
3. If $\mathrm{Label}(u) = \wedge$, then $\mathrm{F}(u, \boldsymbol{q})$ is $(\boldsymbol{q}\backslash\mathrm{Prf}(u))((\cdots(\mathrm{F}(u_1, \mathrm{Prf}(u)) \wedge \mathrm{F}(u_2,$ $\mathrm{Prf}(u)))\ldots) \wedge \mathrm{F}(u_k, \mathrm{Prf}(u)))$ where $u_l, \ldots, u_k$ are u's children in the order they occur in $\mathrm{Ch}(u)$.

The *complete concrete formula of a regular node u* is denoted by $\mathrm{F}(u)$ and defined to be $\mathrm{F}(u, \epsilon)$. The *complete (abstract) formula of a regular node u* is denoted by $\tilde{\mathrm{F}}(u)$ and defined to be $[\mathrm{F}(u)]$.

Definition 15. A parse dag *for input* $x_1, \ldots, x_k \vdash y_1, \ldots, y_m$ is any parse dag that satisfies the following conditions: (1) $\{\tilde{\mathrm{F}}(u) \mid \mathrm{Key}(u) \in \mathrm{Ch}(r_h)\} = \{[x_1], \ldots, [x_k]\}$; (2) $\{\tilde{\mathrm{F}}(u) \mid \mathrm{Key}(u) \in \mathrm{Ch}(r_q)\} = \{[y_1], \ldots, [y_m]\}$; and (3) Every child u of r_q is decorated with a list $\mathrm{Inputs}(u)$ of all y_i's that satisfy $y_i \in \tilde{\mathrm{F}}(u)$.

Note that the input parse tree as defined in [4] (ignoring the edge labels) is also an input parse dag. For the next stage, we should ensure that there are no two different nodes that represent the same formula. Thus we are interested in a *compressed* input parse dag, as defined next.

Definition 16. A node u in a parse dag D is *unique* if $\tilde{\mathrm{F}}(u') \neq \tilde{\mathrm{F}}(u)$ for any $u' \neq u$. D is called *compressed* if its nodes are all unique, and Label(u) is not $\wedge$ or $\top$ whenever u is a child of a node labeled with $\wedge$.

Proposition 3. *Consider a compressed input parse dag. For every local input formula X, there is exactly one regular node u such that $\tilde{F}(u) = X$.*

Theorem 7. *There is a randomized algorithm with expected linear time and worst-case quadratic time complexities, that constructs a local prefixes dictionary and a compressed input parse dag for a given input.*

4.2 Construction of a Compressed Input Parse Dag

This section is devoted to prove Theorem 7. To facilitate the exposition and the analysis, we will present this algorithm as a composition of sub-algorithms. Initially, we construct (in linear time) a local prefixes dictionary and an initial (uncompressed) input parse dag (in the form of a trie of local prefixes and an input parse tree) exactly as done in [4]. It remains to modify the parse tree into a compressed parse dag. First, we reformat the tree as detailed in Algorithm 4.1. Roughly speaking, this step accounts for the associativity of conjunction. Its time complexity is $O(N)$, where N denotes the number of leaves in the initial parse tree.

Algorithm 4.1. Initial reformatting

1: Traverse the initial parse dag in depth-first manner. Suppose that u, the node currently visited, is labeled with $\wedge$ or $\top$, and that its parent v is labeled with $\wedge$. In that case, for each child w of u, make w a child of v, and delete u from the dag. If v is left with no children, set its label to $\top$.

Time complexity: $O(N)$

Next, we "compress" the resulting tree into a dag. This process requires several additional data structures and fields:

1. A work list C of length N, initialized with (the keys of) all leaf nodes.
2. Two auxiliary arrays A and HT (Hash Table) of length $M = 2^{\lceil \log_2(N) \rceil}$. The entries of A are node keys, while those of HT are lists of keys. Initially $A[i] = -1$ and $HT[i] = \emptyset$ for all $i < M$.
3. Numeric fields Counter(u) and Hash(u) for each node u, initialized with 0 and a random number $< M$ (respectively).

The compression works iteratively using the work list C. In each iteration, the nodes in C are made unique. Initially, C includes all leaf nodes. When a node u is removed from C, it increments the counter in its parent v. If $\mathrm{Counter}(v)$ reaches $|\mathrm{Ch}(v)|$ (the length of the list $\mathrm{Ch}(v)$), v is added to C for the next iteration. Thus, the following invariant is preserved:

Invariant 8. *Children of nodes in C are all unique, and each node in C or ancestor of a node in C has a unique parent node.*

Compression of the Leaves. Before the first iteration, the work list C includes all leaf nodes. They are compressed using a *plagiarism checker*:

Definition 17. An element a_j in an array $L = (a_0, \ldots, a_{k-1})$ is *original* if $a_i \neq a_j$ for any $i < j$. If a_i is original, $i \leq j$ and $a_i = a_j$ then a_i is the *original of* a_j. A *plagiarism checker* for the array L is an array B of length k such that every $a_{B[j]}$ is the original of a_j.

Theorem 9. *There is an algorithm that, given an array L of d-tuples of natural numbers $< M$ and an array A of length M initialized with (-1)'s, computes the plagiarism checker B for L and re-initializes the array A with (-1)'s (so it can be reused to compute future plagiarism checkers). This algorithm takes $O(|L|d)$ time.*

The plagiarism checker is computed on an array L that includes the *extended labels* of the leaf nodes. For each node u in C, the extended label of u, denoted by $\mathrm{EL}(u)$, is a constant-size tuple that satisfies the following property: for every two nodes u_1 and u_2 in C, $\mathrm{EL}(u_1) = \mathrm{EL}(u_2)$ iff $\tilde{\mathrm{F}}(u_1) = \tilde{\mathrm{F}}(u_2)$. For a leaf node u, the extended label $\mathrm{EL}(u)$ is taken to be the ordered pair $(\mathrm{PrfKey}(u), \mathrm{Label}(u))$ if $\mathrm{Label}(u)$ is a propositional variable, or just $\top$ if $\mathrm{Label}(u) = \top$. It is easy to see that this definition of $\mathrm{EL}(u)$ guarantees the required property for the (leaf) nodes in C. From this observation, Algorithm 4.2 for compression of the leaves follows. Note that computing the extended label of a leaf takes constant time. Thus computing L and U takes $O(N)$ time. Since extended labels have constant length, the plagiarism checker can be computed in $O(N)$ time as well.

Compression of Internal Nodes. After applying Algorithm 4.2 in the first iteration, all leaf nodes are unique. In addition, Algorithm 4.2 prepares it for the next iteration, so it includes all nodes whose children are all unique. In fact, the next iteration also applies Algorithm 4.2, with a different definition of the extended labels. We refer to the nodes of C whose label is $\rightarrow$ or $\vee$ as *binary nodes*, and to these whose label is $\wedge$ as *set nodes*. The extended label for the binary nodes is simple, as we can take $\mathrm{EL}(u)$ of a binary node u in C with first child v and second child w to be the ordered tuple $(\mathrm{PrfKey}(u), \mathrm{Label}(u), \mathrm{Key}(v), \mathrm{Key}(w))$.

Proposition 4. *For two binary nodes u_1 and u_2 in C, $\mathrm{EL}(u_1) = \mathrm{EL}(u_2)$ iff $\tilde{F}(u_1) = \tilde{F}(u_2)$.*

Algorithm 4.2. Compression of the nodes in the work list

Copy the work list C to an array U, and set $C \leftarrow \emptyset$.
Compute an array L with corresponding extended labels, i.e. $L[i] = \mathrm{EL}(U[i])$.
Compute the plagiarism checker B of L. ▷ Theorem 9
For i from 0 to $|U| - 1$
  Let u and w be the nodes with keys $U[i]$ and $U[B[i]]$ (respectively).
  Let v be the parent of u.
  If $i \neq B[i]$ then
    Remove u from the parse dag.
    If $v = r_q$, append $Inputs(u)$ to $Inputs(w)$.
    Replace $U[i]$ in $\mathrm{Ch}(v)$ by $U[B[i]]$.
  Increment $\mathrm{Counter}(v)$.
  If v is regular and $\mathrm{Counter}(v) = |\mathrm{Ch}(v)|$ then add v to the work list C.

Time complexity: $O(|C|)$

The compression of the set nodes, however, is more involved, and requires some preprocessing to account for the idempotence of $\wedge$, and to compute the extended labels of the set nodes. Several additional notations are used in this preprocessing stage:

- N_C denotes the sum $\sum |\mathrm{Ch}(u)|$ for all set nodes in C.
- For each set node u in C, $CH(u) = \{w \mid w \text{ is a child of } u\}$.

The preprocessing for the set nodes consists of two steps. First, we reformat the parse dag, by removing duplicate children of set nodes, as well as contracting set nodes that are left with only one child (this may add new binary nodes to C). Algorithm 4.3 provides the technical details. Intuitively, this step accounts for the idempotence of $\wedge$.

Algorithm 4.3. Reformatting of set nodes in C

For each set node u of in C
  Copy $\mathrm{Ch}(u)$ to an array U.
  Compute the plagiarism checker B of $\mathrm{Ch}(u)$. ▷ Theorem 9
  $\mathrm{Ch}(u) \leftarrow \emptyset$.
  For i from 0 to $|B|$
    If $B[i] = i$, then append $U[i]$ to $\mathrm{Ch}(u)$.
  If $|\mathrm{Ch}(u)| = 1$ then
    Let v be u's parent and w be u's child.
    Remove u from parse dag and replace it with w in $\mathrm{Ch}(v)$.
    If $v = r_q$, append $\mathrm{Inputs}(u)$ to $\mathrm{Inputs}(w)$
    Increment $\mathrm{Counter}(v)$.
    If v is regular and $\mathrm{Counter}(v) = |\mathrm{Ch}(v)|$, add v to the work list C.

Time complexity: $O(N_C)$.

Next, we compute the extended labels for the set nodes. This step involves a hash table, where the hash function assigns to each node u the initially chosen

random number $\mathrm{Hash}(u)$. Note that $(\wedge, \mathrm{Ch}(u))$ cannot serve as an extended label (since two set nodes with different permutations of the same list of children would have diffrent extended labels).

Compute Extended Labels. We assume that each set node u is decorated with an additional field called *set label* and denoted by $\mathrm{SL}(u)$. For each set node u in C, $\mathrm{SL}(u)$ is initialized with $\mathrm{Hash}(u_1) \oplus \cdots \oplus \mathrm{Hash}(u_k)$, where $u_1, \ldots, u_k$ are the children of u whose keys are listed in $\mathrm{Ch}(u)$, and $\oplus$ is the bitwise XOR operation (between binary representations of numbers). The computation of $\mathrm{SL}(u)$ takes $O(|\mathrm{Ch}(u)|)$ time for each node u. Hence the computation for all set nodes in C takes $O(N_C)$ time. Note that for two set nodes u and v with $CH(u) = CH(v)$, we have $\mathrm{SL}(u) = \mathrm{SL}(v)$; the converse, however, is "almost always" true as the following lemma shows.

Lemma 1. *For every two set nodes,* $P(\mathrm{SL}(u) = \mathrm{SL}(v))$ *is* 1 *if* $CH(u) = CH(v)$, *and* $1/M$ *otherwise.*

It follows that SL cannot serve as an extended label for the set nodes. To generate the extended labels EL, we use a hash table for detecting and fixing collisions in SL. This is described in Algorithm 4.4. We explain the time complexity for this computation. Let $u_1, u_2, \ldots, u_{|C|}$ be the set nodes in C and suppose that the loop in line 1 process them in that order. For $i < j \leq |C|$, let X_{ij} be a random variable which takes value 1, if $\mathrm{EL}(u_i) = (\wedge, \mathrm{Key}(u_i))$ and $\mathrm{SL}(u_i) = \mathrm{SL}(u_j)$; and 0, otherwise. Let T be the random variable that gives the time complexity for this computation. T is the sum over j of the time needed to compute $\mathrm{EL}(u_j)$. To compute $\mathrm{EL}(u_j)$ we check the nodes in the entry $HT[\mathrm{SL}(u_j)]$. The length of $HT[\mathrm{SL}(u_j)]$ is $\sum_{i=1}^{j-1} X_{ij}$ and each comparison (the check if $CH(u) = CH(v)$) takes $O(|\mathrm{Ch}(u_j)|)$ time. Therefore, $T = \sum_{j=1}^{|C|} \left(O(|\mathrm{Ch}(u_j)|) \cdot \sum_{i=1}^{j-1} X_{ij} \right)$. In the worst-case, for any two set nodes u and v we have $\mathrm{SL}(u) = \mathrm{SL}(v)$ but $CH(u) \neq CH(v)$. This yields an execution time of $O(|C| \cdot N_C)$. To see that $E(T) = O(N_C)$, it suffices to show that $\sum_{i=1}^{j-1} E(X_{ij})$ is constant. Now, $E(X_{ij}) = P(X_{ij} = 1) \leq P(\mathrm{SL}(u_i) = \mathrm{SL}(u_j))$, and by Lemma 1 we obtain that $E(X_{ij}) \leq 1$ if $CH(u_i) = CH(u_j)$, and $E(X_{ij}) \leq 1/M$ otherwise. Algorithm 4.4 stores at most one set node u_i with $CH(u_i) = CH(u_j)$ in $HT[\mathrm{SL}(u_j)]$. Hence, $\sum_{i=1}^{j-1} E(X_{ij}) \leq 1 + \frac{j-2}{M} \leq 1 + \frac{N}{M} \leq 2$.

Compression of Internal Nodes. Equipped with extended labels for all nodes in the work list C, we apply the compression for these nodes. Since each two nodes u_1 and u_2 in C have $\tilde{\mathrm{F}}(u_1) = \tilde{\mathrm{F}}(u_2)$ iff $\mathrm{EL}(u_1) = \mathrm{EL}(u_2)$, we can compress the nodes in C exactly as we did for the leaves using Algorithm 4.2. This algorithm also prepares C for the next iteration.

This concludes the computation of a compressed parse dag from the parse tree. Algorithm 4.5 gives a summary of this construction. To see the time complexity of Algorithm 4.5, note that the inner step of the loop takes expected time proportional to the number of nodes in C plus the number of their children. Since every node is added to C exactly one time, summing this over all iterations we

Algorithm 4.4. Computing extended labels for set nodes

For each set node u in C
    If $CH(u) = CH(v)$ for some $v \in HT[\mathrm{SL}(u)]$ then
        $\mathrm{EL}(u) \leftarrow (\wedge, \mathrm{Key}(v))$.
    Else
        $\mathrm{EL}(u) \leftarrow (\wedge, \mathrm{Key}(u))$.
        Append u to $HT[\mathrm{SL}(u)]$.
For each set node u in C, set $HT[\mathrm{SL}(u)] \leftarrow \emptyset$.

Expected time complexity: $O(N_C)$
Time complexity (worst-case): $O(|C| \cdot N_C)$

get expected time proportional to the number of nodes. In a similar way, we get $O(N^2)$ time in the worst-case for the inner loop. The complexities of all other steps were explained above. Finally, note that N (the number of leaves in the initial input tree) is clearly less than the length of the input.

Algorithm 4.5. Construction of a compressed parse dag

Construct a parse tree and a local prefixes dictionary
Perform initial reformatting of the parse tree. ▷ Algorithm 4.1
Construct a work list C initialized with a list of the (keys of) leaf nodes, field Counter(u) for each node u initialized with 0, arrays $A = (-1, -1, \ldots, -1)$ and $HT = (\emptyset, \emptyset, \ldots, \emptyset)$ of length M, and a field Hash(u) for each node u initialized with a random number $< M$.
Compress the nodes in C. ▷ Algorithm 4.2
While C is not empty
    Reformat the set nodes. ▷ Algorithm 4.3
    Compute a set label SL(u) for every set node u.
    Compute an extended label EL(u) for every set node u. ▷ Algorithm 4.4
    Compress the nodes in C. ▷ Algorithm 4.2

Expected time complexity: $O(N)$
Worst time complexity: $O\left(N^2\right)$

4.3 Deriving Local Formulas

The second stage of algorithm computes all derivable queries. This is done similarly to the corresponding stage for **PIL** [4]. First, we traverse the parse dag and decorate each regular node u with a boolean flag Der(u). It is initialized to `0`, unless $\mathrm{Label}(u) = \top$ or u represents a hypothesis (u is a child of r_h) in which case Der(u) is initialized to `1`. $\mathrm{Der}(u) =$ `0` indicates that $\tilde{\mathrm{F}}(u)$ has not been derived from the hypotheses yet, and in this case we say that u is *raw*. $\mathrm{Der}(u) =$ `1` indicates that $\tilde{\mathrm{F}}(u)$ has been derived from the hypotheses, and u is called *pending*. We also construct a *pending queue*, that contains all pending nodes. To *make pending* a node u means to insert u to the pending queue and set $\mathrm{Der}(u) =$ `1`. The following invariant holds throughout the execution of the algorithm.

Invariant 10. *Whenever a node u becomes pending, the formula $\tilde{F}(u)$ is derivable from the hypotheses in* **SPIL**.

To *apply a rule R to u* means to make pending every raw node w for which there are pending nodes $v_1, v_2, \ldots, v_k$, such that $u \in \{v_1, v_2, \ldots, v_k\}$ and $\tilde{\mathrm{F}}(v_1), \ldots, \tilde{\mathrm{F}}(v_k)$ / $\tilde{\mathrm{F}}(w)$ is an instance of the rule R. The algorithm repeatedly takes a node u from the pending queue, applies as many rules to it as possible and then removes u from the pending queue. The algorithm terminates when the pending queue is empty. We explain how to apply each rule R to a node u, and show (for several cases) that these applications preserve Invariant 10. Note that an additional numeric field Counter(u) (initialized to 0) is used for each node u labeled with $\wedge$.

($\tilde{\wedge}$e) If $\mathrm{Label}(u) = \wedge$, then make pending every raw child of u.
Justification: Let $u_1, \ldots, u_k$ be the children of u in the order they appear in $\mathrm{Ch}(u)$. Then $\tilde{\mathrm{F}}(u) = [\mathrm{Prf}(u)((\cdots(\mathrm{F}(u_1, \mathrm{Prf}(u)) \wedge \mathrm{F}(u_2, \mathrm{Prf}(u))\ldots) \wedge \mathrm{F}(u_k, \mathrm{Prf}(u)))]$. Since u is pending, $\tilde{\mathrm{F}}(u)$ is derivable. This entails that $\tilde{\mathrm{F}}(u_i) = [\mathrm{Prf}(u)(\mathrm{F}(u_i, \mathrm{Prf}(u))]$ is derivable as well. To see this, note that $\boldsymbol{q}\, x_i$ is derivable from $\boldsymbol{q}\,((\cdots(x_1 \wedge x_2)\ldots) \wedge x_k)$ in **PIL** for every concrete formulas $x_1, \ldots, x_k$, quotation prefix $\boldsymbol{q}$, and $1 \leq i \leq k$. By Theorem 1, we have that $\tilde{\mathrm{F}}(u) \vdash \tilde{\mathrm{F}}(u_i)$.

($\tilde{\wedge}$i) For every raw parent w of u labeled with $\wedge$, increment Counter(w) and make w pending if it exceeds the number of children of w.
Justification: Let $u_1, \ldots, u_k$ be the children of w in the order they appear in $\mathrm{Ch}(w)$. Then $\tilde{\mathrm{F}}(w) = [\mathrm{Prf}(w)((\cdots(\mathrm{F}(u_1, \mathrm{Prf}(w)) \wedge \mathrm{F}(u_2, \mathrm{Prf}(w))\ldots) \wedge \mathrm{F}(u_k, \mathrm{Prf}(w)))]$. If Counter($w$) was incremented k times, then each $\tilde{\mathrm{F}}(u_i) = [\mathrm{Prf}(w)(\mathrm{F}(u_i, \mathrm{Prf}(w))]$ is derivable. This entails that $\tilde{\mathrm{F}}(w)$ is derivable as well (again using Theorem 1, since $\boldsymbol{q}\,((\cdots(x_1 \wedge x_2)\ldots) \wedge x_k)$ is derivable from $\boldsymbol{q}\, x_1, \ldots, \boldsymbol{q}\, x_k$ in **PIL** for every concrete formulas $x_1, \ldots, x_k$ and quotation prefix $\boldsymbol{q}$).

($\tilde{\vee}$i) Make pending every raw parent w of u labeled with $\vee$.

($\tilde{\rightarrow}$i) For every raw parent w of u such that Label(w) is $\rightarrow$ and u is the second child of w, make w pending.

($\tilde{\rightarrow}$e) $\tilde{\mathrm{F}}(u)$ can be used as the left or the right premise of ($\tilde{\rightarrow}$e). Accordingly, we have two substeps: (1) For every pending parent w of u, such that Label(w) is $\rightarrow$ and u is the first child of w, make the second child of w pending if it is raw; (2) If Label(u) is $\rightarrow$ and the first child u_1 of u is pending, then make pending the second child u_2 of u if it is raw.

When the pending queue is empty, the algorithm prints a list of the derivable concrete queries. To do so, walk through the nodes $u_1, \ldots, u_m$ that represent queries (i.e. the children of the node r_q). If $\mathrm{Der}(u_i) = \mathtt{1}$ then print the strings in $\mathrm{Inputs}(u_i)$. Since separate concrete queries are separate segments of the input, the printing process takes linear time.

Theorem 11. *The decision algorithm for* **SPIL** *is sound and complete, and it works in expected linear time and quadratic time in the worst-case.*

References

1. Beklemishev, L., Gurevich, Y.: Propositional primal logic with disjunction. Journal of Logic and Computation 24(1), 257–282 (2014)
2. Beklemishev, L., Prokhorov, I.: On computationally efficient subsystems of propositional logic (to appear)
3. Blass, A., De Caso, G., Gurevich, Y.: An Introduction to DKAL. Microsoft Research technical report, MSR-TR-2012-108 (2012)
4. Cotrini, C., Gurevich, Y.: Basic primal infon logic. Journal of Logic and Computation (2013)
5. Cotrini, C., Gurevich, Y.: Transitive primal infon logic. The Review of Symbolic Logic 6, 281–304 (2013)
6. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. J. Autom. Reason. 31(1), 33–72 (2003)
7. Gurevich, Y., Neeman, I.: Logic of infons: The propositional case. ACM Trans. Comput. Logic 12(2), 9:1–9:28 (2011)

From Display Calculi to Deep Nested Sequent Calculi: Formalised for Full Intuitionistic Linear Logic

Jeremy E. Dawson[1], Ranald Clouston[2], Rajeev Goré[1], and Alwen Tiu[3]

[1] Research School of Computer Science, Australian National University, Australia
[2] Department of Computer Science, Aarhus University, Denmark
[3] School of Computer Engineering, Nanyang Technological University, Singapore

Abstract. Proof theory for a logic with categorical semantics can be developed by the following methodology: define a sound and complete display calculus for an extension of the logic with additional adjunctions; translate this calculus to a shallow inference nested sequent calculus; translate this calculus to a deep inference nested sequent calculus; then prove this final calculus is sound with respect to the original logic. This complex chain of translations between the different calculi require proofs that are technically intricate and involve a large number of cases, and hence are ideal candidates for formalisation. We present a formalisation of this methodology in the case of Full Intuitionistic Linear Logic (FILL), which is multiplicative intuitionistic linear logic extended with par.

1 Introduction

Belnap's Display Calculus [1] is a powerful modular approach to structural proof theory. Display calculi are often easy to design for a given logic [9] and enjoy a generic algorithm for cut-elimination. However they usually require the logic to be expanded with new structural connectives, raising the question of conservativity, and hence soundness, with respect to the original logic. They also do not enjoy a genuine subformula property and hence are ill-suited to backwards proof search. Various authors [4,15,10,11,14,5] have addressed these shortcomings by using some variation of *nested sequent calculus* with *deep inference* [13]. Such deep nested calculi employ a syntax similar to display calculi, but lack their ease of design and generic cut-elimination algorithm. Conversely, deep nested calculi can be designed to have a genuine subformula property, and a "separation property" that trivially yields conservativity results [10,11,5]. Since display calculi and deep nested calculi can have contrasting strengths, it is useful to provide sequent calculi in both styles for a given logic. The crux of such a development is the proof of equivalence between the display and deep nested calculi.

Proving the equivalence of display and deep nested calculi is technically intricate and can involve the verification of hundreds of cases. Such proofs proceed via an intermediate calculus, a *shallow* inference nested sequent calculus, and it is the proof of the equivalence of shallow and deep calculi that is the most

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 250–264, 2014.

demanding, requiring that every possible interaction of shallow and deep proof rules be covered. We hence have a fruitful proof theoretic methodology which cries out both for mechanised proof checking to increase confidence in its results, and for the use of automated tactics to reduce the drudgery of attaining them. We describe such a formalisation for Full Intuitionistic Linear Logic (FILL) [12], following our earlier work on display and deep nested calculi for this logic [5].

Schellinx [16] considered the standard multiple-conclusioned sequent calculus for intuitionistic logic (where the right-implication rule is restricted to prevent collapse to classical logic) without weakening and contraction, and showed that it does not enjoy cut-elimination. Hyland and de Paiva [12] gave this logic (with cut) the name Full Intuitionistic Linear Logic, and defined categorical semantics for it, giving several natural examples of categories exhibiting the required structure. They further claimed to have found a cut-free sequent calculus for FILL, in which term-assignments on formulae are put to novel use to block unsound applications of right-implication, via a freeness check on abstracted variables. Reasoning about freeness in the presence of binders is a well known source of subtle error, and a major topic of formalisation research (e.g. [19]). Indeed Bierman [2] found a counter-example to Hyland and de Paiva's cut-elimination proof exploiting a binding-related error, and presented two solutions using even more complex type-annotations, one due to Bellin. Braüner and de Paiva [3] subsequently suggested a cut-free calculus relying on annotations on sequents, rather than formulae. Two previous claims in the literature to annotation-free sequent calculi for FILL were erroneous, as discussed in [5].

Our recent contribution [5] to this rather vexed history was to show that annotations are not necessary; we gave a sound and complete display calculus for FILL and showed how it can be compiled into two equivalent nested sequent calculi, one with shallow inference and the other with deep inference. In particular the deep calculus is cut-free complete for FILL, enjoys the sub-formula property, and supports terminating backward proof-search, from which we obtained the NP-completeness of the validity problem for FILL. The derivation of these results, given in more detail in [6], is unavoidably highly technical, and given its difficulty and the history of FILL outlined above we sought to formalise our results in the proof assistant Isabelle/HOL. The completed formalisation presented in this paper finally establishes the correctness of a sequent calculus for this logic. In fact an initial attempt to prove the soundness of our calculus was found to be flawed only when we tried to formalise it (see below for more details), so this development has been an invaluable part of even our 'pen-and-paper' work.

We now outline our proof stategy [5] as formalised in this paper. FILL has the usual relation between multiplicative conjunction $\otimes$ and implication $\multimap$, where $\rightarrow$ denotes an arrow in a category (see §3):

$$(A \otimes B) \rightarrow C \text{ iff } (B \otimes A) \rightarrow C \text{ iff } A \rightarrow (B \multimap C) \tag{1}$$

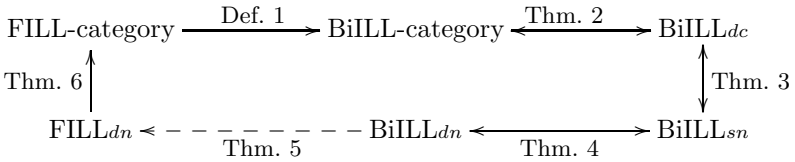
The *display property*, which underlies the generic cut-elimination algorithm of display calculi, requires the introduction of new structural connectives, so it is clarifying to regard a display calculus as defining a larger logic with new logical

connectives. In this case we have a multipicative *exclusion* $-\!\!<$, defined with respect to multiplicative disjunction (or *par*) $\parr$ in a manner dual to (1):

$$C \to (A \parr B) \text{ iff } C \to (B \parr A) \text{ iff } (C -\!\!< B) \to A \tag{2}$$

In §2 and §3, we extend the syntax of FILL with $-\!\!<$ and extend the semantics of FILL to obtain Bi-Intuitionistic Linear Logic (BiILL). In §4 we give a display calculus BiILL$_{dc}$ which is easily seen to be sound and complete for BiILL and hence complete for the sublogic FILL. The soundness of BiILL$_{dc}$ for FILL corresponds to the conservativity of BiILL over FILL. We first attempted to prove the soundness result directly via a rewriting strategy which removed occurrences of exclusion from a BiILL$_{dc}$-derivation of a FILL-formula to give an exclusion-free BiILL$_{dc}$-derivation of the same FILL-formula. This rewriting strategy turned out to be flawed, as it may not always terminate. Instead, we define two nested sequent calculi for BiILL: BiILL$_{sn}$ with shallow inference in §5, and BiILL$_{dn}$ with deep inference in §6. The equivalence of BiILL$_{sn}$ and BiILL$_{dn}$, established as Thm. 4 in §6, is the technical highlight of the formalisation, with 616 cases verified. Thm. 5 in §7 shows that because of a *separation* property, BiILL$_{dn}$ easily specialises to a deep nested calculus FILL$_{dn}$ with no trace of exclusion. Thm. 6 then shows that the calculus FILL$_{dn}$ is sound for FILL, thereby proving conservativity of BiILL over FILL. §8 concludes.

Our methodology is summarised below, where a solid arrow indicates that every valid formula in the source is also valid in the target, and a dashed arrow represents the same notion restricted to FILL formulae only:



We used Isabelle/HOL 2005 so that we could rework the cut-elimination proofs from our previous work on formalising cut elimination for display calculi [7]. As discussed in §5, one problem we found was the lack of support for nested datatypes involving multisets. The Isabelle/HOL theory files for our formalisation are at: `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/fill`

2 Formulae, Sequents and Derivations

We explain briefly the data structures we use to encode formulae, structures, sequents and derivations. The language of formulae for BiILL is defined using the grammar below where p denotes a propositional variable:

$$A ::= p \mid I \mid \bot \mid A \otimes A \mid A \parr A \mid A \multimap A \mid A -\!\!< A$$

A FILL formula is just a BiILL formula without any occurrences of $-\!\!<$.

BiILL formulae are defined formally in Isabelle as follows. We let the type variable `'s` be (Isabelle) strings.

```
datatype 's pformula =
    Btimes pformula pformula ("_ &&& _" [68,68] 67)
  | Bplus pformula  pformula ("_ +++ _" [66,66] 65)
  | Blolli pformula pformula ("_ --o _" [64,64] 63)
  | Bexcl pformula pformula ("_ --< _" [64,64] 63)
  | Btrue ("T") | Bfalse("F") | FV string | PP string
```

Here the binary constructors correspond to, respectively, $\otimes$, $\parr$, $\multimap$ and $\mathbin{-\!\!<}$, the unary constructors `Btrue` and `Bfalse` encode the units I and $\bot$ respectively, the constructor `PP` is used to encode propositional variables, and `FV` is used to encode "scheme variables"; we shall come back to these shortly.

We define the immediate (proper) subformula relation, `ipsubfml`.

```
ipsubfml     :: "('a pformula * 'a pformula) set"
inductive "ipsubfml" (* proper immediate subformula relation *)
  intrs ips_and1 "(P, P &&& Q) : ipsubfml" (etc)
```

A BiILL-sequent is a pair $X_a \vdash X_s$ of an *antecedent* and a *succedent* structure, defined respectively as follows:

$$X_a ::= A \mid \Phi \mid X_a, X_a \mid X_a < X_s \qquad\qquad X_s ::= A \mid \Phi \mid X_s, X_s \mid X_a > X_s$$

where Φ is a structural constant. A FILL-sequent is a BiILL-sequent containing no occurrence of $<$ or $\mathbin{-\!\!<}$. The relation between formulae and structures will be made precise in the next section.

Structures are represented by the datatype below:

```
datatype 's pstructr = Comma ('s pstructr) ('s pstructr)
  | Gt ('s pstructr) ('s pstructr) | Lt ('s pstructr) ('s pstructr)
  | Phi | Structform ('s pformula) | SV 's
```

where `Comma`, `Gt`, `Lt` and `Phi` correspond to the structural connectives ',', $>$, $<$ and Φ. The operator `Structform` casts a formula into a structure. The constructor `SV` represents scheme variables for structures. Since we must reason about arbitrary derivations, we have to allow derivations to contain structure variables and reason about their instantiations. We do not encode explicitly the notion of antecedent/succeedent structures in the data type; these notions are enforced via separate predicates when needed (see e.g. §5).

Sequents and rules of the calculus are represented by

```
datatype 'a sequent = Sequent 'a 'a
types 'a psc = "'a list * 'a" (* single step inference *)
types 'a rule = 'a sequent psc
```

The premises of a rule are represented using a list of sequents while the conclusion is a single sequent. Thus `(prems, concl)` means a rule with premises `prems` and conclusion `concl`. We write `$X |- $Y` to denote `(Sequent X Y)`.

We now briefly describe the functions we used to encode derivability. A fuller account is given in [8]. This framework is general in that a rule merely consists of "premises" and a "conclusion", and is independent of whether the things derived are formulae or sequents, but we will refer to them as formulae.

```
consts  derl, adm :: "'a psc set => 'a psc set"
        derrec    :: "'a psc set => 'a set => 'a set"
        dersl     :: "'a psc set => ('a list * 'a list) set"
        dersrec   :: "'a psc set => 'a set => 'a list set"
```

An inference rule `(ps, c) : 'a psc` is a list of premises `ps` and a conclusion `c`. Then, `derl rls` is the set of rules derivable from the rule set `rls`, and `derrec rls prems` is the set of formulae derivable using rules `rls` from the set `prems` of premises. These were defined as inductive sets, using auxiliary functions `dersl` and `dersrec`, which concern the derivability of all members of a list. So to say $(\texttt{ps}, \texttt{c}) \in \texttt{derl rls}$ reflects the shape of a derivation tree: `ps` is a list of exactly the premises used, in the correct order, whereas $\texttt{c} \in \texttt{derrec rls prems}$ holds even for any set of premises `prems` containing those required.

Since we use cut-admissibility for Display Calculi, and also some rules of BiILL$_{sn}$ are *admissible* (not *derivable*) in BiILL$_{dn}$ (see Theorem 4), we we need to formalise the notion that a rule of one system is *admissible* in another system: (ps, c) is admissible iff: if all premises in ps are derivable, then c is derivable:

$$(ps, c) \in \texttt{adm rls} \iff (\mathit{set}\ ps \subseteq \texttt{derrec rls} \quad \Rightarrow \quad c \in \texttt{derrec rls})$$

3 Formalising Categorial Semantics

Definition 1. *A* FILL-category *is a category equipped with*

- *a symmetric monoidal closed structure* $(\otimes, I, \multimap)$
- *a symmetric monoidal structure* $(\parr, \bot)$
- *a natural family of* weak distributivity *arrows* $A \otimes (B \parr C) \to (A \otimes B) \parr C$.

A BiILL-category *is a FILL-category where the* $\parr$ *bifunctor has a* co-closure $-\!\!<$, *so there is a natural isomorphism between arrows* $A \to B \parr C$ *and* $A -\!\!< B \to C$.

To interpret BiILL sequents in the category semantics, we use the following translation from (antecedent/succeedent) structures to formulae:

	A	Φ	X, Y	$X > Y$	$X < Y$
τ^a	A	I	$\tau^a(X) \otimes \tau^a(Y)$		$\tau^a(X) -\!\!< \tau^s(Y)$
τ^s	A	$\bot$	$\tau^s(X) \parr \tau^s(Y)$	$\tau^a(X) \multimap \tau^s(Y)$	

Definition 2. *A FILL- (resp. BiILL-) sequent* $X \vdash Y$ *is* satisfied *by a FILL- (resp. BiILL-) category if, given any valuation of its propositional variables as objects, there exists an arrow* $I \to \tau^a(X) \multimap \tau^s(Y)$. *It is FILL- (resp. BiILL-)* valid *if it is satisfied by all such categories.*

To establish validity it suffices to show a hom-set is non-empty inthe *free* FILL- (resp. BiILL-) categories. We sketch the definitions of these below, omitting the *equations* that hold between arrows in these categories.

Category: $A \xrightarrow{id} A \qquad A \xrightarrow{f' \circ f} A''$

Symmetric Monoidal: $A \heartsuit B \xrightarrow{f \heartsuit g} A' \heartsuit C \qquad (A \heartsuit B) \heartsuit C \underset{\alpha^{-1}}{\overset{\alpha}{\rightleftarrows}} A \heartsuit (B \heartsuit C)$

$$K \heartsuit A \underset{\lambda^{-1}}{\overset{\lambda}{\rightleftarrows}} A \qquad A \heartsuit K \underset{\rho^{-1}}{\overset{\rho}{\rightleftarrows}} A \qquad A \heartsuit B \xrightarrow{\gamma} B \heartsuit A$$

Closed:

$$A \multimap B \xrightarrow{A \multimap g} A \multimap C \qquad (A \multimap B) \otimes A \xrightarrow{\varepsilon} B \qquad A \xrightarrow{\eta} B \multimap A \otimes B$$

Weak Distributivity: $A \otimes (A' \parr A'') \xrightarrow{\omega} (A \otimes A') \parr A''$

Co-Closed:

$$A \mathrel{-\!\!<} B \xrightarrow{f \mathrel{-\!\!<} B} A' \mathrel{-\!\!<} B \qquad A \parr B \mathrel{-\!\!<} A \xrightarrow{\varepsilon} B \qquad A \xrightarrow{\eta} B \parr (A \mathrel{-\!\!<} B)$$

Fig. 1. Arrows of the free BiILL-category

Definition 3. *The* free *FILL- (resp. BiILL-) category has FILL- (resp. BiILL) formulae as objects, and, given objects A, A, A'', B, C and arrows $f : A \to A', f' : A' \to A'', g : B \to C$, and $(\heartsuit, K) \in \{(\otimes, I), (\parr, \bot)\}$, has arrows as in Fig. 1, where the co-closure arrows exist in the free BiILL-category only.*

Def. 3 can also be written as a deducibility relation between formulae where we use the turnstile $\vdash$ to assert the existence of an arrow, without specifying how it is constructed. For example, the rules below capture the definition of 'closed' above, in which the finer details of the construction $A \multimap g$ are elided:

$$\frac{B \vdash C}{A \multimap B \vdash A \multimap C} \qquad \frac{}{(A \multimap B) \otimes A \vdash B} \qquad \frac{}{A \vdash B \multimap (A \otimes B)}$$

We formalise this deducibility relation using the sequents of the previous section, but with only a formula (not a more complex structure) on each side. Then the identity arrow and composition of arrows become the usual identity and cut rules. As another example, the "closure" rules above are encoded as:

```
lolli_monoR == (["B" |- "C"], "A" --o "B" |- "A" --o "C")
lolliD == ([], ("A" --o "B") &&& "A" |- "B")
lolliI == ([], "A" |- "B" --o ("A" &&& "B"))
```

Our encoding is faithful because each arrow required by Def. 3 is encoded as one such rule giving the rules `biill_cat_rules` with subset `fill_cat_rules`.

4 Formalising Display Calculi

Our formalisation of the display calculus BiILL$_{dc}$ is very similar to that in [7], so we shall not give full details here. The display system BiILL$_{dc}$ is given in Fig. 2, where double-lined inference rules are invertible.

Cut and identity:

$$\text{(id)}\quad p \vdash p \qquad \text{(cut)}\ \frac{X \vdash A \qquad A \vdash Y}{X \vdash Y}$$

Logical rules:

$$(I \vdash)\ \frac{\Phi \vdash X}{I \vdash X} \qquad (\bot \vdash)\quad \bot \vdash \Phi \qquad (\vdash I)\quad \Phi \vdash I \qquad (\vdash \bot)\ \frac{X \vdash \Phi}{X \vdash \bot}$$

$$(\otimes \vdash)\ \frac{A, B \vdash X}{A \otimes B \vdash X} \qquad (\vdash \otimes)\ \frac{X \vdash A \qquad Y \vdash B}{X, Y \vdash A \otimes B}$$

$$(⅋ \vdash)\ \frac{A \vdash X \qquad B \vdash Y}{A ⅋ B \vdash X, Y} \qquad (\vdash ⅋)\ \frac{X \vdash A, B}{X \vdash A ⅋ B}$$

$$(\multimap\vdash)\ \frac{X \vdash A \qquad B \vdash Y}{A \multimap B \vdash X > Y} \qquad (\vdash\multimap)\ \frac{X \vdash A > B}{X \vdash A \multimap B}$$

$$(\prec \vdash)\ \frac{A < B \vdash X}{A \prec B \vdash X} \qquad (\vdash \prec)\ \frac{X \vdash A \qquad B \vdash Y}{X < Y \vdash A \prec B}$$

Structural rules:

$$\text{(rp)}\ \frac{X \vdash Y > Z}{X, Y \vdash Z} \quad \text{(rp)}\ \frac{X, Y \vdash Z}{Y \vdash X > Z} \quad \text{(drp)}\ \frac{X < Y \vdash Z}{X \vdash Y, Z} \quad \text{(drp)}\ \frac{X \vdash Y, Z}{X < Z \vdash Y}$$

$$(\Phi \vdash)\ \frac{X, \Phi \vdash Y}{X \vdash Y} \quad (\vdash \Phi)\ \frac{X \vdash \Phi, Y}{X \vdash Y} \quad \text{(Com} \vdash)\ \frac{X, Y \vdash Z}{Y, X \vdash Z} \quad (\vdash \text{Com})\ \frac{X \vdash Y, Z}{X \vdash Z, Y}$$

$$\text{(Ass} \vdash)\ \frac{W, (X, Y) \vdash Z}{(W, X), Y \vdash Z} \qquad (\vdash \text{Ass})\ \frac{W \vdash (X, Y), Z}{W \vdash X, (Y, Z)}$$

$$\text{(Grnb} \vdash)\ \frac{W, (X < Y) \vdash Z}{(W, X) < Y \vdash Z} \qquad (\vdash \text{Grnb})\ \frac{W \vdash (X > Y), Z}{W \vdash X > (Y, Z)}$$

Fig. 2. Display calculus BiILL$_{dc}$ for BiILL

As in our previous formalisation [7], structure variables like X are encoded as `$"X"` and formula variables like A are encoded as `"A"`. The quotes are necessary since we handle substitutions explicitly rather than via Isabelle variables [7]. For example, the cut rule is encoded as:

```
cutr == ([($"X" |- "A"), ("A" |- $"Y")], ($"X" |- $"Y"))
```

To prove the cut-admissibility result we largely reuse the code we used to prove cut-admissibility for the display calculus for relation algebras [7]. Belnap [1] gave eight conditions, C1-C8, which guarantee cut-elimination for a given display calculus. Previous work has shown that all except C8 are trivial or can be checked automatically [7]. The proof that a connective satisfies Belnap's C8 condition has to be coded in part individually for each connective, but, even so, we were able to reuse most of our previous code. Each display logic rule of BiILL$_{dc}$ is encoded as shown above for cut, giving the set of rules named `biilldc`, and its strict subset `biilldc_cf` which excludes the cut rule.

Given a set `S` of encoded rules, the set `rulefs S` is the (infinite) set of substitutional instances of members of `S`. From our previous work [8], the rule `(ps, c)` with list of premise (sequents) `ps` and conclusion (sequent) `c` is an admissible rule of `S` if the following holds, where colon is set-membership $\in$:

```
(?ps : dersrec ?rls {} --> ?c : derrec ?rls {}) ==> (?ps, ?c) : adm ?rls
```

Cut and identity: $\dfrac{}{p \Rightarrow p}\ id \qquad \dfrac{\mathcal{S} \Rightarrow \mathcal{U}, A \quad A, \mathcal{V} \Rightarrow \mathcal{T}}{\mathcal{S}, \mathcal{V} \Rightarrow \mathcal{U}, \mathcal{T}}\ cut$

Structural rules:

$$\frac{\mathcal{S} \Rightarrow \mathcal{T}, \mathcal{T}'}{(\mathcal{S} \Rightarrow \mathcal{T}) \Rightarrow \mathcal{T}'}\ drp_1 \quad \frac{\mathcal{S}, \mathcal{T} \Rightarrow \mathcal{T}'}{\mathcal{S} \Rightarrow (\mathcal{T} \Rightarrow \mathcal{T}')}\ rp_1 \quad \frac{(\mathcal{S} \Rightarrow \mathcal{S}'), \mathcal{T} \Rightarrow \mathcal{T}'}{(\mathcal{S}, \mathcal{T} \Rightarrow \mathcal{S}') \Rightarrow \mathcal{T}'}\ gl$$

$$\frac{(\mathcal{S} \Rightarrow \mathcal{T}) \Rightarrow \mathcal{T}'}{\mathcal{S} \Rightarrow \mathcal{T}, \mathcal{T}'}\ drp_2 \quad \frac{\mathcal{S} \Rightarrow (\mathcal{T} \Rightarrow \mathcal{T}')}{\mathcal{S}, \mathcal{T} \Rightarrow \mathcal{T}'}\ rp_2 \quad \frac{\mathcal{S} \Rightarrow (\mathcal{S}' \Rightarrow \mathcal{T}'), \mathcal{T}}{\mathcal{S} \Rightarrow (\mathcal{S}' \Rightarrow \mathcal{T}', \mathcal{T})}\ gr$$

Logical rules: $\dfrac{}{\bot \Rightarrow \cdot}\ \bot_l \qquad \dfrac{\mathcal{S} \Rightarrow \mathcal{T}}{\mathcal{S} \Rightarrow \mathcal{T}, \bot}\ \bot_r \qquad \dfrac{\mathcal{S} \Rightarrow \mathcal{T}}{\mathcal{S}, \mathrm{I} \Rightarrow \mathcal{T}}\ \mathrm{I}_l \qquad \dfrac{}{\cdot \Rightarrow \mathrm{I}}\ \mathrm{I}_r$

$$\frac{\mathcal{S}, A, B \Rightarrow \mathcal{T}}{\mathcal{S}, A \otimes B \Rightarrow \mathcal{T}}\ \otimes_l \qquad \frac{\mathcal{S} \Rightarrow A, \mathcal{T} \quad \mathcal{S}' \Rightarrow B, \mathcal{T}'}{\mathcal{S}, \mathcal{S}' \Rightarrow A \otimes B, \mathcal{T}, \mathcal{T}'}\ \otimes_r$$

$$\frac{\mathcal{S}, A \Rightarrow \mathcal{T} \quad \mathcal{S}', B \Rightarrow \mathcal{T}'}{\mathcal{S}, \mathcal{S}', A \parr B \Rightarrow \mathcal{T}, \mathcal{T}'}\ \parr_l \qquad \frac{\mathcal{S} \Rightarrow A, B, \mathcal{T}}{\mathcal{S} \Rightarrow A \parr B, \mathcal{T}}\ \parr_r$$

$$\frac{\mathcal{S} \Rightarrow A, \mathcal{T} \quad \mathcal{S}', B \Rightarrow \mathcal{T}'}{\mathcal{S}, \mathcal{S}', A \multimap B \Rightarrow \mathcal{T}, \mathcal{T}'}\ \multimap_l \qquad \frac{\mathcal{S} \Rightarrow \mathcal{T}, (A \Rightarrow B)}{\mathcal{S} \Rightarrow \mathcal{T}, A \multimap B}\ \multimap_r$$

$$\frac{\mathcal{S}, (A \Rightarrow B) \Rightarrow \mathcal{T}}{\mathcal{S}, A \prec B \Rightarrow \mathcal{T}}\ \prec_l \qquad \frac{\mathcal{S} \Rightarrow A, \mathcal{T} \quad \mathcal{S}', B \Rightarrow \mathcal{T}'}{\mathcal{S}, \mathcal{S}' \Rightarrow A \prec B, \mathcal{T}, \mathcal{T}'}\ \prec_r$$

Fig. 3. The shallow inference system BiILL$_{sn}$

We then proved cut-admissibility as below, where colon now simply states the name of the theorem. The formal proof reuses the work described in [7].

Theorem 1 (Cut-Admissibility). *From cut-free* BiILL$_{dc}$*-derivations of* $X \vdash A$ *and* $A \vdash Y$ *we can obtain a cut-free* BiILL$_{dc}$*-derivation of* $X \vdash Y$*.*

```
dc_cut_adm : "rulefs {cutr} <= adm (rulefs biilldc_cf)"
```

We can now gain our first result linking proof theory and semantics. In the following, `(?A |- ?B)` is a sequent with arbitrary formulae on each side. Arbitrary structures would appear as `($?A |- $?B)` [7].

Theorem 2. BiILL$_{dc}$ *is* sound *and* cut-free complete *for BiILL-validity (where the appellation* `cf` *captures cut-free).*

```
dc_cat_equiv_cf : "((?A |- ?B) : derrec (rulefs biilldc_cf) {}) =
  ((?A |- ?B) : derrec (rulefs biill_cat_rules) {})"
```

5 Shallow Nested Sequent Calculi

In [5] nested sequents are defined as below, where A_i and B_j are formulae:

$$S \quad T ::= S_1, \ldots, S_k, A_1, \ldots, A_m \Rightarrow B_1, \ldots, B_n, T_1, \ldots, T_l$$

We use Γ and Δ for multisets of formulae and use P, Q, S, T, X, Y, etc., for nested sequents, and $\mathcal{S}$, $\mathcal{X}$, etc., for multisets of nested sequents and formulae. The empty multiset is $\cdot$ ('dot'). A nested sequent is essentially a display structure, but with the associativity and commutativity of the comma structural connective implicit in the use of multisets. The sequent arrow $\Rightarrow$ overloads both $>$ and $<$, depending on whether it occurs in an antecedent or a succeedent position in the sequent. This overloading simplifies the presentation of the nested sequent rules [5]. The shallow inference system BiILL$_{sn}$ for BiILL is given in Fig. 3.

The most faithful encoding of a nested sequent would be one that uses multisets as a datatype, which is supported by recent versions of Isabelle [18]. However due to incompatibilities between versions of Isabelle we have been constrained to use an older version of Isabelle to allow us to reuse proofs for display calculi developed in that version [7]. Our definition of nested sequents is thus as below:

```
datatype nested = NComma nested nested | Nseq nested nested
                | NPhi | NStructform formula | NSV string
```

In our definition, `NSeq` is the nested sequent turnstile $\Rightarrow$, `NComma` is the comma of nested sequent calculi and `NPhi` is its unit. As for display calculi, we allow `=>` instead of `Nseq` and `,,,` instead of `NComma`. In our Isabelle formalisation, BiILL$_{sn}$ rules are prefixed by `sn`, e.g., the *rp* rule is named `sn_rp`. The entire set of rules is called `biillsn` and its cut-free subset is `biillsn_cf` (see file `N_Rls.thy`). As with the display calculus, we define a function `nrulefs` to generate the (infinite) set of all substitution instances of a given rule.

We defined a relation `ms_deep_equiv` of *multiset-equivalence*, under which any two Isabelle nested sequents are equivalent if they would be the same if a collection of Isabelle nested sequents, separated by commas, were considered as a multiset. This includes where the difference between two Isabelle nested sequents occurs at any depth. Its definition relies on a function `ms_of_ns` for "multiset of Isabelle nested sequent" which turns (eg) the Isabelle nested sequent $(S,T),U$ into the multiset `{# S, T, U #}`, and a relation `ms_ms_deep_equiv`, which expresses equivalence of multisets of Isabelle nested sequents.

To prove BiILL$_{dc}$ and BiILL$_{sn}$ equivalent, we define translation functions

```
consts nested_to_str :: "bool => nested => structr"
       nested_to_seq :: "nested => structr sequent"
       seq_to_nested :: "structr sequent => nested"
       str_to_nested :: "structr => nested"
```

where the translation from a nested sequent to a display calculus structure depends on whether it is in an antecedent or succedent position.

The first two of these functions convert a nested sequent to a display calculus sequent or structure by converting '$\Rightarrow$' to '$\vdash$' (for `nested_to_seq`), to '$>$' (for `nested_to_str True`), or to '$<$' (for `nested_to_str False`), and converting comma to comma. The latter two convert '$\vdash$', '$>$', and '$<$' to '$\Rightarrow$'.

For example `nested_to_seq` takes $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$ to $A < B \vdash C > D$, and `seq_to_nested` does the reverse.

Considering the set of display calculus sequents with $<$ and $>$ only in antecedent and succedent positions respectively (expressed as `seq_LtGtOK`), and the

set of nested sequents, then we have mutually inverse bijections `nested_to_seq` and `seq_to_nested` between these sets. We can express this by:

```
nest_seq_equiv : "(seq_LtGtOK ?seq & seq_to_nested ?seq = ?nes) =
      ((EX a s. ?nes = ($a => $s)) & nested_to_seq ?nes = ?seq)"
```

The proof systems BiILL$_{dc}$ and BiILL$_{sn}$ are very similar; their structural rules are the same (modulo some notational variance). The only difference is that BiILL$_{dc}$ requires that logical rules be applied only to a 'displayed' formula, i.e. the principal formula must appear in isolation either on the left or on the right of the turnstile. Their equivalence is not surprising, so we state their equivalence here and refer the reader to the proof scripts for details.

Theorem 3. *The display sequent $A \vdash B$ is cut-free* BiILL$_{dc}$*-derivable iff the nested sequent $A \Rightarrow B$ is cut-free* BiILL$_{sn}$*-derivable, and* $\mathrm{I} \vdash A$ *is cut-free* BiILL$_{dc}$*-derivable iff the nested sequent $\cdot \Rightarrow A$ is cut-free* BiILL$_{sn}$*-derivable.*

```
dc_sn_equiv_alt = "((?A |- ?B) : derrec (rulefs biilldc_cf) {}) =
         ((?A => ?B) : derrec (nrulefs biillsn_cf) {})"
dc_sn_equiv : "((T |- ?A) : derrec (rulefs biilldc_cf) {}) =
         (($NPhi => ?A) : derrec (nrulefs biillsn_cf) {})"
```

6 Deep Nested Sequent Calculi

Deep inference rules for nested sequents are applied in a *context*, i.e., a nested sequent with a hole [ ]. We use several notions of contexts in our formalisation. The first two accept a set of nested sequent rules and return a set of nested sequent rules while the third accepts and returns a set of Isabelle nested sequents

```
ctxt :: "nested psc set => nested psc set"
dctxt :: "nested psc set => nested psc set"
hctxt :: "nested set => nested set"
```

For example, if $([P], C) \in R$, where $[P]$ is a singleton list (rather than a context) containing one premise, then $([X[P]], X[C]) \in$ `ctxt` R is also a single premise rule. Likewise, if $C \in R$ and $X[\,]$ is a hollow context then $X[C] \in$ `hctxt` R.

Some of the proofs involving `ctxt` were easier using a related definition `dctxt` where $X[S \Rightarrow T]$ means adding nested sequents to S or to T, rather than to $S \Rightarrow T$. For example, if $([P_1 \Rightarrow P_2], (C_1 \Rightarrow C_2)) \in$ `dctxt` R, then $([P_1, X \Rightarrow P_2], (C_1, X \Rightarrow C_2)) \in$ `dctxt` R. Similarly, if $([P_1 \Rightarrow P_2], (C_1 \Rightarrow C_2)) \in$ `dctxt` R then $([(P_1 \Rightarrow P_2) \Rightarrow X], ((C_1 \Rightarrow C_2) \Rightarrow X)) \in$ `dctxt` R

The nested sequent system BiILL$_{dn}$ is given in Fig. 4. Notice that it lacks the structural rules. The zero-premise rules require that certain sequents or contexts are *hollow*, i.e., contain no occurrences of formulae. The branching rules require operations to merge contexts and nested sequents, which are explained below.

The *merge set* $X_1 \bullet X_2$ of two sequents X_1 and X_2 is defined as:

$$
\begin{aligned}
X_1 \bullet X_2 = \{\ & (\Gamma_1, \Gamma_2, Y_1, \ldots, Y_m \Rightarrow \Delta_1, \Delta_2, Z_1, \ldots, Z_n) \ \mid \\
& X_1 = (\Gamma_1, P_1, \ldots, P_m \Rightarrow \Delta_1, Q_1, \ldots, Q_n) \ \text{ and} \\
& X_2 = (\Gamma_2, S_1, \ldots, S_m \Rightarrow \Delta_2, T_1, \ldots, T_n) \ \text{ and} \\
& Y_i \in P_i \bullet S_i \text{ for } 1 \leq i \leq m \text{ and } Z_j \in Q_j \bullet T_j \text{ for } 1 \leq j \leq n \ \}
\end{aligned}
$$

Propagation rules:

$$\frac{X[\mathcal{S} \Rightarrow (A, \mathcal{S}' \Rightarrow \mathcal{T}'), \mathcal{T}]}{X[\mathcal{S}, A \Rightarrow (\mathcal{S}' \Rightarrow \mathcal{T}'), \mathcal{T}]}\ pl_1 \qquad \frac{X[(\mathcal{S} \Rightarrow \mathcal{T}, A), \mathcal{S}' \Rightarrow \mathcal{T}']}{X[(\mathcal{S} \Rightarrow \mathcal{T}), \mathcal{S}' \Rightarrow A, \mathcal{T}']}\ pr_1$$

$$\frac{X[\mathcal{S}, A, (\mathcal{S}' \Rightarrow \mathcal{T}') \Rightarrow \mathcal{T}]}{X[\mathcal{S}, (\mathcal{S}', A \Rightarrow \mathcal{T}') \Rightarrow \mathcal{T}]}\ pl_2 \qquad \frac{X[\mathcal{S} \Rightarrow \mathcal{T}, A, (\mathcal{S}' \Rightarrow \mathcal{T}')]}{X[\mathcal{S} \Rightarrow \mathcal{T}, (\mathcal{S}' \Rightarrow \mathcal{T}', A)]}\ pr_2$$

Identity and logical rules: In branching rules, $X[\ ] \in X_1[\ ] \bullet X_2[\ ]$, $\mathcal{S} \in \mathcal{S}_1 \bullet \mathcal{S}_2$ and $\mathcal{T} \in \mathcal{T}_1 \bullet \mathcal{T}_2$.

$$\frac{X[\ ], \mathcal{U} \text{ and } \mathcal{V} \text{ are hollow.}}{X[\mathcal{U}, p \Rightarrow p, \mathcal{V}]}\ id^d \qquad \frac{X[\ ], \mathcal{U} \text{ and } \mathcal{V} \text{ are hollow.}}{X[\bot, \mathcal{U} \Rightarrow \mathcal{V}]}\ \bot_l^d \qquad \frac{X[\mathcal{S} \Rightarrow \mathcal{T}]}{X[\mathcal{S} \Rightarrow \mathcal{T}, \bot]}\ \bot_r^d$$

$$\frac{X[\mathcal{S} \Rightarrow \mathcal{T}]}{X[\mathcal{S}, \mathrm{I} \Rightarrow \mathcal{T}]}\ \mathrm{I}_l^d \qquad \frac{X[\ ], \mathcal{U} \text{ and } \mathcal{V} \text{ are hollow.}}{X[\mathcal{U} \Rightarrow \mathrm{I}, \mathcal{V}]}\ \mathrm{I}_r^d$$

$$\frac{X[\mathcal{S}, A, B \Rightarrow \mathcal{T}]}{X[\mathcal{S}, A \otimes B \Rightarrow \mathcal{T}]}\ \otimes_l^d \qquad \frac{X_1[\mathcal{S}_1 \Rightarrow A, \mathcal{T}_1] \quad X_2[\mathcal{S}_2 \Rightarrow B, \mathcal{T}_2]}{X[\mathcal{S} \Rightarrow A \otimes B, \mathcal{T}]}\ \otimes_r^d$$

$$\frac{X_1[\mathcal{S}_1 \Rightarrow A, \mathcal{T}_1] \quad X_2[\mathcal{S}_2, B \Rightarrow \mathcal{T}_2]}{X[\mathcal{S}, A \multimap B \Rightarrow \mathcal{T}]}\ \multimap_l^d \qquad \frac{X[\mathcal{S} \Rightarrow \mathcal{T}, (A \Rightarrow B)]}{X[\mathcal{S} \Rightarrow \mathcal{T}, A \multimap B]}\ \multimap_r^d$$

$$\frac{X_1[\mathcal{S}_1, A \Rightarrow \mathcal{T}_1] \quad X_2[\mathcal{S}_2, B \Rightarrow \mathcal{T}_2]}{X[\mathcal{S}, A \parr B \Rightarrow \mathcal{T}]}\ \parr_l^d \qquad \frac{X[\mathcal{S} \Rightarrow A, B, \mathcal{T}]}{X[\mathcal{S} \Rightarrow A \parr B, \mathcal{T}]}\ \parr_r^d$$

$$\frac{X[\mathcal{S}, (A \Rightarrow B) \Rightarrow \mathcal{T}]}{X[\mathcal{S}, A \prec B \Rightarrow \mathcal{T}]}\ \prec_l^d \qquad \frac{X_1[\mathcal{S}_1 \Rightarrow A, \mathcal{T}_1] \quad X_2[\mathcal{S}_2, B \Rightarrow \mathcal{T}_2]}{X[\mathcal{S} \Rightarrow A \prec B, \mathcal{T}]}\ \prec_r^d$$

Fig. 4. The deep inference system BiILL$_{dn}$

The merge set $X_1[\ ] \bullet X_2[\ ]$ of two contexts $X_1[\ ]$ and $X_2[\ ]$ is defined in Figure 5. If $X[\ ] = X_1[\ ] \bullet X_2[\ ]$ we say $X_1[\ ]$ and $X_2[\ ]$ are a *partition* of $X[\ ]$. We extend the notion of a merge set between multisets of formulae and sequents as follows. Given $\mathcal{X} = \Gamma \cup \{X_1, \ldots, X_n\}$ and $\mathcal{Y} = \Delta \cup \{Y_1, \ldots, Y_n\}$ their merge set contains all multisets of the form: $\Gamma \cup \Delta \cup \{Z_1, \ldots, Z_n\}$ where $Z_i \in X_i \bullet Y_i$.

In Isabelle we defined merged sequents using triples, so in effect $(X_1, X_2, X) \in$ `rmerge` means X is a sequent in $X_1 \bullet X_2$. This is easier to define than in the paper where multisets are used, because we require simply that each structural atom (formula or structure variable) in X is replaced by Φ in exactly one of X_1 or X_2. That is, each atom has to go in one partition or the other. Likewise, to express the idea of $X_1[\ ] \bullet X_2[\ ]$ we define $(X_1, X_2, X) \in$ `rmerge1`(Y_1, Y_2, Y) to be similar except that at one spot X contains Y, where X_i contains Y_i.

We illustrate here some key steps in the formalisation of the equivalence between BiILL$_{sn}$ and BiILL$_{dn}$. We show only the translation from shallow nested sequent proofs to deep nested sequent proofs, which is the more difficult part of the equivalence. The key lemma here is that the rules rp_i, drp_i, gl and gr, which are in BiILL$_{sn}$ but not in the deep calculus BiILL$_{dn}$, are admissible in BiILL$_{dn}$.

$$
\begin{array}{ll}
\text{If } X_1[\,] = [\,] \text{ and } X_2[\,] = [\,] \text{ then } X_1[\,] \bullet X_2[\,] = \{[\,]\} \\
\text{If } X_1[\,] & = (\Gamma_1, Y_1[\,], P_1, \ldots, P_m \Rightarrow \Delta_1, Q_1, \ldots, Q_n) \text{ and} \\
\quad X_2[\,] & = (\Gamma_2, Y_2[\,], S_1, \ldots, S_m \Rightarrow \Delta_2, T_1, \ldots, T_n) \text{ then} \\
X_1[\,] \bullet X_2[\,] & = \{\ (\Gamma_1, \Gamma_2, Y[\,], U_1, \ldots, U_m \Rightarrow \Delta_1, \Delta_2, V_1, \ldots, V_n) \quad | \\
& \quad\ Y[\,] \in Y_1[\,] \bullet Y_2[\,] \text{ and } U_i \in P_i \bullet S_i \text{ for } 1 \leq i \leq m \text{ and} \\
& \quad\ V_j \in Q_j \bullet T_j \text{ for } 1 \leq j \leq n\ \} \\
\text{If } X_1[\,] & = (\Gamma_1, P_1, \ldots, P_m \Rightarrow \Delta_1, Y_1[\,], Q_1, \ldots, Q_n) \text{ and} \\
\quad X_2[\,] & = (\Gamma_2, S_1, \ldots, S_m \Rightarrow \Delta_2, Y_2[\,], T_1, \ldots, T_n) \text{ then} \\
X_1[\,] \bullet X_2[\,] & = \{\ (\Gamma_1, \Gamma_2, U_1, \ldots, U_m \Rightarrow \Delta_1, \Delta_2, Y[\,], V_1, \ldots, V_n) \mid \\
& \quad\ Y[\,] \in Y_1[\,] \bullet Y_2[\,] \text{ and } U_i \in P_i \bullet S_i \text{ for } 1 \leq i \leq m \text{ and} \\
& \quad\ V_j \in Q_j \bullet T_j \text{ for } 1 \leq j \leq n\ \}
\end{array}
$$

Fig. 5. Merging of contexts

Lemma 1. *The rules drp_1, rp_1, drp_2, rp_2, gl, and gr permute up over all logical rules of* BiILL$_{dn}$.

This is the permutation lemma, that (in general) where one of the shallow rules in question follows a rule of BiILL$_{dn}$ in a derivation, then the derivation can be re-ordered so that the shallow rule precedes the deep rule. We illustrate here a step in the proof of this permutation lemma, i.e., when permuting structural rules over a single-premise logical rule. We proved theorems of the following form

```
p_irp_anda : "?c = ($?ca => $?cs) -->
  ([?p], ?c) : ctxt (nrulefs {sn_anda}) -->
  (?c, ?c') : ms_deep_equiv --> ([?c'], ?d') : nrulefs {invert sn_rp} -->
  (EX p' q q' d. (?p, p') : ms_deep_equiv &
    ([p'], q) : nrulefs {invert sn_rp} & (q, q') : ms_deep_equiv &
    ([q'], d) : dctxt (nrulefs {sn_anda}) & (d, ?d') : ms_deep_equiv)"
```

That is, where a structural rule, e.g., rp_1, appears below a deep logical rule in a derivation, the derivation steps may be permuted so that the logical rule follows the other rule. It may be noted that this result uses `dctxt`, not `ctxt`, in the conclusion. This made semi-automatic proof easier; the lemmas for all the logical rules concerned (6 of them) for all the structural rules involved (6 of them) were done using just three separate sets of tactics.

The proofs of the permutation lemma involved a large number of cases, because a sequent expression such as $X[\mathcal{S} \Rightarrow \mathcal{T}]$ can match a given sequent Z in numerous ways, for two reasons:

- for multisets $\mathcal{S}$ and $\mathcal{T}$, there can be multiple ways to achieve $(\mathcal{S}, \ldots) \in$ `ms_deep_equiv`
- the size of context $X[\,]$ is arbitrary, so $\mathcal{S} \Rightarrow \mathcal{T}$ can match any part of Z.

The attempted proof encounters many obviously impossible cases such as a formula matching $\mathcal{S} \Rightarrow \mathcal{T}$. After these are eliminated, we counted the cases where a goal (such as the conclusion of `p_rp_anda`) is actually solved. These cases numbered 616, which shows the value of automating the process as much as possible.

Theorem 4. *All rules cut-free derivable in* BiILL$_{sn}$ *are (cut-free) admissible in* BiILL$_{dn}$.

```
sn_dn_der : "derl (nrulefs biillsn_cf) <= adm biilldn"
```

A sequent is cut-free provable (derivable from the empty set of assumption sequents) in BiILL$_{sn}$ *iff it is provable* BiILL$_{dn}$.

```
sn_dn_equiv : "(?r : derrec (nrulefs biillsn_cf) {}) =
               (?r : derrec biilldn {})"
```

The formula A is cut-free provable in the display calculus iff it is (cut-free) provable in BiILL$_{dn}$.

```
dc_dn_equiv : "((T |- ?A) : derrec (rulefs biilldc_cf) {}) =
               (($NPhi => ?A) : derrec biilldn {})"
```

7 Soundness of the Deep Nested Calculus FILL$_{dn}$

Definition 4. *A nested sequent is a nested* FILL-sequent *if it has no nesting of sequents on the left of* $\Rightarrow$*, and no occurrences of* $-\!\!<$.

BiILL$_{dn}$ enjoys the *separation* property that rule applications with FILL-sequents as their conclusions may only have FILL-sequents as their premises; note that the display calculus BiILL$_{dc}$ obviously lacks this property, given (drp). We hence define FILL$_{dn}$ as the proof system obtained from BiILL$_{dn}$ by restricting to FILL-sequents and removing the unnecessary rules pr_1, pl_2, $-\!\!<^d_l$ and $-\!\!<^d_r$. Our goal here is to show that FILL$_{dn}$ is sound with respect to FILL categories.

The formula translation of τ^s (see §3) can be adapted straightforwardly to map (nested) FILL-sequents to FILL-formulae. Such a sequent S is FILL-valid if there is an arrow $I \to \tau^s(S)$ in the free FILL-category.

In our formalisation, we in fact defined the rules `filldn` of FILL$_{dn}$ without requiring the sequents involved to be FILL-sequents. We then defined a corresponding set `sfilldn` of rules, requiring that the sequents are FILL-sequents.

Theorem 5. *A* BiILL$_{dn}$*-derivation of a* FILL$_{dn}$*-sequent is a* FILL$_{dn}$*-derivation.*

```
dn_der_biill_sfill : "[| ?c : derrec biilldn ?ps;
    ALL U. (U, ntau True ?c) : ipsubfml^* --> U ~: excl_fmls |] ==>
    ?c : derrec sfilldn ?ps"
```

The soundness proof consists of a series of lemmas showing that the rules of FILL$_{dn}$ preserves validity going downward (from premises to conclusion). We illustrate one particularly challenging lemma that involves context merging.

Lemma 2. *Take* $X[\,] \in X_1[\,] \bullet X_2[\,]$ *and* $\mathcal{T} \in \mathcal{T}_1 \bullet \mathcal{T}_2$*. Then the following arrows exist in the free FILL-category for all* A, B, Γ_1 *and* Γ_2*:*

(a) $\tau^s(X_1[\Gamma_1 \Rightarrow A, \mathcal{T}_1]) \otimes \tau^s(X_2[\Gamma_2 \Rightarrow B, \mathcal{T}_2]) \;\to\; \tau^s(X[\Gamma_1, \Gamma_2 \Rightarrow A \otimes B, \mathcal{T}])$;
(b) $\tau^s(X_1[\Gamma_1 \Rightarrow A, \mathcal{T}_1]) \otimes \tau^s(X_2[\Gamma_2, B \Rightarrow \mathcal{T}_2]) \;\to\; \tau^s(X[\Gamma_1, \Gamma_2, A \multimap B \Rightarrow \mathcal{T}])$;
(c) $\tau^s(X_1[\Gamma_1, A \Rightarrow \mathcal{T}_1]) \otimes \tau^s(X_2[\Gamma_2, B \Rightarrow \mathcal{T}_2]) \;\to\; \tau^s(X[\Gamma_1, \Gamma_2, A \parr B \Rightarrow \mathcal{T}])$;

This lemma corresponds to soundness of branching logical rules of FILL$_{dn}$. We proved the inductive part (involving the contexts X_i and X) once, in the form of `fill_rmerge1_der_nseq`. Then we proved the base case (without X_i and X) for each connective, resulting in three theorems `lem27s`, of which one is shown.

```
fill_rmerge1_der_nseq : "[| (?Ta, ?Tb, ?Tc) : rmerge1 (?A, ?B, ?C);
  ?prems = {ntau True ?A &&& ntau True ?B |- ntau True ?C};
  (?W, ntau False ?C) : ipsubfml^* & ?W : excl_fmls;
  ALL U. (U, ntau True ?Tc) : ipsubfml^* --> U ~: excl_fmls |] ==>
  (ntau True ?Ta &&& ntau True ?Tb |- ntau True ?Tc) :
    derrec (rulefs fill_cat_rules) ?prems"
lem27s (first one) :
  "[| (?A, ?B, ?C) : dn_ands; (?Ta, ?Tb, ?Tc) : rmerge1 (?A, ?B, ?C);
    ALL U. (U, ntau True ?Tc) : ipsubfml^* --> U ~: excl_fmls |] ==>
  (ntau True ?Ta &&& ntau True ?Tb |- ntau True ?Tc) :
    derrec (rulefs fill_cat_rules) {}"
```

Theorem 6. *For every rule of* FILL$_{dn}$*, if the premises are FILL-valid then so is the conclusion.*

```
filldn_rules_valid : "[| (?ps, ?c) : filldn;
  ALL U. (U, ntau True ?c) : ipsubfml^* --> U ~: excl_fmls |] ==>
  (T |- ntau True ?c) : derrec (rulefs fill_cat_rules)
    ((%p. T |- ntau True p) ` set ?ps)"
```

Theorem 7. *A formula is* FILL*-valid iff it is* FILL$_{dn}$*-provable, and BiILL is conservative over FILL.*

8 Conclusion and Future Work

Finding a cut-free sequent calculus for FILL has been a notoriously difficult problem, as we have reviewed in our introduction, and involved candidate proof systems that turned out to be incomplete. Our formalisation finally establishes convincingly that our deep nested calculus FILL$_{dn}$ is both sound and complete for FILL. Apart from our FILL$_{dn}$, all other existing proof calculi for FILL still require complex annotations to ensure cut-elimination.

The formalisation and verification described here was a significant task: it was the major activity for an experienced Isabelle user (Dawson) for about seven months, not including some months more working on the proof which ultimately was found to be flawed (see §1). and not counting the proof of Thm 1, reused from [7]. The most difficult single part of it was the proof of Lemma 1, discussed in §6. The difficulty in defining a nested sequent datatype containing multisets of nested sequents (see §5) was also significant. The value of the formal verification is clear since it led us to find the flaw in the previous attempt at a proof.

Taking a broader perspective, we have shown a detailed formalisation of a methodology for deriving a deep nested sequent calculus for a logic from its categorical semantics via a display calculus and a shallow nested sequent calculus for a natural extension containing additional connectives. For future work, we plan to apply this same formalised methodology to derive deep nested sequent calculi for a wide range of logics [4,15,10,11,17,14]. A difference between these

logics and FILL is the use of "additive" context splitting in branching rules, where contexts are duplicated across premises. In the presence of contraction rules, our multiplicative context splitting can simulate such additive splitting, just as in the traditional sequent calculus. That is, one can apply contraction to duplicate every formula occurrence in the context before splitting them. Thus we think a similar formalisation effort for, say, nested sequent calculi for modal logics [11], would benefit significantly from our current formalisation.

References

1. Belnap, N.D.: Display logic. J. Philos. Logic 11, 375–417 (1982)
2. Bierman, G.M.: A note on full intuitionistic linear logic. Ann. Pure Appl. Logic 79(3), 281–287 (1996)
3. Bräuner, T., de Paiva, V.: A formulation of linear logic based on dependency-relations. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 129–148. Springer, Heidelberg (1998)
4. Brünnler, K.: Deep sequent systems for modal logic. Arch. Math. Logic 48(6), 551–577 (2009)
5. Clouston, R., Dawson, J.E., Goré, R., Tiu, A.: Annotation-free sequent calculi for full intuitionistic linear logic. In: CSL, pp. 197–214 (2013)
6. Clouston, R., Dawson, J.E., Goré, R., Tiu, A.: Annotation-free sequent calculi for full intuitionistic linear logic - extended version. CoRR, abs/1307.0289 (2013)
7. Dawson, J.E., Goré, R.P.: Formalised cut admissibility for display logic. In: Carreño, V.A., Muñoz, C.A., Tahar, S. (eds.) TPHOLs 2002. LNCS, vol. 2410, pp. 131–147. Springer, Heidelberg (2002)
8. Dawson, J.E., Goré, R.: Generic methods for formalising sequent calculi applied to provability logic. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 263–277. Springer, Heidelberg (2010)
9. Goré, R.: Substructural logics on display. Log. J. IGPL 6(3), 451–504 (1998)
10. Goré, R., Postniece, L., Tiu, A.: Cut-elimination and proof search for bi-intuitionistic tense logic. In: AiML, pp. 156–177 (2010)
11. Goré, R., Postniece, L., Tiu, A.: On the correspondence between display postulates and deep inference in nested sequent calculi for tense logics. LMCS 7(2) (2011)
12. Hyland, M., de Paiva, V.: Full intuitionistic linear logic (extended abstract). Ann. Pure Appl. Logic 64(3), 273–291 (1993)
13. Kashima, R.: Cut-free sequent calculi for some tense logics. Studia Log. 53, 119–135 (1994)
14. Park, J., Seo, J., Park, S.: A theorem prover for boolean BI. In: POPL 2013, pp. 219–232 (2013)
15. Poggiolesi, F.: The method of tree-hypersequents for modal propositional logic. In: Trends in Logic IV, pp. 31–51 (2009)
16. Schellinx, H.: Some syntactical observations on linear logic. J. Logic Comput. 1(4), 537–559 (1991)
17. Straßburger, L.: Cut elimination in nested sequents for intuitionistic modal logics. In: Pfenning, F. (ed.) FOSSACS 2013 (ETAPS 2013). LNCS, vol. 7794, pp. 209–224. Springer, Heidelberg (2013)
18. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In: LICS, pp. 596–605 (2012)
19. Urban, C.: Nominal techniques in isabelle/HOL. J. Automat. Reason. 40(4), 327–356 (2008)

Towards a Coalgebraic Chomsky Hierarchy
(Extended Abstract)*

Sergey Goncharov[1], Stefan Milius[1], and Alexandra Silva[2]

[1] Lehrstuhl für Theoretische Informatik, Friedrich-Alexander-Universität Erlangen-Nürnberg
[2] Radboud University Nijmegen and Centrum Wiskunde & Informatica, Amsterdam

Abstract. The Chomsky hierarchy plays a prominent role in the foundations of theoretical computer science relating classes of formal languages of primary importance. In this paper we use recent developments on coalgebraic and monad-based semantics to obtain a generic notion of a $\mathbb{T}$-*automaton*, where $\mathbb{T}$ is a monad, which allows the uniform study of various notions of machines (e.g. finite state machines, multi-stack machines, Turing machines, weighted automata). We use the *generalized powerset construction* to define a generic (trace) semantics for $\mathbb{T}$-automata, and we show by numerous examples that it correctly instantiates for some known classes of machines/languages captured by the Chomsky hierarchy. Moreover, our approach provides new generic techniques for studying expressivity power of various machine-based models.

1 Introduction

In recent decades much interest has been drawn to studying generic abstraction devices not only formally generalizing various computation models and tools, but also identifying core principles and reasoning patterns behind them. An example of this kind is given by the notion of *computational monad* [21], which made an impact both on the theory of programming (as an organization tool for denotational semantics [10, 23]) and on the practice (e.g. being implemented as a programming language feature of Haskell [1] and F# [32]). Another example is given by the theory of *coalgebras* [26], which provides a uniform framework for concurrency theory and observational semantics of systems.

In this paper, we use previous work on monads and coalgebras to give a combined (bialgebraic) perspective of the classical automata theory as well as of some less standard models such as weighted automata. This does not only provide a unifying framework to study various computational models but also suggests new perspectives for proving expressivity bounds for various kinds of machines in a generic way.

We base our framework on the notion of a $\mathbb{T}$-*automaton*, i.e. a coalgebra of the form

$$m : X \to B \times (TX)^A,$$

where T is the functor part of a monad $\mathbb{T}$, which we understand as a mathematical abstraction of a *computational effect* (in the sense of [21]) happening in conjunction with

* An extended version of our paper containing all proofs is avaiblable at `http://arxiv.org/abs/1401.5277`.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 265–280, 2014.

state transitions of the automaton; A is the set of inputs; and B is the set of outputs. According to this view, e.g. nondeterminism is the underlying effect of nondeterministic finite state machines. Analogously, we show that certain (nondeterministic) transformations of the pushdown store form the underlying effect of pushdown automata, etc. By instantiating the operational analysis of computational effects from [23] to our setting we arrive at syntactic fixpoint expressions representing $\mathbb{T}$-automata and prove a Kleene-style theorem for them, thus generalizing previous work of the third author [30].

A crucial ingredient of our framework is the *generalized powerset construction* [31], which serves as a coalgebraic counterpart of classical Rabin-Scott determinization algorithm [25]. It allows us to define *trace semantics* of $\mathbb{T}$-automata and fixpoint expressions denoting their behavior.

We give a formal argument indicating that it is unlikely to capture languages beyond $\mathsf{NTIME}(n)$ using coalgebraic (trace) semantics in a straightforward way (i.e., in our case, using the generalized powerset construction) — the phenomenon known before as a property of *real-time machines* [4]. The requirement to be real-time is an inherent coalgebraic phenomenon of reactivity (or productivity), which restricts the class of behaviors that can be modeled. This led us to formulate a more general *observational semantics*, that allows us to take into account internal (or silent τ-)transitions coalgebraically. The latter furthermore enabled us to capture recursively enumerable languages by a special instance of $\mathbb{T}$-automata called tape automata and that are very similar to Turing machines. Capturing any kind of Turing complete formalism by coalgebras has been a long standing open problem, to which the present paper provides an answer. This results brings us closer to having a coalgebraic Chomsky hierarchy and a new abstract understanding of computability theory.

Related Work. We build on previous work on coalgebraic modelling and monad-based semantics. Most of the applications of coalgebra to automata and formal languages however addressed rational models (e.g. rational streams, regular languages) from which we note [27] (regular languages and finite automata), [15] (bialgebraic view of Kleene algebra and regular expressions), [30, 20, 22, 3] (coalgebraic regular expressions). More recently, some further generalizations were proposed. In recent work [34] a coalgebraic model of context-free grammars is given, without however an analogous treatment of push-down automata. In [13] some initial results on $\mathbb{T}$-automata over stacks by the first author were presented, which the present work extends considerably.

2 Preliminaries: Deterministic Moore Automata, Coalgebraicaly

In this section we recall the main definitions and existing results on coalgebraic modelling of state machines. This material, as well as the material of the following sections, uses the language of category theory, hence we assume readers to be familiar with basic notions. We use $\mathbf{Set}$ as the main underlying category throughout. Further abstraction from $\mathbf{Set}$ to a more general category, while possible (and often quite straightforward), will not be pursued in this paper. The central notion in this paper is that of an F-*coalgebra* is a pair $(X, f : X \to FX)$ where F is an endofunctor on $\mathbf{Set}$ called *transition type*, X is a set called the *state space* and f is a map called *transition structure*. We shall occasionally identify a coalgebra with its state space if no confusion arises.

Coalgebras of a fixed transition type F form a category whose morphisms are maps of the state spaces commuting with the transition structure: $h\colon X \to Y$ is a coalgebra morphism from $(X, f\colon X \to FX)$ to $(Y, g\colon Y \to FY)$ iff $g \circ h = Fh \circ f$. A final object of this category (if it exists) plays a particularly important role and is called *final coalgebra*. We denote the final F-coalgebra by $(\nu F, \iota\colon \nu F \to F\nu F)$, and write $\hat{f}\colon X \to \nu F$ for the unique homomorphism from (X, f) to $(\nu F, \iota)$.

Our core example is the standard formalization of Moore automata as coalgebras [26]. For the rest of the paper we fix a finite set A of *actions* and a set B of *outputs*. We call the functor $L = B \times (\text{–})^A$ the *language functor* (over A, B). The coalgebras for L are given by a set X of states with a transition structure on X given by maps

$$o : X \to B \qquad \text{and} \qquad \partial_a : X \to X, \qquad (a \in A)$$

where the left-hand map, called the *observation map*, represents an output function (e.g. an acceptance predicate if $B = 2$) and the right-hand maps, called *a-derivatives*, are the next state functions indexed by input actions from A. Finite L-coalgebras are hence precisely classical Moore automata. It is straightforward to extend a-derivatives to w-derivatives with $w \in A^*$ by induction: $\partial_\epsilon(x) = x$; $\partial_{aw}(x) = \partial_a(\partial_w(x))$.

The final L-coalgebra νL always exists and is carried by the set of all *formal power series* B^{A^*}. The transition structure is given by $o(\sigma) = \sigma(\epsilon)$ and $\partial_a(\sigma) = \lambda w.\sigma(aw)$ for every formal power series $\sigma : A^* \to B$. The unique homorphism from an L-coalgebra X to the B^{A^*} assigns to every state $x_0 \in X$ a formal power series that we regard as the *(trace) semantics* of X with x_0 as an initial state. Specifically, if $B = 2$ then finite L-coalgebras are deterministic automata and $B^{A^*} \cong \mathcal{P}(A^*)$ is the set of formal languages on A and the trace semantics assigns to every state of a given finite deterministic automaton the language accepted by that state.

Definition 2.1 (Trace Semantics, Trace Equivalence). Given an L-coalgebra (X, f) and $x \in X$, we write $[\![-]\!]_X : X \to B^{A^*}$ for the unique L-coalgebra morphism. For every $x \in X$ we call $[\![x]\!]_X$ the *trace semantics* of x (w.r.t. X). *Trace equivalence* identifies exactly those x and y for which $[\![x]\!]_X = [\![y]\!]_Y$ (for possibly distinct coalgebras X and Y); this is denoted by $x \sim y$.

The following result easily follows by definition (see e.g. [27, Theorem 9.1]).

Proposition 2.2. *Given $x \in X$ and $y \in Y$ where X and Y are L-coalgebras, $x \sim y$ iff for any $w \in A^*$, $o(\partial_w(x)) = o(\partial_w(y))$.*

It is well-known that Moore automata, i.e. *finite* L-coalgebras, can be characterized in terms of the formal power series occurring as their trace semantics (see e.g. [27]).

Definition 2.3 (Regular Power Series). We call a formal power series σ *regular* if the set $\{\partial_w(\sigma) \mid w \in A^*\}$ is finite.

The following result is a rephrasing of a classical result on Moore automata (see e.g. Eilenberg [9]).

Proposition 2.4. *A formal power series is accepted by a Moore automaton if and only if it is regular.*

Remark 2.5. Formal power series are usually considered when $B = k$ is a semiring, in which case one usually also speaks of *rational formal power series* as behaviours of finite weighted automata over k (see e.g. [8]). Our notion of *regular formal power series* (Definition 2.3) generally disagrees with the latter one (unless B is finite) and is in conceptual agreement with such notions as 'regular events' and 'regular trees' [12, 7]. Regular formal power series as the semantics of precisely the finite L-coalgebras are a special instance of a general coalgebraic phenomenon [2, 20]. Let F be any finitary endofunctor on $\mathbf{Set}$. Define the set ϱF to be the union of images of all *finite* F-coalgebras $(X, f : X \to FX)$ under the final morphism $\hat{f} : X \to \nu F$. Then ϱF is a subcoalgebra of νF with an isomorphic transition structure map called the *rational fixpoint* of F. It is (up to isomorphism) uniquely determined by either of the two following universal properties: (1) as an F-coalgebra it is the final locally finite coalgebra and (2) as an F-algebra it is the initial iterative algebra. We refer to [2] for more details.

The characteristic property of regular formal power series can be used as a definitional principle. In fact, given a regular power series σ and assuming that $A = \{a_1, \ldots, a_n\}$, we can view $\{\sigma_1, \ldots, \sigma_k\} = \{\partial_w(\sigma) \mid w \in A^*\}$ as the solution of a system of recursive equations of the form

$$\sigma_i = a_1.\sigma_{i_1} \pitchfork \cdots \pitchfork a_n.\sigma_{i_n} \pitchfork c_i, \quad i = 1, \ldots, k, \tag{2.1}$$

which should be read as follows: for all $1 \leq i, j \leq k$, $\partial_{a_j}(\sigma_i) = \sigma_{i_j}$ and $\sigma_i(\epsilon) = c_i$. Here we introduce $\pitchfork$ as a notation allowing us to syntactically glue together the information about the "head" of a regular formal series and all its derivatives. Reading the $\sigma_1, \ldots, \sigma_k$ as recursion variables, the system (2.1) uniquely determines the corresponding regular power series: for every i it defines $\sigma_i(\epsilon)$ as c_i and for $w = au$ it reduces calculation of $\sigma_i(w)$ to calculation of some $\sigma_j(u)$ — this induction is obviously well-founded.

Any recursive equation (2.1) can be compactly written as

$$\sigma_i = \mu\sigma_i.\ (a_1.\sigma_{i_1} \pitchfork \cdots \pitchfork a_n.\sigma_{i_n} \pitchfork c_i) \tag{2.2}$$

where μ is the fixpoint operator binding the occurrences of σ_i in the right term. One can successively eliminate all the σ_i except σ using the equations (2.2) as assignments and thus obtain a "solution" $\sigma = t$ of (2.1) in σ where t is a closed term given by the following grammar:

$$\gamma ::= \mu X.\ (a_1.\delta \pitchfork \cdots \pitchfork a_n.\delta \pitchfork B) \qquad\qquad \delta ::= X \mid \gamma \tag{2.3}$$

Here X refers to an infinite stock of variables. Equation $\sigma = t$ is then nothing but a condensed representation of system (2.1) and as such it uniquely defines σ. On the other hand, expressions of the form (2.3) suggest a far reaching generalization of classical regular expressions and the fact that they capture exactly regular power series together with Proposition 2.4 can be viewed as a coalgebraic reformulation of Kleene's theorem. This view has been advanced recently (in a rather more general form) in [30, 22] and is of crucial importance for the present work.

Proposition 2.4 in conjunction with the presentation of regular formal power series as expressions (2.3) suggest that every expression gives rise to a finite L-coalgebra

generated by it, whose state space consists of expressions. This is indeed true and can be viewed as a coalgebraic counterpart of the classical Brzozowski's theorem for regular expressions [5]. Given an expression $e = \mu x.\,(a_1.e_1 \pitchfork \cdots a_n.e_n \pitchfork c)$, let

$$o(e) = c \qquad \text{and} \qquad \partial_{a_i}(e) = e_i[e/x]. \tag{2.4}$$

Proposition 2.6. *Let e be a closed expression* (2.3)*. Then the set $\{\partial_w(e) \mid w \in A^*\}$ forms a finite L-coalgebra under the transition structure* (2.4)*.*

3 Monads and Algebraic Theories

In the previous section we have presented a coalgebraic picture of deterministic Moore automata, essentially capturing the Type-3 level of Chomsky hierarchy (modulo the generalization from languages to power series). In order to deal with other levels we introduce *(finitary) monads* and *algebraic theories* as a critical ingredient of our formalization, thus building on top of the recent previous work [17, 31].

In this work we find it easiest to work with monads in the form of *Kleisli triples*.

Definition 3.1 (Kleisli Triple). A Kleisli triple $(T, \eta, \text{–}^\dagger)$ consists of an object assignment T sending sets to sets, a family of maps $\eta_X : X \to TX$ and an operator, called *Kleisli lifting*, sending any $f : X \to TY$ to $f^\dagger : TX \to TY$. These data are subject to the following axioms: $\eta^\dagger = \mathrm{id}$, $f^\dagger \eta = f$ and $(f^\dagger g)^\dagger = f^\dagger g^\dagger$.

It is well-known that the definition of a monad as a Kleisli triple is equivalent to the usual definition of a monad $\mathbb{T}$ as an endofunctor T equipped with natural transformations $\eta : Id \to T$ (*unit*) and $\mu : T^2 \to T$ (*multiplication*). A $\mathbb{T}$-*algebra* is given by a set X and a map $f : TX \to X$ satisfying standard coherence conditions: $f\eta_X = \mathrm{id}_X$ and $\mu_X Tf = f\mu_X$, and a morphism of $\mathbb{T}$-algebras is just a morphism of algebras for the functor T (see [19]). The category of $\mathbb{T}$-algebras and their morphisms is called *Eilenberg-Moore category of* $\mathbb{T}$ and is denoted by $\mathbf{Set}^{\mathbb{T}}$.

In what follows we occasionally use Haskell-style do-notation: for any $p \in TX$ and $q : X \to TY$ we write $\mathsf{do}\ x \leftarrow p; q(x)$ to denote $q^\dagger(p) \in TY$; and $p \in T(X \times Y)$ we write $\mathsf{do}\ \langle x, y\rangle \leftarrow p; q(x, y)$. This notation allows for a more convenient point-full reasoning with Kleisli morphisms, effectively avoiding potential tedious calculations due to strength. A monad $\mathbb{T}$ is *finitary* if the underlying functor T is finitary, i.e., T preserves filtered colimits. Informally, T being finitary means that T is determined by its action on finite sets. In addition, finitary monads admit an equivalent presentation in terms of (finitary) *algebraic theories*.

Definition 3.2 (Algebraic Theory). An *algebraic signature* Σ consists of operation symbols f, each of which comes together with its *arity* n, which is a nonnegative integer — we denote this by $f\colon n \to 1$. Symbols of zero arity are also called *constants*. Terms over Σ are constructed from operations and variables in the usual way. An *algebraic theory* over Σ is given by a set of term equations closed under under inference of the standard equational logic. As usual, an algebraic theory arises as the deductive closure of a set of its *axioms*.

Example 3.3 (Monads, Algebraic Theories). Standard examples of computationally relevant monads include (cf. [21]):

– The *finite and unbounded powerset monads* $\mathcal{P}_\omega$ and $\mathcal{P}$. Only the first one is finitary and corresponds to the algebraic theory of join-semilattices with bottom.
– The *store monad* over a store S. The functorial part given as $X \mapsto (X \times S)^S$. Typically, S is the set of maps $L \to V$ from locations L to values V. A function $f : X \to (Y \times S)^S$ represents a computation that takes a value in X and, depending on the current contents of the store S returns a value in Y and a new store content. As shown in [24], if V is finite then the corresponding store monad can be captured by an algebraic theory over operations $\{lookup_l : V \to 1\}_{l \in L}$ and $\{update_{l,v} : 1 \to 1\}_{l \in L, v \in V}$.
– The *continuation monad*. Given any set R, the assignment $X \mapsto (R^X \to R)$ yields a monad under the following definitions: $\eta(x) = \lambda f.\, f(x)$ and $f^\dagger(k) = \lambda c.\, k(\lambda x.\, f(x)(c))$. This monad is known to be non-finitary, unless $R = 1$.

The following class of examples is especially relevant for the coalgebraic modelling of state-based systems.

Example 3.4 (Semimodule Monad, Semimodule Theory). Given a semiring R, the semimodule monad $\mathbb{T}_R$ assigns to a set X the free left R-semimodule $\langle X \rangle_R$ over X. Explicitly, $\langle X \rangle_R$ consists of all formal linear combinations of the form

$$r_1 \cdot x_1 + \cdots + r_n \cdot x_n \qquad (r_i \in R, x_i \in X). \tag{3.1}$$

Equivalently, the elements of $\langle X \rangle_R$ are maps $f : X \to R$ with finite support (i.e. with $|\{x \in X \mid f(x) \neq 0\}| < \omega$). The assignment $X \mapsto \langle X \rangle_R$ extends to a monad, which we call the *(free) semimodule monad*: η_X sends any $x \in X$ to $1 \cdot x$ and $\sigma^\dagger(p)$ applies the substitution $\sigma : X \to \langle Y \rangle_R$ to $p \in \langle X \rangle_R$ and renormalizes the result as expected.

The semimodule monad corresponds to the algebraic theory of R-semimodules. Explicitly, we have a constant $\emptyset : 0 \to 1$, a binary operation $+ : 2 \to 1$, and every $r \in R$ gives rise to a unary operation $\bar{r} : 1 \to 1$. Terms of the theory are then build over these operations and modulo the laws of commutative monoids for $+$ and $\emptyset$, plus the following ones of a (left) action of R on a monoid:

$$\begin{aligned} \bar{r}(x + y) &= \bar{r}(x) + \bar{r}(y) & \bar{r}(\emptyset) &= \emptyset \\ \bar{r}(x) + \bar{s}(x) &= \overline{r + s}(x) & \bar{0}(x) &= \emptyset \\ \bar{r}(\bar{s}(x)) &= \overline{r \cdot s}(x) & \bar{1}(x) &= x \end{aligned}$$

It can be shown that any term can by normalized to the form $\bar{r}_1(x_1) + \cdots + \bar{r}_n(x_n)$ and the latter coherently represents the element (3.1) of $\langle X \rangle_R$, which allows us to identify them. Some notable instances of the semimodule monad $\mathbb{T}_R$ for semirings R of interest are the following:

– If R is the Boolean semiring $\{0, 1\}$ then $\mathbb{T}_R$ is (isomorphic to) the finite powerset monad $\mathcal{P}_\omega$.
– If R is the semiring of natural numbers then $\mathbb{T}_R$ is the *multiset* monad: the elements of $\langle X \rangle_R$ are in bijective correspondence with finite multisets over X.

– If R is the interval $[0, +\infty)$ then $\mathbb{T}_R$ is the monad of *finite valuations* used for modelling probabilistic computations [33].

Finally, the following example is critical for modelling the push-down store.

Example 3.5 (Stack Monad, Stack Theory). Given a finite set of stack symbols Γ, the *stack monad (over Γ)* is the submonad $\mathbb{T}$ of the store monad $(- \times \Gamma^*)^{\Gamma^*}$ for which the elements $\langle r, t\rangle$ of $TX \subseteq (X \times \Gamma^*)^{\Gamma^*}$ satisfy the following restriction: there exists k depending on r, t such that for every $w \in \Gamma^k$ and $u \in \Gamma^*$, $r(wu) = r(w)$ and $t(wu) = t(w)u$. Intuitively, a function $f : X \to TY$ (cf. Example 3.3) has to compute its output in Y and result stack in Γ^* using only a portion of the stack of a predeclared size k that does not depend on the current content of the stack.

The *stack theory* w.r.t. $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ consists of operations $pop : n + 1 \to 1$ and $push_i : 1 \to 1$ $(1 \leq i \leq n)$. The intuition behind these operations is as follows (in each case the variables under an operation represent continuations, i.e. computations that will be performed once the operation has completed its task, cf. [23]):

– $pop(x_1, \ldots, x_n, y)$ proceeds with y if the stack is empty; otherwise it removes the top element of it and proceeds with x_i where $\gamma_i \in \Gamma$ is the removed stack element.
– $push_i(x)$ adds $\gamma_i \in \Gamma$ on top of the stack and proceeds with x.

These operations are subject to the following axioms:

$$push_i(pop(x_1, \ldots, x_n, y)) = x_i$$
$$pop(push_1(x), \ldots, push_n(x), x) = x$$
$$pop(x_1, \ldots, x_n, pop(y_1, \ldots, y_n, z)) = pop(x_1, \ldots, x_n, z)$$

As shown in [13] the stack theory is precisely the algebraic theory of the stack monad.

Finally we introduce a monad and the corresponding theory underlying the tape of a Turing machine. We introduce the following notation: given an integer $i \in \mathbb{Z}$, a nonnegative integer k and a map $\sigma : \mathbb{Z} \to \Gamma$, we write $\sigma =_{i\pm k} \sigma'$ ($\sigma =^{i\pm k} \sigma'$) if $\sigma(j) = \sigma'(j)$ for all j such that $|i - j| \leq k$ ($|i - j| > k$).

Definition 3.6 (Tape Monad, Tape Theory). Let Γ be a finite set of tape symbols. The *tape monad (over Γ)* is the submonad $\mathbb{T}$ of the store monad $(- \times \mathbb{Z} \times \Gamma^{\mathbb{Z}})^{\mathbb{Z}\times\Gamma^{\mathbb{Z}}}$ for which TX consists of exactly those maps $\langle r, z, t\rangle : \mathbb{Z} \times \Gamma^{\mathbb{Z}} \to (X \times \mathbb{Z} \times \Gamma^{\mathbb{Z}})$, which satisfy restriction: there is $k \geq 0$ such that for any $i, j \in \mathbb{Z}$ and $\sigma, \sigma' : \mathbb{Z} \to \Gamma$ if $\sigma =_{i\pm k} \sigma'$ then

$$t(i, \sigma) =_{i\pm k} t(i, \sigma'), \qquad r(i, \sigma) = r(i, \sigma'), \qquad |z(i, \sigma) - i| \leq k,$$
$$t(i, \sigma) =^{i\pm k} \sigma, \qquad z(i, \sigma) = z(i, \sigma'), \qquad t(i, \sigma_{+j}) = t(i + j, \sigma)_{+j},$$
$$r(i, \sigma_{+j}) = r(i + j, \sigma), \qquad z(i, \sigma_{+j}) = z(i + j, \sigma) - j.$$

where σ_{+j} denotes $\sigma \circ (\lambda i.\, i + j)$. The *tape signature* w.r.t. $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ consists of the operations $read : n \to 1$, $write_i : n \to 1$ $(1 \leq i \leq n)$, $lmove : 1 \to 1$, $rmove : 1 \to 1$, which we interpret over any TX as follows:

$$[\![read]\!](p_1, \ldots, p_n)(z, \sigma) = p_{\sigma(z)}(z, \sigma) \qquad [\![lmove]\!](p)(z, \sigma) = p(z - 1, \sigma)$$
$$[\![write_i]\!](p)(z, \sigma) = p(z, \sigma[z \mapsto \gamma_i]) \qquad [\![rmove]\!](p)(z, \sigma) = p(z + 1, \sigma)$$

where $\sigma[z \mapsto \gamma]$ overwrites σ with the assignment $z \mapsto \gamma$, that is: $\sigma[z \mapsto \gamma](z) = \gamma$ and $\sigma[z \mapsto \gamma](z') = \sigma(z')$ for $z' \neq z$. The *tape theory* w.r.t. Γ consist of all those equations $p = q$ in the tape signature, which are valid over every TX.

In contrast to the stack theory the tape theory is given indirectly. We leave the question of finding an appropriate complete axiomatization of the tape theory for future work. Meanwhile, we report the following surprising result:

Theorem 3.7. *The tape theory is not finitely axiomatizable.*

Proof (Sketch). It is easy to verify that for any i, j, k the following identity

$$write_i(lmove^k(write_j(rmove^k(x)))) = lmove^k(write_j(rmove^k(write_i(x))))$$

belongs to the tape theory. The left-hand term represents a computation that first writes γ_i at the current head position then moves k steps to the left, writes γ_j and finally moves k steps back to the right; the left-hand computation does the same in a different order. It is then possible for any finite set $\mathcal{A}$ of identities to construct a model that does not satisfy the above identity for a suitable k depending on the size of $\mathcal{A}$. □

4 Reactive $\mathbb{T}$-algebras and $\mathbb{T}$-automata

As in Section 2 we consider a finite set of actions A. We first consider $\mathbb{T}$-algebras which are equipped with a transition structure similar to that of Moore automata but which, in addition, preserves the algebraic structure. Such a transition structure extends a $\mathbb{T}$-algebra with dynamic behaviour (making it into a coalgebra) and hence we call such structures *reactive* $\mathbb{T}$-algebras.

Definition 4.1 (Reactive $\mathbb{T}$-algebra). Let B and X be $\mathbb{T}$-algebras. Then X is a *reactive* $\mathbb{T}$-*algebra* if X is an L-coalgebra for which $\partial_a : X \to X$ and $o : X \to B$ are $\mathbb{T}$-algebra morphisms.

Observe that the functor $LX = B \times X^A$ lifts to $\mathbf{Set}^{\mathbb{T}}$; in fact, each LX can be equipped with the pointwise $\mathbb{T}$-algebra structure. Thus, reactive $\mathbb{T}$-algebras are simply coalgebras for this lifting of L to $\mathbf{Set}^{\mathbb{T}}$.

Given a $\mathbb{T}$-algebra B, the set of all formal power series B^{A^*} being the final L-coalgebra can be viewed as a reactive $\mathbb{T}$-algebra with the pointwise $\mathbb{T}$-algebra structure, for which ∂_a and o are easily seen to be $\mathbb{T}$-algebra morphisms. Since any reactive $\mathbb{T}$-algebra is an L-coalgebra, reactive $\mathbb{T}$-algebras inherit the general coalgebraic theory from Section 2. In particular, we use for reactive $\mathbb{T}$-algebras the same notions of trace semantics and trace equivalence as for L-coalgebras.

Definition 4.2 ($\mathbb{T}$-automaton). Suppose, B is finitely generated, i.e. there is a finite set B_0 of *generators* and a surjection $TB_0 \to B$ underlying a $\mathbb{T}$-algebra morphism. A $\mathbb{T}$-*automaton* m is given by a triple of maps

$$o^m : X \to B, \qquad t^m : A \times X \to TX, \qquad a^m : TB \to B, \qquad (\bigstar)$$

where a^m is a $\mathbb{T}$-algebra and X is finite. The first two maps in ($\bigstar$) can be aggregated into a coalgebra transition structure, which we write as $m : X \to B \times (TX)^A$ slightly abusing the notation.

A simple nontrivial example of a $\mathbb{T}$-automaton is given by *nondeterministic finite state machines (NFSM)* by taking $B = \{0,1\}$, $\mathbb{T} = \mathcal{P}_\omega$ and $a^m(s \subseteq \{0,1\}) = 1$ iff $1 \in s$.

In order to introduce the trace semantics of a $\mathbb{T}$-automaton it suffices to convert it into a reactive $\mathbb{T}$-algebra, for the trace semantics of the latter is settled by Definition 2.1. This conversion is called the *generalized powerset construction* [25] as it generalizes the classical Rabin and Scott NFSM determinization [31]. Observe that LTX is a $\mathbb{T}$-algebra, since TX is the free $\mathbb{T}$-algebra on X and L lifts to $\mathbf{Set}^{\mathbb{T}}$. Therefore, given a $\mathbb{T}$-automaton ($\bigstar$), $m : X \to B \times (TX)^A$ there exists a unique $\mathbb{T}$-algebra morphism $m^\sharp : TX \to B \times (TX)^A$ such that $m^\sharp \eta = m$. This $m^\sharp$ is a reactive $\mathbb{T}$-algebra on TX. Therefore, we define the trace semantics of ($\bigstar$) as follows: $[\![x]\!]_m = [\![\eta(x)]\!]_{TX}$.

Note that the generalized powerset construction does not reduce a $\mathbb{T}$-automaton to a Moore automaton over TX as TX need not be finite, although when it is the case, e.g. $\mathbb{T} = \mathcal{P}_\omega$, the semantics of a $\mathbb{T}$-automaton falls within regular power series, which is precisely the reason why the languages recognized by deterministic and nondeterministic FSM coincide. Surprisingly, all $\mathbb{T}$-automata with a finite B have the same property, which is a corollary of Theorem 7.1 we prove in Section 7.

Proposition 4.3. *For every $\mathbb{T}$-automaton ($\bigstar$) with finite B and $x \in X$, $[\![x]\!]_m : A^* \to B$ is regular.*

We are now ready to introduce fixpoint expressions for $\mathbb{T}$-automata similar to (2.3).

Definition 4.4 (Reactive Expressions). Let Σ be an algebraic signature and let B_0 be a finite set. *Reactive expressions* w.r.t. these data are closed terms δ defined according to the following grammar:

$$\begin{aligned} \delta &::= x \mid \gamma \mid f(\delta, \dots, \delta) && (x \in X, f \in \Sigma) \\ \gamma &::= \mu x.\ (a_1.\delta \pitchfork \cdots \pitchfork a_n.\delta \pitchfork \beta) && (x \in X) \\ \beta &::= b \mid f(\beta, \dots, \beta) && (b \in B_0) \end{aligned}$$

where we assume $A = \{a_1, \dots, a_n\}$ and an infinite collection of variables X.

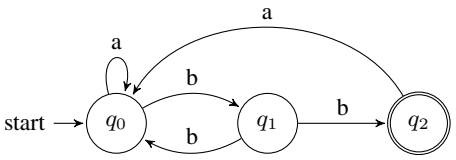
Let $\mathbb{T}$ be a finitary monad, corresponding to an algebraic theory $\mathcal{E}$ over the signature Σ and let B be a finitely generated $\mathbb{T}$-algebra over a finite set of generators B_0. Let us denote by E_{Σ,B_0} the set of all reactive expressions over Σ and B_0. We define a reactive $\mathbb{T}$-coalgebra structure on E_{Σ,B_0}. First, notice that E_{Σ,B_0} is obviously a Σ-algebra. Then we introduce the L-coalgebra structure on E_{Σ,B_0} as follows: first notice that expressions b according to the β-clause in Definition 4.4 are just Σ-terms on the generators from B_0; for every Σ-term t in n variables let $t^B : B^n \to B$ denote the map evaluating t in B and define

$$\begin{aligned} o(f(e_1, \cdots, e_n)) &= f^B(o(e_1), \dots, o(e_n)), \\ \partial_{a_i}(f(e_1, \cdots, e_n)) &= f(\partial_{a_i}(e_1), \dots, \partial_{a_i}(e_n)), \\ o(\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)) &= t^B(b_1, \dots, b_k), \\ \partial_{a_i}(\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)) &= e_i[\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)/x], \end{aligned}$$

where $b = t(b_1, \dots, b_k)$ with $b_1, \dots, b_k \in B_0$.

We call on Definition 2.1 to endow E_{Σ,B_0} with the trace semantics $[\![-]\!] : \mathsf{E}_{\Sigma,B_0} \to B^{A^*}$ and with the trace equivalence relation $\sim$.

$$\begin{cases} q_0 = a.q_0 \pitchfork b.q_1 \pitchfork \bot \\ q_1 = a.\varnothing \pitchfork b.(q_0 + q_2) \pitchfork \bot \\ q_2 = a.q_0 \pitchfork b.\varnothing \pitchfork \top \end{cases}$$



$$q_0 = \mu x.\,(a.x \pitchfork b.\mu y.\,(a.\varnothing \pitchfork b.(x + \mu z.\,(a.x \pitchfork b.\varnothing \pitchfork \top)) \pitchfork \bot) \pitchfork \bot)$$

Fig. 1. A $\mathcal{P}_\omega$-automaton over $A = \{a, b\}$, $B = \{\top, \bot\}$ as a system of recursive definitions (left); as a nondeterministic FSM (right); as a reactive expression (bottom)

Theorem 4.5. *The quotient $\mathsf{E}_{\Sigma,B_0}/{\sim}$ is a reactive $\mathbb{T}$-algebra whose L-coalgebra part is inherited from E_{Σ,B_0} and whose $\mathbb{T}$-algebra part is a quotient of the Σ-algebra structure on E_{Σ,B_0}.*

The following theorem is the main result of this section — a Kleene type theorem. Like its classical counterpart it allows to convert $\mathbb{T}$-automata to closed expressions and vice versa.

Theorem 4.6 (Kleene Theorem). *For any expression $e \in \mathsf{E}_{\Sigma,B_0}$ there is a corresponding $\mathbb{T}$-automaton ($\bigstar$) and $x \in X$ such that $[\![e]\!] = [\![x]\!]_m$; and conversely for any $\mathbb{T}$-automaton ($\bigstar$) and $x \in X$ there is an expression $e \in \mathsf{E}_{\Sigma,B_0}$ such that $[\![e]\!] = [\![x]\!]_m$.*

Fig. 1 depicts a simple instance of the general correspondence established by Theorem 4.6 in the particular standard case of nondeterministic FSM.

5 $\mathbb{T}$-automata: Examples

As indicated above, a nondeterministic FSM is a specific case of a $\mathbb{T}$-automaton under $B = 2$ and $\mathbb{T} = \mathcal{P}_\omega$. More generally, we have the following definition.

Definition 5.1 (Weighted $\mathbb{T}$-automata). *Weighted $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton ($\bigstar$) with $\mathbb{T}$ being the semimodule monad for the semiring R (see Example 3.4).

Let R be the underlying semiring of a semimodule monad $\mathbb{T}$. Besides $R = B = 2$ in which case we obtain nondeterministic FSMs, we obtain standard weighted automata [8] under $R = B = N$ (B is the free $\mathbb{T}$-algebra finitely generated by $\{1\}$).

Weighted $\mathbb{T}$-automata can be further generalized as follows. We call a monad *additive* (cf. [6]) if the corresponding algebraic theory supports operations $+ : 2 \to 1$ and $\emptyset : 0 \to 1$ subject to the axioms of commutative monoids. We call a $\mathbb{T}$-automaton *additive* if $\mathbb{T}$ is additive. Additive automata allow for a more relaxed syntax of reactive expressions. Specifically, we define *additive reactive expressions* as closed *guarded* expressions over an additive theory given by the grammar

$$\gamma ::= b \mid x \mid \mu x.\,\gamma \mid a.\gamma \mid f(\gamma, \ldots, \gamma), \tag{5.1}$$

where guardedness means that for any subterm of the form $\mu x.\,e$ (the recursive call of) x is guarded in e, which is defined by induction over e as follows: x is guarded in b, in

any variable $x' \neq x$, in any $\mu x.\, e'$ and in any $a.e'$; and x is guarded in $f(e_1, \ldots, e_n)$ whenever x is guarded in each of the e_i.

Given a reactive expression we obtain an additive reactive expression by replacing recursively each $\pitchfork$ with $+$. Conversely, any additive reactive expression can be transformed to a reactive one. The latter transformation is inverse to the former modulo $\sim$. We now give one example of an additive $\mathbb{T}$-automaton, which is not a weighted $\mathbb{T}$-automaton.

Example 5.2 (Segala $\mathbb{T}$-automata). *(Simple) Segala systems* [28, 29] are systems combining probability and nondeterminism and are essentially coalgebras of transition type $\mathcal{P}(\mathcal{D} \times A) \cong (\mathcal{P}\mathcal{D})^A$ where $\mathcal{D}$ is a probability distribution functor. Although $\mathcal{P}\mathcal{D}$ is not a monad, as elaborated in [16], it can be modelled by a monad $\mathbb{T}$ whose functorial part is the composition CM of two functors given as follows: for any X, MX are finite valuations over X (see Example 3.4); for any semimodule U, $C(U)$ consists of all subsets of U, which are *convex* and nonempty. Convexity of a set S here means that a convex combination $p_1 \cdot \xi_1 + \cdots + p_n \cdot \xi_n$, i.e. $\sum_i p_i = 1$, belongs to S once $\xi_i \in S$ for any i. *Segala $\mathbb{T}$-automata* generalize non-deterministic automata by replacing the powerset functor $\mathcal{P}$ with CM. Concretely, in the generic definition ($\bigstar$) we take $B = 2$ and $\mathbb{T}$ defined as above.

A radically different kind of examples is offered by submonads of the store monad. A prominent instance of such is the stack monad (Example 3.5), which we use for modelling push-down automata.

Definition 5.3 (Stack $\mathbb{T}$-automaton). *Stack $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton ($\bigstar$) for which $\mathbb{T}$ is the stack monad over Γ; B is the set of predicates over Γ^* consisting of all those $p \in 2^{\Gamma^*}$ for each of which there is k such that $p(wu) = p(w)$ whenever $|w| \geq k$; $a^{\mathcal{M}} : TB \to B$ is given by evaluation; it restricts the morphism $\mathsf{ev}^{\Gamma^*} : (2^{\Gamma^*} \times \Gamma^*)^{\Gamma^*} \to 2^{\Gamma^*}$, where $\mathsf{ev} : 2^{\Gamma^*} \times \Gamma^* \to 2$ is the evaluation morphism: $a^{\mathcal{M}}(r, t)(s) = r(s)(t(s))$.

Intuitively, $o^{\mathcal{M}} : X \to B \subseteq 2^{\Gamma^*}$ models the acceptance condition by final states and the stack content. As B obeys essentially the same constraints as TX, scanning an unbounded portion of the stack by $o^{\mathcal{M}}$ is disallowed.

Theorem 5.4. *Let $\mathcal{M}$ be a stack $\mathbb{T}$-automaton. Then for any $x_0 \in X$ and any $\gamma_0 \in \Gamma$, $\{w \in A^* \mid [\![x_0]\!]_{\mathcal{M}}(w)(\gamma_0)\}$ is a real-time deterministic context-free language. The converse is also true: for any real-time deterministic context-free language $\mathcal{L} \subseteq A^*$ there exists a stack $\mathbb{T}$-automaton ($\bigstar$) such that $\mathcal{L}$ can be represented as the above formal language with some $x_0 \in X$, $\gamma_0 \in \Gamma$.*

As we shall see later, it is not difficult to obtain an analogous characterization of context-free languages for which the "real-time" clause is dropped (essentially because for push-down automata the restriction of being real-time is vacuous). However, as we shall see in the next section (Theorem 6.5), this restriction, being somewhat inherent for coalgebraic models, presents an actual boundary for capturing by $\mathbb{T}$-automata formal languages beyond the context-free ones.

6 Monad Tensors for Store and Nondeterminism

Tensor products of monads (resp. algebraic theories) have been introduced by Freyd [11] in the context of universal algebra. Later, computational relevance of this operation has been demonstrated by Hyland et al. [14]. Here, we use tensors of monads as a tool for studying $\mathbb{T}$-automata.

Definition 6.1 (Tensor). Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two algebraic theories. Then the tensor product $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2$ is the algebraic theory, whose equations are obtained by joining the equations of $\mathcal{E}_1$ and $\mathcal{E}_2$ and adding for any $f : n \to 1$ of $\mathcal{E}_1$ and any $g : m \to 1$ of $\mathcal{E}_2$ the following axiom

$$f(g(x_1^1, \ldots, x_m^1), \ldots, g(x_1^n, \ldots, x_m^n)) = g(f(x_1^1, \ldots, x_1^n), \ldots, f(x_m^1, \ldots, x_m^n))$$

called the *tensor laws*. Given two finitary monads $\mathbb{T}_1$ and $\mathbb{T}_2$, their tensor product arises from the algebraic theory $\mathcal{E}_{\mathbb{T}_1} \otimes \mathcal{E}_{\mathbb{T}_2}$.

Intuitively, tensor product of two monads captures a noninterfering combination of the corresponding computational effects. In the present work we shall use two kinds of tensor products: (1) tensors with *submonads of the store monad* and (2) tensors with *semimodule monads*.

It has been shown in [14] that tensoring with the store monad is equivalent to the application of the store monad transformer sending any monad $\mathbb{T}$ to the *store monad transform* $\mathbb{T}_S$ whose functorial part is given by $T_S X = T(X \times S)^S$. Here we establish a similar result about the stack monad (Example 3.5).

Proposition 6.2. *Let $\mathbb{S}$ be the stack monad over Γ. Then for any finitary $\mathbb{T}$, $\mathbb{S} \otimes \mathbb{T}$ is the submonad $\mathbb{R}$ of the store monad transform of $\mathbb{T}$ with Γ^* as the store, for which $p : \Gamma^* \to T(X \times \Gamma^*)$ is in $\mathbb{R}X$ iff there exists m such that $p(su) = \mathsf{do}\, \langle x, s' \rangle \leftarrow p(s); \eta \langle x, s'u \rangle$ whenever $|s| \geq m$; and the monad structure is inherited from the monad transform.*

One can thus combine two stacks by computing the tensor square of the stack monad. Specifically, the resulting monad $\mathbb{T}$ has maps $\langle r, t_1, t_2 \rangle : \Gamma^* \times \Gamma^* \to X \times \Gamma^* \times \Gamma^*$ as inhabitants of TX. This allows one to define $\mathbb{T}$-stack automata over two and more stacks analogously to the one-stack case from Definition 5.3. Before we do this formally in Definition 6.4 we discuss the perspectives of forming tensors with semimodule monads.

Proposition 6.3 (Freyd [11]). *Tensor product of any finitary monad with a semimodule monad is isomorphic to some semimodule monad.*

Proposition 6.3 in conjunction with Proposition 6.2 offer two perspectives on machines with memory and nondeterminism. E.g. we shall consider the tensor product of $\mathcal{P}_\omega$ with the stack monad to model push-down automata. As Proposition 6.2 indicates, this monad embeds into the monad with functorial part $TX = \mathcal{P}_\omega(X \times \Gamma^*)^{\Gamma^*}$. On the other hand, by Proposition 6.3, this tensor product is equivalent to a semimodule monad.

Definition 6.4 (Multi-stack Nondeterministic $\mathbb{T}$-automaton). A *Multi-stack nondeterministic $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton ($\bigstar$) for which $\mathbb{T}$ is the tensor of m copies of the stack monad and $\mathcal{P}_\omega$; B is the set of m-ary predicates over Γ^* consisting of all

those $p \in 2^{\Gamma^* \times \cdots \times \Gamma^*}$ for each of which there is a k such that if for any i, $|w_i| \geq k$ then $p(w_1 u_1, \ldots, w_m u_m) = p(w_1 \ldots, w_m)$; and for any $s \in (\Gamma^*)^m$, $f : (\Gamma^*)^m \to \mathcal{P}_\omega(B \times (\Gamma^*)^m) \in TB$ we have $a^{\mathfrak{m}}(f)(s)$ iff $\exists s' \in (\Gamma^*)^m.\, \exists p \in B.\ f(s)(p, s') \wedge p(s')$.

We now obtain the following result.

Theorem 6.5. *For any m let $\mathcal{L}_m$ be the following class of all languages $\{w \in A^* \mid [\![x_0]\!]_{\mathfrak{m}}(w)(\gamma_0, \ldots, \gamma_0)\}$ with $\mathfrak{m}$ ranging over nondeterministic multistack $\mathbb{T}$-automata with m stacks, x_0 ranging over the state space of $\mathfrak{m}$ and γ_0 ranging over Γ. Then $\mathcal{L}_1$ contains exactly context-free languages; for all $m > 2$, $\mathcal{L}_m$ contains exactly nondeterministic linear time languages, i.e. $\mathcal{L}_m = \mathsf{NTIME}(n)$; and $\mathcal{L}_2$ sits properly between $\mathcal{L}_1$ and $\mathcal{L}_3$.*

Theorem 6.5 shows, on the one hand, that the coalgebraic formalization of nondeterministic pushdown automata as nondeterministic $\mathbb{T}$-automata over one stack is adequate in the sense that it recognizes the same class of languages. On the other hand, it indicates the boundaries of the present model: it seems unlikely to capture languages beyond $\mathsf{NTIME}(n)$ (e.g. all recursive ones) by a computationally feasible class of $\mathbb{T}$-automata. This is not surprising in view of the early work on (quasi-)real-time recognizable languages [4], which underlies the proof of Theorem 6.5. We return to this issue in Section 7 where we provide an extension of the present semantics that allows us to capture language classes up to recursively enumerable ones.

We conclude this section with an easy corollary of Theorem 6.5 and Proposition 2.2 contrasting the results in [20, 3].

Corollary 6.6. *Trace equivalence of $\mathbb{T}$-automata is Π_1^0-complete.*

7 CPS-transforms of $\mathbb{T}$-automata and r.e.-languages

Theorem 6.5 suggests that the present trace semantics is unlikely to produce languages beyond $\mathsf{NTIME}(n)$ under a computationally convincing choice of the components of ($\bigstar$). The approach suggested by the classical formal language theory is to replace A with the set $A_\tau = A \cup \{\tau\}$, where τ is a new *unobservable action*, but use the formal power series $A^* \to B$ as the semantic domain instead of $A_\tau^* \to B$. The new *observational semantics* is supposed to be obtainable from the standard one by "eliminating" the unobservable action.

Before we proceed, we introduce an important reformulation of our standard trace equivalence, which is of independent interest. Let $a : TB \to B$ be a $\mathbb{T}$-algebra. We denote by $\mathbb{T}_B$ the continuation monad with $T_B X = B^X \to B$. We can map $\mathbb{T}$ to $\mathbb{T}_B$ by sending any $p : TX$ to $\kappa_X(p) = \lambda f.\, (a \cdot Tf(p)) \in T_B X$. This $\kappa : \mathbb{T} \to \mathbb{T}_B$ is a monad morphism; in fact, it is well known that for any monad $\mathbb{T}$ on a category with powers there is a bijective correspondence between Eilenberg-Moore algebras on B and monad morphisms from $\mathbb{T}$ to $\mathbb{T}_B$ (see e.g. Kock [18, Theorem 3.2]).

Now, given a $\mathbb{T}$-automaton ($\bigstar$), we define a $\mathbb{T}_B$-automaton[1] $m_* : X \to B \times (T_B X)^A$ by $o^{m_*} = o^m$, $t^{m_*} = \kappa_X t^m$, $a^{m_*} = \lambda t.\, t(\mathrm{id})$ where it is easy to see that $a^{m_*} : T_B B \to B$ is a $\mathbb{T}$-algebra. We call this automaton the *CPS-transform*[2] of ($\bigstar$).

Theorem 7.1. *The trace semantics of a $\mathbb{T}$-automaton and of its CPS-transform agree; more precisely, for every $\mathbb{T}$-automaton ($\bigstar$) and state $x \in X$ we have: $[\![x]\!]_m = [\![x]\!]_{m_*}$.*

This theorem implies Proposition 4.3 announced previously in Section 4. Indeed, if B in ($\bigstar$) is finite then, by definition, $T_B X$ is also finite. Thus, the generalized powerset construction performed on the CPS-transform m_* yields a Moore automaton, and hence we obtain the desired result from Proposition 2.4.

We now proceed with the definition of the new semantics.

Definition 7.2 (ω-additive $\mathbb{T}$-automata). A $\mathbb{T}$-automaton ($\bigstar$) is *ω-additive* if B (besides being $\mathbb{T}$-algebra) is an algebra for the countably supported multiset monad.

In other words, for ($\bigstar$) to be ω-additive B needs to be a commutative monoid with infinite summation. We call such a monoid *ω-additive*.

Lemma 7.3. *If B is an ω-additive monoid and a $\mathbb{T}$-algebra then for any X, $T_B X$ is an ω-additive monoid.*

The ω-additive monoid structure on $T_B X$ allows us to define for any given $\mathbb{T}$-automaton over the alphabet A_τ a $\mathbb{T}_B$-automaton over A. To this end, we first form the CPS-transform of the given $\mathbb{T}$-automaton and then use infinite summation to get rid of unobservable actions τ: given a $\mathbb{T}$-automaton $m : X \to B \times (TX)^{A_\tau}$, we construct $m_v : X \to B \times (T_B X)^A$ with $a^{m_v} = a^{m_*} = \lambda t.\, t(\mathrm{id})$ and with t^{m_v}, o^{m_v} defined as

$$t^{m_v}(x_0, a) = \sum_{i=1}^{\infty} \mathsf{do}\ x_1 \leftarrow t^{m_*}(x_0, \tau); \ldots; x_{i-1} \leftarrow t^{m_*}(x_{i-2}, \tau); t^{m_*}(x_{i-1}, a),$$

$$o^{m_v}(x_0) = o^{m_*}(x_0) + \sum_{i=1}^{\infty} \big(\mathsf{do}\ x_1 \leftarrow t^{m_*}(x_0, \tau); \ldots; t^{m_*}(x_{i-1}, \tau)\big)(o^{m_*}).$$

We define the observational trace semantics for m to be the trace semantics for m_v.

Definition 7.4. Given a $\mathbb{T}$-automaton ($\bigstar$) over input alphabet A_τ, let $[\![x]\!]^\tau_m = [\![x]\!]_{m_v}$.

We proceed to define the class of $\mathbb{T}$-automata corresponding to classical Turing machines, for which the introduced observational trace semantics yields precisely all recursively enumerable languages.

Definition 7.5 (Tape $\mathbb{T}$-automaton). A *tape automaton* is a $\mathbb{T}$-automaton ($\bigstar$) for which $\mathbb{T}$ is the tape monad over Γ; B is the set of predicates over $\mathbb{Z} \times \Gamma^{\mathbb{Z}}$ consisting of all those $p \in 2^{\mathbb{Z} \times \Gamma^{\mathbb{Z}}}$ for each of which there is a k such that $p(i, \sigma) = p(i, \sigma')$ and $p(i, \sigma_{+j}) = p(i + j, \sigma)$ if $\sigma =_{i \pm k} \sigma'$; and $a^m : TB \to B$ is given by evaluation as in Definition 5.3.

[1] We abuse terminology here since $\mathbb{T}_B$ is not finitary.

[2] CPS = *continuation-passing style*; we chose the name because the construction is reminiscent of tranforming a functional program into CPS.

It can be shown that tape $\mathbb{T}$-automata over A_τ are equivalent to deterministic 2-tape Turing machines with input alphabet A, where the first tape is a special read-only and right-only tape holding the input word at the beginning of a computation. Thus, we obtain that tape automata represent all the recursively enumerable languages.

Theorem 7.6. *For every tape automaton m over A_τ, Γ with $|\Gamma| \geq 2$ containing a special blank symbol $\square$, and every state $x \in X$ the following language is recursively enumerable: $\{w \in A^* \mid [\![x]\!]^\tau_m(w)(0, \sigma_\square) = 1\}$, where $\sigma_\square$ is the constant function on $\square$. Conversely, every recursively enumerable language can be represented in this way.*

8 Conclusions and Future Work

In this paper, we have presented the first steps towards a coalgebraic Chomsky hierarchy. We have given a coalgebraic account of machines, languages and expressions and presented several results of our theory including a generic Kleene-style theorem (Theorem 4.6). We have also given the first treatment of Turing machines in a coalgebraic setting: the observational trace semantics of tape automata yields precisely the recursively enumerable languages.

There are several possible directions for future work. We plan to derive a sound calculus of reactive expressions extending [3] and explore the boundaries for completeness (by Corollary 6.6 completeness is only possible for specific choices of $\mathbb{T}$); capture further language and complexity classes, such as the context-sensitive languages. Capturing various classes of machines under the umbrella of coalgebra results in standard tools such as bisimulation proof methods becoming available for those classes of machines and their language semantics. Hence, further investigations into such proof principles are of interest.

Acknowledgements. We would like to thank anonymous referees for their suggestions on improving the presentation and for pointing out relevant literature.

References

[1] Haskell 98 Language and Libraries — The Revised Report. Cambridge University Press (2003); also: J. Funct. Prog. 13 (2003)
[2] Adámek, J., Milius, S., Velebil, J.: Iterative algebras at work. Math. Structures Comput. Sci. 16(6), 1085–1131 (2006)
[3] Bonsangue, M.M., Milius, S., Silva, A.: Sound and complete axiomatizations of coalgebraic language equivalence. ACM Trans. Comput. Log. 14(1), 7:1–7:52 (2013)
[4] Book, R.V., Greibach, S.A.: Quasi-realtime languages. Mathematical Systems Theory 4(2), 97–111 (1970)
[5] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
[6] Coumans, D., Jacobs, B.: Scalars, monads, and categories. In: Sadrzadeh, C.H.M., Grefenstette, E. (eds.) Quantum physics and linguistics. A compositional, diagrammatic discourse, pp. 184–216. Oxford University Press (2013)
[7] Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science 25(2), 95–169 (1983)
[8] Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer (2009)

[9] Eilenberg, S.: Automata, Languages, and Machines. Pure and Applied Mathematics, vol. A. Academic Press (1974)
[10] Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the π-calculus. Inf. Comput. 179(1), 76–117 (2002)
[11] Freyd, P.: Algebra valued functors in general and tensor products in particular. Colloq. Math. 14, 89–106 (1966)
[12] Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. J. ACM 24(1), 68–95 (1977)
[13] Goncharov, S.: Trace semantics via generic observations. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 158–174. Springer, Heidelberg (2013)
[14] Hyland, M., Levy, P.B., Plotkin, G.D., Power, J.: Combining algebraic effects with continuations. Theor. Comput. Sci. 375(1-3), 20–40 (2007)
[15] Jacobs, B.: A bialgebraic review of deterministic automata, regular expressions and languages. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 375–404. Springer, Heidelberg (2006)
[16] Jacobs, B.: Coalgebraic trace semantics for combined possibilitistic and probabilistic systems. Electr. Notes Theor. Comput. Sci. 203(5), 131–152 (2008)
[17] Jacobs, B., Silva, A., Sokolova, A.: Trace semantics via determinization. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 109–129. Springer, Heidelberg (2012)
[18] Kock, A.: On double dualization monads. Math. Scand. 27, 151–165 (1970)
[19] Lane, S.M.: Categories for the Working Mathematician. Springer (1971)
[20] Milius, S.: A sound and complete calculus for finite stream circuits. In: Proc. 25th Annual Symposium on Logic in Computer Science (LICS 2010), pp. 449–458. IEEE Computer Society (2010)
[21] Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)
[22] Myers, R.: Rational Coalgebraic Machines in Varieties: Languages, Completeness and Automatic Proofs. Ph.D. thesis, Imperial College London (2013)
[23] Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002)
[24] Power, J., Shkaravska, O.: From comodels to coalgebras: State and arrays. In: CMCS 2004. ENTCS, vol. 106, pp. 297–314 (2004)
[25] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 114–125 (1959)
[26] Rutten, J.: Universal coalgebra: A theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
[27] Rutten, J.J.M.M.: Behavioural differential equations: A coinductive calculus of streams, automata, and power series. Theor. Comput. Sci. 308(1-3), 1–53 (2003)
[28] Segala, R.: Modelling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
[29] Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
[30] Silva, A.: Kleene coalgebra. Ph.D. thesis, Radboud Univ. Nijmegen (2010)
[31] Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing determinization from automata to coalgebras. LMCS 9(1) (2013)
[32] Syme, D., Granicz, A., Cisternino, A.: Expert F#. Apress (2007)
[33] Varacca, D., Winskel, G.: Distributing probability over non-determinism. Math. Struct. Comput. Sci. 16, 87–113 (2006)
[34] Winter, J., Bonsangue, M.M., Rutten, J.J.M.M.: Coalgebraic characterizations of context-free languages. LMCS 9(3) (2013)

Strong Completeness for Iteration-Free Coalgebraic Dynamic Logics

Helle Hvid Hansen[1,2,⋆], Clemens Kupke[3,⋆⋆], and Raul Andres Leal[4]

[1] Radboud University Nijmegen & CWI Amsterdam, The Netherlands
[2] University of Strathclyde, UK
[3] Cap Gemini, Amsterdam, The Netherlands

Abstract. We present a (co)algebraic treatment of iteration-free dynamic modal logics such as Propositional Dynamic Logic (PDL) and Game Logic (GL), both without star. The main observation is that the program/game constructs of PDL/GL arise from monad structure, and the axioms of these logics correspond to certain compatibilty requirements between the modalities and this monad structure. Our main contribution is a general soundness and strong completeness result for PDL-like logics for T-coalgebras where T is a monad and the "program" constructs are given by sequential composition, test, and pointwise extensions of operations of T.

1 Introduction

Modal logics are a much used formalism in automated verification thanks to the good balance between their expressive power and their computational properties. Recently, it has been shown that modal logics can be developed in the general framework of coalgebra [4,18], and that the expressiveness and complexity results for Kripke semantics hold more generally across many types of structures [29,30].

In this paper, we aim to develop a coalgebraic framework for dynamic modal logics such as Propositional Dynamic Logic (PDL) [5,12] and Game Logic (GL) [25,26]. In PDL, modalities are indexed by programs whose semantics is given by relations, and program constructs are interpreted by relation algebra. Similarly, in GL, modalities are indexed by games whose semantics is given by monotonic neighbourhood functions.

Our framework for coalgebraic dynamic modal logic builds on the basic observation that in PDL and GL programs/games are interpreted as maps of the form $X \to TX$ where T is a monad. For PDL, T is the covariant powerset monad, and for GL, T is the monotonic neighbourhood monad (both are described in detail later). Such maps can be viewed as arrows in the Kleisli category of the monad T which yields semantics of sequential composition as Kleisli composition. Alternatively, a map $X \to TX$ can be viewed as a T-coalgebra which

⋆ Supported by NWO-Veni grant 639.021.231.
⋆⋆ Supported by University of Strathclyde starter grant.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 281–295, 2014.

leads to a (coalgebraic) modal logic of T-computations. Other constructs, such as choice ($\cup$) and dual (^{d}) in GL, are interpreted by algebraic structure on the set $(TX)^X = \{X \to TX\}$ which arises pointwise from algebraic structure on TX. We formalise such constructs using natural operations on functors. We also note that PDL and GL are usually interpreted over so-called standard models, in which the program/game constructs have a certain intended meaning. In our general framework this leads to the notion of a standard model relative to some algebraic structure θ on T. In the current paper, we include tests, but not iteration which will require more assumptions on the monad.

Our main contributions are: (i) a method for associating rank-1 axioms to natural operations, (ii) a method for axiomatising tests, and (iii) strong completeness for the ensuing dynamic modal logic.

The rest of the paper is organised as follows. In Section 2 we recall the basics of PDL and GL, and of coalgebraic modal logic and monads. In Section 3, we introduce our general framework for coalgebraic dynamic modal logic. In Section 4, we show how to obtain axioms for sequential composition and natural operations, and provide sufficient conditions for their soundness. In Sections 5 and 6, we prove our strong completeness result which builds on the generic strong completeness result in [31] by showing that a quasi-canonical model can be modified to validate also the non-rank-1 sequential composition axioms. Finally, in Section 7 we conclude and discuss related work. The proofs can be found in the technical report [10].

2 Preliminaries

2.1 PDL and GL

We briefly recall the basics of the two dynamic modal logics that form our guiding examples. See the references given for more detail and background information.

Propositional Dynamic Logic (PDL) [5,12] is a modal logic for reasoning about program correctness. Modalities are indexed by programs, and a formula $[\alpha]\varphi$ should be read as "after all halting executions of program α, φ holds". PDL programs are built inductively from a set $Prog_0$ of atomic programs using the operations of *sequential composition* (;), *choice* ($\cup$) and *iteration* (*). Moreover, a formula φ can be turned into a program $\varphi?$ by the *test* operation ?. The semantics of PDL is given by multi-modal Kripke models that contain a relation R_α for each program α. These models are generally assumed to be *standard* which means that relations for complex programs are defined inductively via composition, union and reflexive, transitive closure of relations over some given interpretation of atomic programs, and a test program $\varphi?$ is interpreted by restricting the identity relation to the states that satisfy φ. As a deductive system, PDL is the least normal multi-modal logic that contains the axioms:

$$\begin{array}{lll} [\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi & [\alpha \cup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi & [\psi?]\varphi \leftrightarrow (\psi \to \varphi) \\ \varphi \wedge [\alpha][\alpha^*]\varphi \leftrightarrow [\alpha^*]\varphi & \varphi \wedge [\alpha](\varphi \to [\alpha]\varphi) \to [\alpha^*]\varphi & \end{array} \quad (1)$$

for all programs α, β and all formulas ψ, φ. It is well known that PDL is (weakly) complete with respect to the class of standard PDL models. Strong completeness fails due to the presence of * which makes PDL non-compact.

Game Logic (GL) [25,26] is a modal logic for reasoning about strategic ability in determined 2-player games. Here, a modal formula $[\gamma]\varphi$ should be read as "player 1 has a strategy in the game γ to ensure an outcome where φ holds". The modal language of GL is obtained by extending the program operations of PDL with the game operation *dual* (^{d}) which corresponds to a role switch of the two players. Game Logic semantics is given by multi-modal monotone neighbourhood models [3,8,9]. We refer to [25,26] for the details. As a deductive system, GL is defined to be the least monotone multi-modal logic containing the following axioms:

$$\begin{array}{lll} [\gamma;\delta]\varphi \leftrightarrow [\gamma][\delta]\varphi & [\gamma \cup \delta]\varphi \leftrightarrow [\gamma]\varphi \vee [\delta]\varphi & [\psi?]\varphi \leftrightarrow (\psi \wedge \varphi) \\ \varphi \vee [\gamma][\gamma^*]\varphi \rightarrow [\gamma^*]\varphi & \dfrac{\varphi \vee [\gamma]\varphi \rightarrow \psi}{[\gamma^*]\varphi \rightarrow \psi} & [\gamma^d]\varphi \leftrightarrow \neg[\gamma]\neg\varphi \end{array} \quad (2)$$

Both iteration-free GL and dual-free GL are known to be complete for standard GL models (restricted to the appropriate fragment), however, completeness of GL with both * and ^{d} remains an open question. One indication of why completeness for full GL is difficult is that GL can be viewed as a fragment of the modal μ-calculus that spans all levels of the alternation hierarchy [2,26].

2.2 Coalgebraic Modal Logic

Coalgebraic modal logic [4,18] is a general framework which encompasses many known modal logics such as normal, classical, graded and probabiliistic modal logic. The uniform treatment of these is achieved by viewing the corresponding semantic structures as coalgebras for a functor T [28]. In the present paper, we only consider coalgebras for functors on Set, the category of sets and functions. Let T be a Set-(endo)functor. A *T-coalgebra* is a map $\xi\colon X \to TX$, and a *T-coalgebra morphism* from $\xi\colon X \to TX$ to $\xi'\colon X' \to TX'$ is a map $f\colon X \to X'$ such that $\xi' \circ f = Tf \circ \xi$. T-coalgebras and their morphisms form a category $\mathsf{Coalg}(T)$.

We follow the notation from [31] in defining syntax and semantics of coalgebraic modal logic. A modal signature Λ consists of a collection of modal operators with associated arities. Given a modal signature Λ and a countable set P of atomic propositions, the set $\mathcal{F}(\Lambda)$ of Λ-formulas is generated by the following grammar:

$$\varphi \quad ::= \quad p \in P \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_\lambda(\varphi_1, \ldots, \varphi_n)$$

where $\Box_\lambda \in \Lambda$ is n-ary. For any set X, $\mathsf{Prop}(X)$ denotes the set of all propositional formulas over X, and $\Lambda(X) = \{\Box_\lambda(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in X, \Box_\lambda \in \Lambda \text{ is } n\text{-ary}\}$.

Modal formulas will be interpreted in coalgebras. We use the approach to coalgebraic modal logic in which modalities are interpreted via predicate liftings. First, we denote by $\mathcal{Q} : \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}}$ the *contravariant powerset functor* which

maps a set X to its powerset, and a function f to its inverse image map. An *n-ary predicate lifting for T* is a natural transformation $\lambda\colon \mathcal{Q}^n \Rightarrow \mathcal{Q}\circ T$. A *$(\Lambda, T)$-model* $\mathfrak{M}$ consists of a T-coalgebra $\xi\colon X \to TX$, a valuation $V\colon P \to \mathcal{P}(X)$ of atomic propositions, and an n-ary predicate lifting for each n-ary modality in Λ. For formulas $\varphi \in \mathcal{F}(\Lambda)$ the truth set $[\![\varphi]\!]^{\mathfrak{M}}$ is defined in the expected manner for the atomic propositions and Boolean connectives, and for modal formulas, $[\![\Box_\lambda(\varphi_1,\ldots,\varphi_n)]\!]^{\mathfrak{M}} = \xi^{-1}(\lambda_X([\![\varphi_1]\!]^{\mathfrak{M}},\ldots,[\![\varphi_n]\!]^{\mathfrak{M}}))$. The map $\xi^{-1}\circ\lambda_X\colon \mathcal{P}(X)^n \to \mathcal{P}(X)$ is the *n-ary predicate transformer associated with ξ and λ*. In the remainder of this paper, we will only consider unary modalities and unary predicate liftings.

Example 1. The following well known instances of coalgebraic modal logic will be of central interest to the paper. See e.g. [28,29,31] for many other examples.

(i) Coalgebras for the *covariant powerset functor* $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ are Kripke frames, and $\mathcal{P}$-coalgebra morphisms are bounded morphisms. The Kripke box modality is interpreted via the predicate lifting $\lambda^{\Box}_X(U) = \{V \in \mathcal{P}(X) \mid V \subseteq U\}$.

(ii) The *neighbourhood functor* $\mathcal{N} = \mathcal{Q}^{\mathrm{op}}\mathcal{Q}\colon \mathsf{Set} \to \mathsf{Set}$ is the composition of $\mathcal{Q}$ with its dual $\mathcal{Q}^{\mathrm{op}}$. $\mathcal{N}$ maps a set X to $\mathcal{P}(\mathcal{P}(X))$, and function f to the double-inverse-image map $\mathcal{N}(f) = (f^{-1})^{-1}$. An $\mathcal{N}$-coalgebra $\nu\colon X \to \mathcal{N}(X)$ is known in modal logic as a neighbourhood frame, and $\mathcal{N}$-coalgebra morphisms as bounded neighbourhood morphisms [3,11]. The neighbourhood modality is interpreted via the predicate lifting given by $\lambda_X(U) = \{N \in \mathcal{N}(X) \mid U \in N\}$. In this paper we will refer to $\mathcal{N}$-coalgebras as *neighbourhood functions*.

(iii) The *monotone neighborhood functor* $\mathcal{M}\colon \mathsf{Set} \to \mathsf{Set}$ is the subfunctor of $\mathcal{N}$ which maps a set X to the set of upwards closed neighbourhood collections $H \subseteq \mathcal{P}(X)$, i.e., $\mathcal{M}(X) = \{H \in \mathcal{P}(\mathcal{P}(X)) \mid \forall U \subseteq V \subseteq X : U \in H \Rightarrow V \in H\}$, and for a function f, $\mathcal{M}(f)$ is obtained by restricting $\mathcal{N}(f)$ to upwards closed neighbourhood collections. Similarly, for the predicate lifting that interprets the monotonic neighbourhood modality. $\mathcal{M}$-coalgebras are known in modal logic as monotonic neighbourhood frames [3,8,9]. We will refer to $\mathcal{M}$-coalgebras as *monotonic neighbourhood functions*. The name "monotonic" refers to the upwards closure, and will be explained further in the next remark.

Remark 2. Neighbourhood functions and (unary) predicate transformers are essentially the same mathematical objects. This basic correspondence arises from the adjunction of the contravariant powerset functor $\mathcal{Q}\colon \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}}$ with its dual: $\mathcal{Q}\colon \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}} \dashv \mathcal{Q}^{\mathrm{op}}\colon \mathsf{Set}^{\mathrm{op}} \to \mathsf{Set}$. Hence, for all sets X and Y there is a bijection $\mathsf{Set}(X, \mathcal{Q}^{\mathrm{op}}Y) \cong \mathsf{Set}(Y, \mathcal{Q}X)$ given by exponential transpose $f(x)(y) = \widehat{f}(y)(x)$. Taking $Y = \mathcal{Q}X$, we get a bijection $\mathsf{Set}(X, \mathcal{Q}^{\mathrm{op}}\mathcal{Q}X) \cong \mathsf{Set}(\mathcal{Q}X, \mathcal{Q}X)$ between neighbourhood functions and predicate transformers given by $U \in \nu(x)$ iff $x \in \widehat{\nu}(U)$ for all $x \in X$ and $U \subseteq Y$. Note that $\widehat{\nu}\colon \mathcal{Q}(X) \to \mathcal{Q}(X)$ is a monotonic map (w.r.t. set-inclusion) if and only if $\nu\colon X \to \mathcal{M}(X) \subseteq \mathcal{Q}^{\mathrm{op}}\mathcal{Q}(X)$ is a monotonic neighbourhood function.

The Set-monad arising from the above adjunction is the neighbourhood monad $\mathcal{N} = \mathcal{Q}^{\mathrm{op}}\mathcal{Q}$ (cf. Example 3(2) below) and it will play a central role in what follows.

2.3 Monads

Monads will be used in two different ways. One is related to the view that monads model computational effects [24]. The other is related to their role as abstract algebraic theories [22].

We briefly recall the basic definition. A *monad on* Set is a triple $\mathbb{T} = (T, \eta, \mu)$ where T is an Set-functor, and $\eta\colon \mathrm{Id} \Rightarrow T$ (unit) and $\mu\colon T^2 \Rightarrow T$ (multiplication) are natural transformations that satisfy the following coherence laws: $\mu \circ \eta_T = \mu \circ T\eta = id_T$ and $\mu \circ \mu_T = \mu \circ T\mu$. Due to lack of space, we cannot provide much background on monads. We refer to [21] for the basic definitions of Kleisli category, Eilenberg-Moore algebra (EM-algebra), and monad morphism.

Monads are used to capture computational effects such as I/O and state by viewing (functional) programs as arrows in the Kleisli category [24]. Here, we consider state-based computing rather than functional programming. This means that we generally view programs as functions $X \to TX$ where X is the state-space of the computation. However, the fact that such functions are also Kleisli maps is, of course, essential for the definition of sequential composition. We write $*$ for composition in $\mathcal{K}\ell(T)$.

In order to give semantics to test operations, we need TX to contain an element that represents an aborted computation. We will say that a monad T *is pointed*[1] if for each set X, TX contains a distinguished element $\bot_{TX}$ (or just $\bot$), and for all maps $f\colon X \to Y$, $Tf(\bot) = \bot$.

Example 3. Let X be an arbitrary set. For $U \subseteq X$, we denote by $\uparrow\{U\}$ the up-set of $\{U\}$ in the poset $\mathcal{N}(X)$, i.e., $\uparrow\{U\} = \{N \in \mathcal{N}(X) \mid U \in N\}$.

1. The covariant power set functor $\mathcal{P}$ is a monad with unit $\eta_X(x) = \{x\}$ and multiplication $\mu_X(\{U_i \mid i \in I\}) = \bigcup_{i\in I} U_i$. Arrows in $\mathcal{K}\ell(\mathcal{P})$ are relations, and $*$ is just relation composition. For a set X, $\mathcal{P}(X)$ is the free join-semilattice with bottom on X, and $\mathcal{P}$ is pointed by taking $\bot = \emptyset \in \mathcal{P}(X)$.
2. The neighbourhood functor $\mathcal{N}$ is a monad with

$$\eta_X(x) = \{U \subseteq X \,|\, x \in U\} \qquad \mu_X(W) = \{U \subseteq X \mid \uparrow\{U\} \in W\}.$$

   Also $\mathcal{N}$ is pointed by taking $\bot = \emptyset \in \mathcal{N}(X)$. An arrow $X \to \mathcal{N}Y$ in $\mathcal{K}\ell(\mathcal{N})$ is essentially a predicate transformer $\mathcal{Q}Y \to \mathcal{Q}X$ using the isomorphism via transpose (cf. Remark 2) which translates Kleisli composition of $\mathcal{N}$ into (function) composition of predicate transformers. In particular, for all $\nu_1, \nu_2\colon X \to \mathcal{N}X$, all $x \in X$ and $U \subseteq X$,

$$U \in (\nu_2 * \nu_1)(x) \iff x \in \widehat{\nu_1}(\widehat{\nu_2}(U)) \tag{3}$$

3. The functor $\mathcal{M}$ is also a pointed monad. The unit η and multiplication μ are obtained by restricting the ones for $\mathcal{N}$, and $\bot = \emptyset$. For a set X, $\mathcal{M}(X)$ is the free completely distributive lattice on X, cf. [23] (see also [16, 3.8,4.8]).
4. The functor $L = 1 + \mathrm{Id}$ is the "lift monad" (where $1 = \{*\}$). The unit $\eta_X : X \to 1 + X$ is inclusion. The multiplication μ_X maps $x \in 1 + (1 + X)$ to x iff $x \in X$, and otherwise to $*$. L is pointed by taking $\bot = * \in LX$.

[1] Our notion of pointed monad is equivalent to requiring a natural transformation $1 \Rightarrow T$ where 1 is the constant Set-functor that maps every set to the singleton set.

3 Dynamic Coalgebraic Modal Logic

Our goal is to generalise the situation of PDL and GL to dynamic modal logics for other monads T. For the pointwise operations it seems at first that the natural operations are those coming from EM-algebras of $\mathbb{T}$. For example, PDL *choice* $\cup$ is interpreted via the join-semilattice structure on $\mathcal{P}(X)$. Similarly, the game operations *choice* and *dual* are interpreted via the lattice structure on $\mathcal{M}(X)$. While it is known that all set monads have a presentation in terms of operations and equations (cf. [20]), such a canonical presentation might be a proper class — a property that is not desirable for the design of a clear and concise programming language. As no "small" canonical choice of pointwise operations seems to be given, we generalise pointwise operations such as *choice* and *dual* using the notion of a natural operation and natural algebra.

Definition 4. *Let* $T\colon \mathsf{C} \to \mathsf{Set}$ *be a functor, a* natural n-ary operation on T *is a natural transformation* $\theta\colon T^n \Rightarrow T$. *More generally, given a signature functor* $\Sigma\colon \mathsf{Set} \to \mathsf{Set}$, *a* natural Σ-algebra on T *is a natural transformation* $\theta\colon \Sigma T \Rightarrow T$.

Example 5.
1. All Boolean operations are natural on $\mathcal{Q}^{\mathrm{op}}$. The reason is that the inverse-image map of a function preserves all of those.
2. For similar reasons, all Boolean operations on neighbourhood collections, such as e.g. $N \cup K$ for $N, K \in \mathcal{N}X$, are natural on $\mathcal{N}$. The neighbourhood-wise Boolean operations such as e.g. $N \Cap K = \{U \cap V \mid U \in N, V \in K\}$ are not natural on $\mathcal{N}$.
3. Union and intersection are natural on $\mathcal{M}$ (complement does not preserve monotonicity).
4. The *dual* operation defined for all $N \in \mathcal{N}X$ and $U \subseteq X$ by $U \in N^d$ iff $X \setminus U \notin N$ is natural on $\mathcal{N}$ (and $\mathcal{M}$).
5. The only Boolean operation that is natural on $\mathcal{P}$ is union, because the direct image of a function preserves unions, but not intersections or complements.
6. Apart from identity, the lift monad has only one (rather boring) operation *nil* where $nil_X(t) = *$ for all $t \in LX$.

A natural n-ary operation $\theta\colon T^n \Rightarrow T$ induces for each set X a pointwise operation θ_X^X on $\mathsf{Set}(X, TX) = (TX)^X$ in the expected manner. By cotupling, a natural Σ-algebra $\theta\colon \Sigma T \Rightarrow T$ induces a pointwise Σ-algebra θ_X^X on $(TX)^X$. For n-ary $\sigma \in \Sigma$, we denote the σ-component of θ_X^X by $(\theta_\sigma)_X^X\colon (TX)^n \to TX$.

Just as the syntax and semantics of PDL and GL is defined relative to a particular set of program/game operations, so is our notion of dynamic syntax and semantics. For the syntax, however, one only needs to fix a signature.

Definition 6 (Dynamic syntax). *Given a signature functor* Σ, *a set of atomic actions* A_0 *and a countable set* P *of atomic propositions, we define the set* $\mathcal{F}(P, A_0, \Sigma)$ *of dynamic formulas and the set* $A = A(P, A_0, \Sigma)$ *of complex actions by mutual induction:*

$$\mathcal{F}(P, A_0, \Sigma) \ni \varphi ::= p \in P \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\alpha]\varphi$$
$$A(P, A_0, \Sigma) \ni \alpha ::= a \in A_0 \mid \alpha; \alpha \mid \sigma(\alpha_1, \ldots, \alpha_n) \mid \varphi?$$

where $\sigma \in \Sigma$ *is* n*-ary.*

For the semantics, we first note that the multi-modal structures of PDL and GL easily generalise to a coalgebra $X \to (TX)^A$ for the "labelled functor" T^A. A T^A-coalgebra will be called standard relative to some choice of natural algebra on T.

Definition 7 (Standard). *Let $\theta\colon \Sigma T \Rightarrow T$ be a natural Σ-algebra on a monad $\mathbb{T}$, and let $\delta\colon \Sigma A \to A$ be given by restricting action formation to Σ-operations. A coalgebra $\xi\colon X \to (TX)^A$ is called* θ-standard *if the transpose $\widehat{\xi}\colon A \to (TX)^X$ is a Σ-algebra morphism, i.e.,*

$$\widehat{\xi} \circ \delta = \theta_X^X \circ \Sigma\widehat{\xi} \tag{4}$$

We say that ξ is ;-standard *if for all $\alpha, \beta \in A$, $\widehat{\xi}(\alpha;\beta) = \widehat{\xi}(\alpha) * \widehat{\xi}(\beta)$.*

We now define the notion of a dynamic model relative to a choice of natural algebra $\theta\colon \Sigma T \Rightarrow T$.

Definition 8 (Dynamic semantics). *Let $\mathbb{T} = (T, \eta, \mu)$ be a pointed monad, and $\theta\colon \Sigma T \Rightarrow T$ a natural Σ-algebra on T. A (P, A_0, θ)-*dynamic $\mathbb{T}$-model *is a triple $\mathfrak{M} = (\xi_0, \lambda, V)$ where $\widehat{\xi_0}\colon A_0 \to (TX)^X$ is an interpretation of atomic actions in $(TX)^X$, $\lambda\colon \mathcal{Q} \Rightarrow \mathcal{Q} \circ T$ is a unary predicate lifting for T, and $V\colon P \to \mathcal{P}(X)$ is a valuation. We define the truth set $[\![\varphi]\!]^{\mathfrak{M}}$ of dynamic formulas and the semantics $\widehat{\xi}\colon A \to (TX)^X$ of complex actions in $\mathfrak{M}$ by mutual induction:*

$$\begin{array}{lll}
[\![p]\!]^{\mathfrak{M}} = V(p), & [\![\varphi \wedge \psi]\!]^{\mathfrak{M}} = [\![\varphi]\!]^{\mathfrak{M}} \cap [\![\psi]\!]^{\mathfrak{M}}, & [\![\neg\varphi]\!]^{\mathfrak{M}} = X \setminus [\![\varphi]\!]^{\mathfrak{M}}, \\
[\![[\alpha]\varphi]\!]^{\mathfrak{M}} & = (\widehat{\xi}(\alpha)^{-1} \circ \lambda_X)([\![\varphi]\!]^{\mathfrak{M}}), & \\
\widehat{\xi}(\sigma(\alpha_1, \ldots, \sigma_n)) & = (\theta_\sigma)_X^X(\widehat{\xi}(\alpha_1), \ldots, \widehat{\xi}(\alpha_n)) & \text{where } \sigma \in \Sigma \text{ is } n\text{-ary}, \\
\widehat{\xi}(\alpha;\beta) & = \widehat{\xi}(\alpha) * \widehat{\xi}(\beta) & \text{(Kleisli composition)}, \\
\widehat{\xi}(\varphi?)(x) & = \eta_X(x) \text{ if } x \in [\![\varphi]\!]^{\mathfrak{M}}, \ \bot_{TX} \text{ otherwise.} &
\end{array}$$

We will sometimes refer to the induced $\widehat{\xi}$, and its transpose $\xi\colon X \to (TX)^A$, simply as a θ-dynamic $\mathbb{T}$-model.

Note that, by definition, a θ-dynamic $\mathbb{T}$-model $\xi\colon X \to (TX)^A$ is both θ-standard and ;-standard.

Remark 9. If we would not include tests, then we could drop the requirement of $\mathbb{T}$ being pointed, and define a $\mathbb{T}$-dynamic (P, A_0, θ)-structure to be a coalgebra $\xi\colon X \to (TX)^A$ whose transpose is the unique $\Sigma \cup \{;\}$-algebra morphism induced by $\widehat{\xi_0}\colon A_0 \to (TX)^X$ and the $\Sigma \cup \{;\}$-algebra structure on $(TX)^X$ given by θ and Kleisli composition.

4 Soundness

In this section we give a general method for finding axioms for (P, A_0, θ)-dynamic $\mathbb{T}$-models. In order for these axioms to be sound, it will be necessary to require

the predicate lifting $\lambda\colon \mathcal{Q} \Rightarrow \mathcal{Q} \circ T$ to interact well with monad structure and pointwise structure.

We start with sequential composition. Not surprisingly, ;-standard models are captured by the axiom $[\alpha;\beta]p \leftrightarrow [\alpha][\beta]p$, for all $\alpha, \beta \in A$.

Lemma 10. *Let $\xi\colon X \to (TX)^A$ be ;-standard. If $\widehat{\lambda}\colon T \to \mathcal{Q}^{\mathrm{op}}\mathcal{Q}$ is a monad morphism, then the axiom $[\alpha;\beta]p \leftrightarrow [\alpha][\beta]p$ is valid in $\mathfrak{M}$.*

Remark 11. As noted in e.g. [17], giving a monad morphism $\mathbb{T} \Rightarrow \mathcal{Q}^{\mathrm{op}}\mathcal{Q}$ is the same as giving an Eilenberg-Moore algebra $T2 \to 2$. The view of modalities as EM-algebras for $\mathbb{T}$ was already suggested in [24], and more recently in [13]. The correspondence (via the Yoneda lemma) between unary predicate liftings and subsets of $T2$ was observed in [29]. Moreover, it is easy to verify that λ corresponds to an EM-algebra iff its Boolean dual $\neg\lambda\neg$ does.

Example 12. (i) The Kripke diamond $\lambda^{\Diamond}_X(U) = \{V \in \mathcal{P}(X) \mid U \cap V \neq \emptyset\}$ corresponds (via Yoneda) to the free $\mathcal{P}$-algebra $\mathcal{P}\mathcal{P}(1) \to \mathcal{P}(1)$, hence the transpose of $\lambda^{\Diamond}$ and of its dual, the Kripke box $\lambda^{\Box}$, are both monad morphisms. (ii) The transpose of the monotonic λ is the natural inclusion $\widehat{\lambda}\colon \mathcal{M} \Rightarrow \mathcal{N}$ and hence a monad morphism. (iii) In [13], the EM-algebras $L2 \to 2$ for the lift monad were shown to correspond to λ^{tl} (total correctness) and λ^{pl} (partial correctness) where $t \in \lambda^{\mathsf{tl}}_X(U)$ iff $t \in U$ and $t \in \lambda^{\mathsf{pl}}_X(U)$ iff $t = *$ or $t \in U$.

Finding axioms for pointwise operations from natural algebras requires a bit more work. We will use the observation that an operation $\sigma\colon (\mathcal{N}X)^n \to \mathcal{N}X$ on neighbourhood functions is isomorphic to an operation $\overline{\sigma} = \psi^{-1} \circ \sigma \circ \psi^n$ on predicate transformers via the bijection $\psi\colon \mathcal{Q}X^{\mathcal{Q}X} \to (\mathcal{Q}^{\mathrm{op}}\mathcal{Q}X)^X$ given in Remark 2. In particular, if $\chi\colon \mathcal{N}^n \Rightarrow \mathcal{N}$ is a natural operation on $\mathcal{N}$ with pointwise lifting $\chi^X_X\colon ((\mathcal{N}X)^X)^n \Rightarrow (\mathcal{N}X)^X$ to neighbourhood functions for any set X, then $\overline{\chi}^X_X = \psi^{-1} \circ \chi^X_X \circ \psi^n$ is concretely given by

$$x \in \overline{\chi}^X_X(m_1, \ldots, m_n)(U) \iff U \in \chi^X_X(\psi(m_1), \ldots, \psi(m_n))(x) \tag{5}$$

for all $m_1, \ldots, m_n \in \mathcal{Q}X^{\mathcal{Q}X}$, $x \in X$ and $U \subseteq X$.

Example 13. The operation on predicate transformers corresponding to the *dual* operation $d\colon \mathcal{N} \Rightarrow \mathcal{N}$ is $\overline{d}(m)(U) = X \setminus m(X \setminus U)$. The operations on predicate transformers corresponding to Boolean operations on $\mathcal{N}$ are $(m_1 \overline{\cup} m_2)(U) = m_1(U) \cup m_2(U)$, $(\overline{\neg} m)(U) = X \setminus m(U)$ and so on.

The axioms for pointwise operations turn operations on labels into operations on predicate transformers. Using the above correspondence, we find the axioms via representations of natural operations on $\mathcal{N}$. For all $\chi\colon \mathcal{N}^n \Rightarrow \mathcal{N}$ and all $\alpha_1, \ldots, \alpha_n \in A$, we will define a rank-1 formula $\varphi(\chi, , \alpha_1, \ldots, \alpha_n, p)$. We start by showing how to do so for unary operations. Let $\chi\colon \mathcal{N} \Rightarrow \mathcal{N}$ be a unary natural operation on $\mathcal{N} = \mathcal{Q}^{\mathrm{op}}\mathcal{Q}$. We have the following correspondence via the adjunction $\mathcal{Q} \dashv \mathcal{Q}^{\mathrm{op}}$ from Remark 2:

$$\frac{\chi_X\colon \mathcal{Q}^{\mathrm{op}}\mathcal{Q}X \to \mathcal{Q}^{\mathrm{op}}\mathcal{Q}X \in \mathsf{Set}}{\widehat{\chi}_X\colon \mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}X \to \mathcal{Q}X \in \mathsf{Set}^{\mathrm{op}}}$$

Therefore χ corresponds uniquely to (a predicate lifting) $\widehat{\chi} = \lambda^{\chi} \colon \mathcal{Q} \Rightarrow \mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}$, and by the Yoneda lemma to an element $\breve{\chi} = \widehat{\chi}_2(id_2) = \widehat{\chi}_2(\{1\}) \in \mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}(2)$. Note that $\mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}(2)$ is the free Boolean algebra on four generators that can be identified with the elements of $\mathcal{Q}(2) = \{\emptyset, \{0\}, \{1\}, 2\}$. Consider the following four natural operations on $\mathcal{Q}^{\mathrm{op}}\mathcal{Q}$ and their Yoneda correspondents:

operation $\chi_X \colon \mathcal{N}X \to \mathcal{N}X, N \in \mathcal{N}X, U \subseteq X$		$\breve{\chi} \colon \mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}(2), N \in \mathcal{Q}^{\mathrm{op}}\mathcal{Q}(2)$
id	$U \in i_X(N) \iff U \in N$	$N \in \breve{\imath} \iff \{1\} \in N$
compl.	$U \in c_X(N) \iff X \setminus U \in N$	$N \in \breve{c} \iff \{0\} \in N$
zero	$U \in z_X(N) \iff \emptyset \in N$	$N \in \breve{z} \iff \emptyset \in N$
top	$U \in t_X(N) \iff X \in N$	$N \in \breve{t} \iff 2 \in N$

Since $\breve{\imath}, \breve{c}, \breve{z}, \breve{t}$ generate all of $\mathcal{Q}\mathcal{Q}^{\mathrm{op}}\mathcal{Q}(2)$ it follows that for every unary natural operation $\chi \colon \mathcal{Q}^{\mathrm{op}}\mathcal{Q} \Rightarrow \mathcal{Q}^{\mathrm{op}}\mathcal{Q}$ the correspondent $\breve{\chi}$ is a Boolean combination over $\breve{\imath}, \breve{c}, \breve{z}, \breve{t}$.

For an n-ary $\chi \colon \mathcal{N}^n \Rightarrow \mathcal{N}$, we get as Yoneda correspondent $\breve{\chi} \in \mathcal{Q}(\mathcal{N}(2)^n) \cong \mathcal{N}(n \cdot \mathcal{Q}(2))$ where $n \cdot \mathcal{Q}(2)$ is the n-fold coproduct of $\mathcal{Q}(2)$. $\mathcal{N}(n \cdot \mathcal{Q}(2))$ is the free Boolean algebra over $n \cdot \mathcal{Q}(2)$, and hence any n-ary natural operation on $\mathcal{N}$ corresponds to a Boolean expression over n copies of the generators $\breve{\imath}, \breve{c}, \breve{z}, \breve{t}$. For example, the binary union $\chi = \cup$ has correspondent $\breve{\chi} = \breve{\imath}_1 \vee \breve{\imath}_2$, i.e., $(N, K) \in \breve{\chi}$ iff $N \in \breve{\imath}_1$ or $K \in \breve{\imath}_2$. This leads to the following definition.

Definition 14. *Let $\{\breve{\imath}_j, \breve{c}_j, \breve{z}_j, \breve{t}_j \mid j = 1, \dots, n\}$ be the generators of $\mathcal{N}(n \cdot \mathcal{Q}(2))$. For $\chi \colon \mathcal{N}^n \Rightarrow \mathcal{N}$ we define $\varphi(\breve{\chi}, \alpha_1, \dots, \alpha_n, p)$ inductively as follows:*

- $\varphi(\breve{\imath}_j, \alpha_1, \dots, \alpha_n, p) = [\alpha_j]p$ *for all* $j = 1, \dots, n$.
- $\varphi(\breve{c}_j, \alpha_1, \dots, \alpha_n, p) = [\alpha_j]\neg p$ *for all* $j = 1, \dots, n$.
- $\varphi(\breve{z}_j, \alpha_1, \dots, \alpha_n, p) = [\alpha_j]\bot$ *for all* $j = 1, \dots, n$.
- $\varphi(\breve{t}_j, \alpha_1, \dots, \alpha_n, p) = [\alpha_j]\top$ *for all* $j = 1, \dots, n$.
- $\varphi(\neg\breve{\chi}, \alpha_1, \dots, \alpha_n, p) = \neg\varphi(\chi, \alpha_1, \dots, \alpha_n, p)$.
- $\varphi(\breve{\chi} \wedge \breve{\delta}, \alpha_1, \dots, \alpha_n, p) = \varphi(\chi, \alpha_1, \dots, \alpha_n, p) \wedge \varphi(\delta, \alpha_1, \dots, \alpha_n, p)$.

For example, the dual operation on $\mathcal{N}$ is $\breve{d} = \neg\breve{c}$ and we have $\varphi(\breve{d}, \alpha, p) = \neg\varphi(\breve{c}, \alpha, p) = \neg[\alpha]\neg p$. Similarly, for $\breve{d}_1 \wedge \neg\breve{t}_2$, we get $\varphi(\breve{d}_1 \wedge \neg\breve{t}_2, \alpha_1, \alpha_2, p) = (\neg[\alpha_1]\neg p) \wedge \neg([\alpha_2]\top)$.

The following theorem says that whenever λ transforms θ-structure on T-coalgebras into χ-structure on neighbourhood functions, for some natural χ, then the rank-1 axioms associated with χ are sound on θ-standard coalgebras.

Theorem 15. *Let $\theta \colon T^n \Rightarrow T$ be a natural operation on T, and let $\xi \colon X \to (TX)^A$ be a θ-standard T^A-coalgebra. Let $\lambda \colon \mathcal{Q} \Rightarrow \mathcal{Q} \circ T$ be a predicate lifting for T. If there is a natural operation $\chi \colon \mathcal{N}^n \Rightarrow \mathcal{N}$ such that*

$$\widehat{\lambda} \circ \theta = \chi \circ \widehat{\lambda}^n \tag{6}$$

then for all $\alpha_1, \dots, \alpha_n \in A$, the θ-axiom $[\underline{\theta}(\alpha_1, \dots, \alpha_n)]p \leftrightarrow \varphi(\breve{\chi}, \alpha_1, \dots, \alpha_n, p)$ is valid in ξ (where $\underline{\theta}$ denotes the syntax/term constructor associated with θ). The above statement generalises to natural Σ-algebras $\theta \colon \Sigma T \Rightarrow T$ by considering the axioms for the components θ_σ and χ_σ for $\sigma \in \Sigma$.

Example 16. Using Theorem 15 we find that

(i) The PDL axiom $[\alpha \cup \beta]p \leftrightarrow [\alpha]p \wedge [\beta]p$ is valid because $\widehat{\lambda^{\Box}} \colon \mathcal{P} \Rightarrow \mathcal{N}$ transforms unions into intersections, i.e., $\widehat{\lambda}^{\Box}_X(U_1 \cup U_2) = \widehat{\lambda}^{\Box}_X(U_1) \cap \widehat{\lambda}^{\Box}_X(U_2)$. That is, we can apply Theorem 15 with $\theta = \cup \colon \mathcal{P}^2 \Rightarrow \mathcal{P}$ and $\chi = \cap \colon \mathcal{N}^2 \Rightarrow \mathcal{N}$.

(ii) The axiom $[\alpha \cup \beta]p \leftrightarrow [\alpha]p \vee [\beta]p$ is valid in standard GL-models because the transpose of the predicate lifting $\lambda_X(U) = \{N \in \mathcal{M}X \mid U \in N\}$ for the monotonic modality is the natural inclusion $\widehat{\lambda} \colon \mathcal{M} \Rightarrow \mathcal{N}$, i.e., $\theta = \chi = \cup$. Similarly, for the dual axiom $[\alpha^d]p \leftrightarrow \neg[\alpha]\neg p$.

(iii) For the lift monad we find that λ^{tl} turns *nil* into χ_{nil} where $\chi_{nil,X}(N) = \emptyset$ for all $N \in \mathcal{N}X$. Hence, we have the axiom $[nil]p \leftrightarrow \bot$. Dually, λ^{pl} turns *nil* into χ_{all} where $\chi_{all,X}(N) = \mathcal{P}(X)$ and we get the axiom $[nil]p \leftrightarrow \top$.

5 Completeness

In this section we will prove a generic strong completeness result for our family of coalgebraic dynamic logics.

Our completeness proof makes use of results from coalgebraic modal logic. Therefore we need to recall some terminology: A modal logic $\mathcal{L} = (\Lambda, \mathrm{Ax}, \mathrm{Fr})$ consists of a modal signature Λ, a collection $\mathrm{Ax} \subseteq \mathsf{Prop}(\Lambda(\mathsf{Prop}(P)))$ of rank-1 axioms, and a collection $\mathrm{Fr} \subseteq \mathcal{F}(\Lambda)$ of frame conditions. For a formula $\varphi \in \mathcal{F}(\Lambda)$, we write $\vdash_{\mathcal{L}} \varphi$ if φ can be derived from $\mathrm{Ax} \cup \mathrm{Fr}$ using propositional reasoning, uniform substitution and the congruence rule: from $\varphi_1 \leftrightarrow \varphi_1, \ldots, \varphi_n \leftrightarrow \psi_n$ infer $\Box_\lambda(\varphi_1, \ldots, \varphi_n) \leftrightarrow \Box_\lambda(\psi_1, \ldots, \psi_n)$ for any n-ary $\Box_\lambda \in \Lambda$. A formula $\varphi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is *one-step derivable*, denoted $\vdash^1_{\mathcal{L}}$, if φ is propositionally entailed by the set $\{\psi\tau \mid \tau : P \to \mathcal{P}(X), \psi \in \mathrm{Ax}\}$. A set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is called *one-step $\mathcal{L}$-consistent* if there are no formulas $\varphi_1, \ldots, \varphi_n \in \Phi$ such that $\vdash^1_{\mathcal{L}} \neg(\varphi_1 \wedge \cdots \wedge \varphi_n)$. Let T be a Set-functor and assume a predicate lifting λ is given for each $\Box_\lambda \in \Lambda$. For a formula $\varphi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ the *one-step semantics* $[\![\varphi]\!]_1 \subseteq TX$ is defined by putting $[\![\Box_\lambda(U_1, \ldots, U_n)]\!]_1 = \lambda_X(U_1, \ldots, U_n)$ and by inductively extending this definition to Boolean combinations of boxed formulas. We say that $\mathcal{L}$ is *separating* if $t \in TX$ is uniquely determined by the set $\{\Phi \in \Lambda(\mathcal{P}(X)) \mid t \in [\![\Phi]\!]_1\}$. Finally, $\mathcal{L}$ is called *one-step sound* if for any one-step derivable formula $\varphi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ we have $[\![\varphi]\!]_1 = TX$, i.e., if any such formula φ is *one-step valid*. $\mathcal{L}$ is called *strongly one-step complete* over finite sets if for every finite set X and every one-step consistent set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is one-step satisfiable.

Throughout the section we assume the following are given: a pointed monad $\mathbb{T}$ on Set, a single, unary predicate lifting $\lambda \colon \mathcal{Q} \Rightarrow \mathcal{Q} \circ T$ for T whose transpose $\widehat{\lambda}$ is a monad morphism, a countable set P of atomic propositions, a set A_0 of atomic actions, and $\theta \colon \Sigma T \Rightarrow T$, a natural Σ-algebra on T. To ensure soundness, we also assume that there is a natural algebra $\chi \colon \Sigma\mathcal{N} \Rightarrow \mathcal{N}$ such that $\widehat{\lambda} \circ \theta = \chi \circ \widehat{\lambda}^n$ (cf. Theorem 15). We let $\Lambda = \{[\alpha] \mid \alpha \in A(P, A_0, \Sigma)\}$.

Let us now clarify which logics we are considering. Firstly, we assume we have a separating, one-step sound and strongly one-step complete rank-1 axiomatisation $\mathrm{Ax}(T, \Box)$ over T-coalgebras in the basic modal language $\mathcal{F}(\{\Box\})$.

The "underlying" logic $(\{\lambda\}, \mathrm{Ax}(T, \Box), \emptyset)$ will be denoted by $\mathcal{L}_b$. Given an action $\alpha \in A$, we denote by $\mathrm{Ax}(T, \Box)_\alpha$ the set of rank-1 axioms over the labelled modal language $\mathcal{F}(\{[\alpha] \mid \alpha \in A\})$ obtained by replacing all ocurrences of $\Box$ by $[\alpha]$, and we let $\mathrm{Ax}(T, \Box)_A = \bigcup_{\alpha \in A} \mathrm{Ax}(T, \Box)_\alpha$ be all labelled instances of rank-1 axioms in $\mathrm{Ax}(T, \Box)$.

Definition 17 (Dynamic logic). *We define*

$$\begin{aligned} \mathrm{Ax} &= \mathrm{Ax}(T, \Box)_A \cup \{\varphi(\sigma, \alpha_1, \ldots, \alpha_n, p) \mid \sigma \in \Sigma, \alpha_i \in A\} \\ \mathrm{Fr} &= \{[\alpha; \beta]p \leftrightarrow [\alpha][\beta]p \mid \alpha, \beta \in A(P, A_0, \Sigma), p \in P\} \\ \mathcal{L}(\theta) &= (\Lambda, \mathrm{Ax}, \emptyset), \\ \mathcal{L}(\theta, ;) &= (\Lambda, \mathrm{Ax}, \mathrm{Fr}). \end{aligned}$$

We refer to $\mathcal{L}(\theta)$ and $\mathcal{L}(\theta, ;)$ as (P, A_0, θ)-dynamic logics.

We are now going to prove completeness of both $\mathcal{L}(\theta)$ and $\mathcal{L}(\theta, ;)$ with respect to θ-standard and $\theta, ;$-standard models, respectively. In order to facilitate our proof we show that θ-standard models can be characterised as those models that are based on T^A_{st}-coalgebras for a suitable subfunctor T^A_{st} of T^A. This is done using the following definition of θ-standard that can be seen as a "point-wise" version of Definition 7.

Definition 18. *We say a function $f : A \to TX$ is θ-standard if f is a Σ-algebra morphism (from $\delta \colon \Sigma A \to A$ to $\theta_X \colon \Sigma TX \to TX$ where δ is as in Def. 7):*

$$f \circ \delta = \theta_X \circ \Sigma f \tag{7}$$

Furthermore we let $T^A_{\mathrm{st}} X = \{f : A \to TX \mid f$ is θ-standard$\}$. It is easy to check that T^A_{st} can be extended to a subfunctor of T^A.

Lemma 19. *A coalgebra $\xi : X \to (TX)^A$ is θ-standard iff ξ is a T^A_{st}-coalgebra.*

Let us now start with our completeness proof. We are first going to check that $\mathcal{L}(\theta)$ is *one-step* sound over θ-standard models.

Proposition 20. *The logic $\mathcal{L}(\theta)$ is one-step sound for T^A_{st}.*

Proposition 21. *The logic $\mathcal{L}(\theta)$ is strongly one-step complete for T^A_{st}.*

The property of a functor preserving inverse limits of surjective ω-cochains is from [31] is one of the main conditions for the existence of quasi-canonical models in Proposition 24 below.[2] Please consult *loc.cit.* for the definition.

Proposition 22. *If T weakly preserves inverse limits of surjective ω-cochains, then so does the functor T^A_{st}.*

Using the results from [31], Propositions 21 and 22 imply that $\mathcal{L}(\theta)$ is sound and strongly complete with respect to T^A_{st}-coalgebras. This is achieved by proving the existence of so-called quasi-canonical models.

[2] The condition in [31] is motivated by a stronger condition used in [19, Thm. 9.4].

Definition 23. *A quasi-canoncial T^A_{st}-model for a dynamic modal logic $\mathcal{L} = (\Lambda, \mathrm{Ax}, \mathrm{Fr})$ is a T^A_{st}-model $(S, \xi : S \to T^A_{\mathrm{st}} S, V)$ that satisfies all axioms* Ax *and frame conditions* Fr *and such that*

- *S is the set of maximal $\mathcal{L}$-consistent sets of formulas,*
- *$V(p) = \{\Delta \in S \mid p \in \Delta\}$ and*
- *for all $\Gamma \in S$, $\alpha \in A$ and all formulas φ we have:*

$$\xi(\Gamma)(\alpha) \in \lambda(\hat{\varphi}) \qquad \textit{iff} \qquad [\alpha]\varphi \in \Gamma, \ \textit{where} \qquad \hat{\varphi} = \{\Gamma \in S \mid \varphi \in \Gamma\}.$$

Proposition 24. *The logic $\mathcal{L}(\theta)$ has a quasi-canonical model. Consequently, $\mathcal{L}(\theta)$ is sound and strongly complete with respect to the class of all θ-standard models.*

Next, we prove that $\mathcal{L}(\theta, ;)$ is complete with respect to all $\theta, ;$-standard dynamic models. Therefore we need to ensure that the frame is well-behaved regarding action composition. (Tests will be discussed in section 6.) In other words, we want to ensure the *validity* of the sequential composition axioms on the quasi-canonical frame. From a coalgebraic perspective this is a non-trivial task as one cannot deal with axioms of rank greater than 1 in a generic coalgebraic way. In particular, we cannot assume that a quasi-canonical model is ;-standard, but we now describe how we can modify a quasi-canonical model into an equivalent ;-standard quasi-canonical model.

Definition 25. *Let (X, ξ, V) be a T^A_{st}-model. We say two elements t and t' of TX are equivalent with respect to boxed atoms (notation: $t \sim_\Box t'$) if for all formulas φ we have*

$$t \in \lambda_X\left([\![\varphi]\!]_{(X,\xi,V)}\right) \qquad \textit{iff} \qquad t' \in \lambda_X\left([\![\varphi]\!]_{(X,\xi,V)}\right).$$

The next lemma can easily be proven by structual induction on the formula.

Lemma 26. *Let (X, ξ, V) be a T^A_{st}-model and let $\sim_\Box$ be its associated boxed atom equivalence. If $\xi' : X \to T^A_{\mathrm{st}} X$ is a coalgebra structure such that for all $x \in X$ and all $\alpha \in A$ we have $\xi(x)(\alpha) \sim_\Box \xi'(x)(\alpha)$, then for all formulas φ we have $[\![\varphi]\!]_{(X,\xi,V)} = [\![\varphi]\!]_{(X,\xi',V)}$.*

We are ready to prove the main result of this section.

Theorem 27. *The logic $\mathcal{L}(\theta, ;)$ is sound and strongly complete with respect to all T^A_{st}-models that are based on a ;-standard T^A_{st}-coalgebra.*

6 Tests

We will now incorporate axioms for tests into our axiomatisation of $\mathcal{L}(\theta, ;)$ and prove soundness and completeness with respect to dynamic models.

When choosing the axioms for tests there are two obvious choices, depending on our choice of underlying modality. This can be best seen at the example

$T = \mathcal{P}$: Taking the Kripke $\Diamond$ as basic modality, the axiom for tests will be $[\psi?]\varphi \leftrightarrow \psi \wedge \varphi$. Taking the Kripke $\Box$, the axiom for tests will be $[\psi?]\varphi \leftrightarrow \psi \to \varphi$.

In order to obtain an axiomatisation that is generic in the functor and chosen modality, we need a definition for when a modal operator is "box-like" or "diamond"-like. Apart from Def. 8 (semantics of tests), this is the only time we need that the monad is pointed, cf. Remark 9.

Definition 28. *Let λ be a predicate lifting for a pointed set monad $\mathbb{T}$. We say λ is "box-like" if for all sets X and all $U \subseteq X$ we have that the distinguished element $\bot \in TX$ is in the λ-lifting of U, i.e., $\bot \in \lambda_X(U)$. Likewise we call λ "diamond-like" if for all sets X and all $U \subseteq X$ we have $\bot \notin \lambda_X(U)$.*

Any modality for a pointed monad falls into one of the above categories: For example, the (monotonic) neighbourhood modality is diamond-like.

Lemma 29. *Let λ be a predicate lifting for a pointed set monad $\mathbb{T}$. Then λ is either box-like or diamond-like.*

This allows us to add test axioms to $\mathcal{L}(\theta, ;)$.

Definition 30. *If λ is box-like, then we define the dynamic logic $\mathcal{L}(\theta, ;, ?)$ by adding the frame condition $[?\psi]p \leftrightarrow (\psi \to p)$ to* Fr *in $\mathcal{L}(\theta, ;)$. If λ is diamond-like, then we define $\mathcal{L}(\theta, ;, ?)$ by adding the frame condition $[?\psi]p \leftrightarrow (\psi \wedge p)$ to* Fr *in $\mathcal{L}(\theta, ;)$.*

Our soundness and completeness results relative to $\theta, ;$-regular models can now be extended to $\mathcal{L}(\theta, ;, ?)$ relative to the dynamic semantics.

Theorem 31. *The logic $\mathcal{L}(\theta, ;, ?)$ is sound and strongly complete with respect to the dynamic semantics (cf. Def. 8).*

As special instances we obtain the following results (of which (i) and (ii) were already known, but to our knowledge item (iii) is a modest new addition).

Corollary 32. *(i) Iteration-free PDL is sound and strongly complete with respect to $\cup$-dynamic $\mathcal{P}$-models. (ii) Iteration-free Game Logic is sound and strongly complete with respect to $\cup, ^d$-dynamic $\mathcal{M}$-models. (iii) Let $\mathcal{L}_l = (\lambda^{\mathsf{tl}}, \mathrm{Ax}, \emptyset)$ be the "underlying logic" for the lift monad L where $\mathrm{Ax} = \{\Box(p \wedge q) \leftrightarrow \Box p \wedge \Box q, \Box(\neg p) \leftrightarrow \Box\top \wedge \neg\Box p\}$. Then the dynamic logic $\mathcal{L}_l(\emptyset, ;, ?)$ (over $\mathcal{L}_l$) is sound and strongly complete with respect to dynamic L-models.*

7 Discussion and Conclusion

We have presented a framework for iteration-free coalgebraic dynamic modal logic where programs are modelled as T-coalgebras for a monad T, and program constructs are modelled via natural operations on T. We have proved a generic strong completeness result relative to a chosen set θ of natural operations. We note that our notion of natural operation is more general than the notion of

algebraic operation [27] which is used in the context of computational effects. For example, it can be checked that *dual* is not an algebraic operation for $\mathcal{M}$. We also note that the fact that intersection is not natural on $\mathcal{P}$ can be seen as an explanation of why PDL with intersection is difficult to axiomatise [1].

We leave it as future work to incorporate iteration into our framwork. From PDL we know that dynamic modal logics with iteration cannot be strongly complete (due to non-compactness). Moreover, the fact that the completeness of GL remains an open problem tells us that a general weak completeness theorem is highly non-trivial. In any case, we will need to assume that the monad in question is order-enriched, perhaps along the lines of [13,6].

We note that our notion of pointed monad is weaker than requiring that the Kleisli category is enriched over the categoy of pointed sets, or over pointed CPOs. For example, it can be checked that the Kleisli category of the pointed monad $\mathcal{M}$ has neither form of enrichment.

A limitation of our framework is that it is unsuitable for designing dynamic modal logics for probabilistic or weighted systems. For probabilistic systems that are coalgebras for the distribution monad $\mathcal{D}_\omega$, there is no monad morphism $\mathcal{D}_\omega \Rightarrow \mathcal{N}$, since there is no EM-algebra $\mathcal{D}_\omega 2 \to 2$, as $2 = \{0, 1\}$ is not closed under convex sums. Similarly, for the weighted semiring monad $S_\omega(X) = \{f \colon X \to S \mid f \text{ has finite support } \}$ (where S is a semiring), 2 is not closed under S-linear combinations if e.g., $S = \mathbb{N}$. Dynamic logics for such quantitative systems seem to require a multi-valued setting where the truth object is $T(1)$ (instead of 2).

Such a multi-valued approach to weakest preconditions for non-deterministic, probabilistic and quantum computation has recently been investigated in a categorical setting via so-called state-and-effect-triangles [14,15], see also [13,6]. Weakest preconditions are closely related to dynamic modal logic, e.g., the weakest precondition for φ with respect to program α is expressed in PDL as $[\alpha]\varphi$. Also in [13,15], as in our Lemma 10, it is noted that weakest preconditions/predicate liftings must be monad morphisms in order to obtain compositionality for sequential composition. An important difference with our work is that [13,15] focus on semantics, and no syntax or axiomatisation is investigated. We would like to investigate further the connections between our work and the multi-valued predicate transformer approach of [13,15].

Acknowledgements. We thank Bart Jacobs, Alexander Kurz and Yde Venema for helpful discussions, and the anonymous referees for their useful comments.

References

1. Balbiani, P., Vakarelov, D.: Iteration-free PDL with intersection: a complete axiomatisation. Fundamenta Informaticae 45, 173–194 (2001)
2. Berwanger, D.: Game Logic is strong enough for parity games. Studia Logica 75(2), 205–219 (2003)
3. Chellas, B.F.: Modal Logic - An Introduction. Cambridge University Press (1980)
4. Cirstea, C., Kurz, A., Pattinson, D., Schröder, L., Venema, Y.: Modal logics are coalgebraic. In: Abramsky, S. (ed.) Visions of Computer Science 2008 (2008)

5. Fischer, M., Ladner, R.: Propositional dynamic logic of regular programs. J. of Computer and System Sciences 18, 194–211 (1979)
6. Goncharov, S., Schröder, L.: A relatively complete generic Hoare logic for order-enriched effects. In: Proceedings of LICS 2013, pp. 273–282. IEEE (2013)
7. Gumm, H.P.: Universal Coalgebras and their Logics (2009)
8. Hansen, H.H.: Monotonic modal logic (Master's thesis). Research Report PP-2003-24, Inst. for Logic, Language and Computation. University of Amsterdam (2003)
9. Hansen, H.H., Kupke, C.: A coalgebraic perspective on monotone modal logic. In: Proceedings of CMCS 2004. ENTCS, vol. 106, pp. 121–143. Elsevier (2004)
10. Hansen, H., Kupke, C., Leal, R.: Strong completeness for iteration-free coalgebraic dynamic logics. Tech. rep., ICIS, Radboud University Nijmegen (to appear, 2014), `https://pms.cs.ru.nl/iris-diglib/src/icis_tech_reports.php`
11. Hansen, H., Kupke, C., Pacuit, E.: Neighbourhood structures: bisimilarity and basic model theory. Logical Meth. in Comp. Sci. 5(2:2) (2009)
12. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. The MIT Press (2000)
13. Hasuo, I.: Generic weakest precondition semantics from monads enriched with order. In: Proceedings of CMCS 2014. LNCS, vol. 8446. Springer (2014)
14. Jacobs, B.: New directions in categorical logic, for classical, probabilistic and quantum logic. In: Logical Meth. in Comp. Sci. (to appear, 2014)
15. Jaocbs, B.: Dijkstra monads in monadic computation. In: Bonsangue, M. (ed.) Proceedings of CMCS 2014. LNCS, vol. 8446, Springer (2014)
16. Johnstone, P.: Stone Spaces. Cambridge University Press (1982)
17. Kelly, G.M., Power, A.J.: Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. J. Pure and Appl. Alg. 89, 163–179 (1993)
18. Kupke, C., Pattinson, D.: Coalgebraic semantics of modal logics: an overview. Theor. Comp. Sci. 412(38), 5070–5094 (2011)
19. Kurz, A., Rosický, J.: Strongly complete logics for coalgebras. Logical Meth. Logical Meth. In: Comp. Sci. 8(3:14) (2012)
20. Linton, F.: Some aspects of equational categories. In: Eilenberg, S., Harrison, D., MacLane, S., Röhrl, H. (eds.) Proceedings of the Conference on Categorical Algebra, pp. 84–94. Springer (1966)
21. MacLane, S.: Categories for the Working Mathematician, 2nd edn. Springer (1998)
22. Manes, E.G.: Algebraic Theories. Springer, Berlin (1976)
23. Markowsky, G.: Free completely distributive lattices. Proc. Amer. Math. Soc. 74(2), 227–228 (1979)
24. Moggi, E.: Notions of computation and monads. Information and Computation 93(1) (1991)
25. Parikh, R.: The logic of games and its applications. In: Topics in the Theory of Computation. Annals of Discrete Mathematics, vol. 14. Elsevier (1985)
26. Pauly, M., Parikh, R.: Game Logic: An overview. Studia Logica 75(2), 165–182 (2003)
27. Plotkin, G.D., Power, A.J.: Semantics for algebraic operations. In: Proceedings of MFPS XVII. ENTCS, vol. 45 (2001)
28. Rutten, J.: Universal coalgebra: A theory of systems. Theor. Comp. Sci. 249, 3–80 (2000)
29. Schröder, L.: Expressivity of coalgebraic modal logic: The limits and beyond. Theor. Comp. Sci. 390, 230–247 (2008)
30. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. ACM Transactions on Computational Logics 10(2:13), 1–33 (2009)
31. Schröder, L., Pattinson, D.: Strong completeness of coalgebraic modal logics. In: Proceedings of STACS 2009, pp. 673–684 (2009)

Quantitative Types for the Linear Substitution Calculus

Delia Kesner[1] and Daniel Ventura[2]

[1] Univ. Paris-Diderot, SPC, PPS, CNRS, France
[2] Univ. Federal de Goiás, INF, Brasil

Abstract. We define two non-idempotent intersection type systems for the linear substitution calculus, a calculus with partial substitutions acting at a distance that is a computational interpretation of linear logic proof-nets. The calculus naturally express *linear-head reduction*, a notion of evaluation of proof nets that is strongly related to abstract machines. We show that our first (resp. second) quantitave type system characterizes linear-head, head and weak (resp. strong) normalizing sets of terms. All such characterizations are given by means of combinatorial arguments, *i.e.* there is a measure based on type derivations which decreases with respect to each reduction relation considered in the paper.

1 Introduction

It is quite difficult to reason about programs by only taking into account their syntax, so that many different semantic approaches were proposed to analyze them in a more abstract way. One typical tool to analyze relevant aspects of programs is the use of *type systems*. In particular, *intersection types* allow to characterize *head/weakly/strongly* normalizing terms, *i.e.* a term t is typable in an intersection type system iff t is head/weakly/strongly normalizing; *quantitative* information about the behaviour of programs can also be obtained if the intersection types enjoy *non-idempotence*.

Intersection Types (IT): *Simply typed* terms are *strongly normalizing* (*cf.* [7]) but the converse does not hold, *e.g.* the term $t := \lambda x.xx$. *Intersection Types* [15] extend the simply typed discipline with a finitary notion of polymorphism, listing type usage, that exactly captures the set of strongly normalizing terms. This is done by introducing a new constructor of types $\wedge$ together with a corresponding set of typing rules. For instance, the previous term t is typable with $((\sigma \to \sigma) \wedge \sigma) \to \sigma$ so that the first (resp. second) occurrence of the variable x is typed with $\sigma \to \sigma$ (resp. σ). Typically, intersection types are *idempotent*, *i.e.* $\sigma \wedge \sigma = \sigma$. Moreover, the intersection constructor is usually *commutative* and *associative*. Intersection types in their full generality provide a characterization of various properties of terms: models of the λ-calculus [8], characterization of head [17] as well as weakly [13,17] and strongly normalizing terms [33].

Non-Idempotent Intersection Types: The use of non-idempotent types [11] gives rise to resource aware semantics, which is suitable for computational

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 296–310, 2014.

complexity since it allows to extract *quantitative* information about reduction sequences. Indeed, the inequality $\sigma \wedge \sigma \neq \sigma$ can be read as the fact that two different uses of the variable x are not isomorphic to a single use. Relationship with Linear Logic [24] and Relevant Logic [23,18] provides an insight on the information refinement aspect of non-idempotent intersection types. The relation between the size of a non-idempotent intersection typing derivation and the *head/weak*-normalization execution time of λ-terms by means of abstract machines was established by D. de Carvalho [21]. Non-idempotence is also used in [9,20] to reason about the longest derivation of *strongly* β-normalizing terms in the λ-calculus by means of combinatorial arguments.

Calculi with Explicit Substitutions (ES) and Intersection Types: Calculi with ES refine the λ-calculus by decomposing β-reduction into small steps in order to specify different evaluation strategies implemented by abstract machines. In traditional calculi with ES [1], the operational semantics specifies the propagation of ES through the structure of the term until they reach a variable occurrence, on which they finally substitute or get garbage collected. But calculi with ES can also be interpreted in *Linear Logic* [22,28,26,5] by implementing another kind of operational semantics: their dynamics is defined using contexts (*i.e.* terms with holes) that allows the ES to act directly *at a distance* on single variable occurrences, with no need to commute with any other constructor in between. In other words, the propagation of substitutions is not performed by structural induction on terms, since they are only consumed according to the multiplicity of the variables.

Idempotent intersection type systems were used to characterize strongly normalizing terms of calculi with ES [34,27] while non-idempotence is used in [10] to prove the exact relationship between typing derivations and the number of steps of the longest reduction sequence of strongly-normalizing terms in the $\lambda\mathtt{s}$-calculus [26] and in the $\lambda\mathtt{lxr}$-calculus [28]. No study about linear-head, head and weak normalizing is provided in those works. Moreover, the systems are not *syntax-directed*, *i.e.* not all the typing derivations of t end with the same typing rule. As a consequence, the formal developments of proofs require a generation lemma which guarantees the existence of some typing derivations having a particular shape. This drawback makes the development of proofs more involved.

Contribution: This paper focuses on *functional programs* specified – via the Curry-Howard isomorphism – by *intuitionistic logic*, in *natural deduction* style. The operational semantics implements resource control by means of reduction rules describing the behaviour of *explicit* operators for *erasure* and *duplication*. The term language is the *linear substitution calculus* [3], called here $\mathtt{M}$*-calculus*, and obtained from Milner's calculus [36] and the structural λ-calculus [5].

Partial substitution allows to express *linear-head reduction* [19,35], a notion of evaluation of proof nets that is strongly related to significant aspects of computer science [32,2,4]. Linear-head reduction cannot be expressed as a simple strategy of the λ-calculus, where substitution acts on all free occurrences of a variable at once; this is probably one of the reasons why there are so few works investigating

it. In this paper we use logical systems to reason about different notions of normalization of terms, including those obtained with linear-head reduction.

More precisely, the quantitative semantics of programs used in this paper is given by two non-idempotent intersection type systems. The first one, based on [21], allows a characterization of linear-head, head and weakly normalizing terms. While full logical characterizations of head/weakly β-normalizing λ-terms were already given in the literature, the use of a logical/type system to directly characterize linear-head normalization in calculi with ES is new. The second system, another main contributions of this paper, gives a characterization of strongly normalizing terms.

Our type systems use multiset notation and are syntax-directed so that no generation lemmas are needed, thus making the development of proofs much more direct. Moreover, the type systems for strong normalization make use of a special notion of *witness* derivation for the arguments (of applications and explicit substitutions) which makes them particularly natural. All the characterizations in the paper are given by means of simple combinatorial arguments, *i.e.* there is a measure that can be associated to each typing derivation which is decreasing with respect to the different reduction relations considered in the paper.

Structure of the Paper: Sec. 2 presents the syntax and semantics of the $\mathtt{M}$-calculus and both typing systems. Sec. 3 presents the Linear-Head, Head and Weak-Normalization characterizations while Sec. 4 presents the Strong-Normalization characterization. We then conclude in Sec. 5.

2 The Linear Substitution Calculus

We first describe the syntax and the operational semantics of the $\mathtt{M}$-calculus, including some particular notions of rewriting such as linear-head reduction. We then introduce a notion of type and two different type systems that play a central role in the first part of the paper.

Syntax: Given a countable infinite set of symbols $x, y, z, \ldots$, three different syntactic categories for terms ($\mathcal{T}_{\mathtt{M}}$) and contexts ($\mathcal{C}_{\mathtt{M}}$) are defined by the following grammars:

$$\begin{array}{rll} \textbf{(terms)} & t, u, v ::= & x \mid tt \mid \lambda x.t \mid t[x/t] \\ \textbf{(term contexts)} & \mathtt{C} ::= & \Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}\ t \mid t\ \mathtt{C} \mid \mathtt{C}[x/t] \mid t[x/\mathtt{C}] \\ \textbf{(list contexts)} & \mathtt{L} ::= & \Box \mid \mathtt{L}[x/t] \end{array}$$

A term x is called a **variable**, tu an **application**, $\lambda x.t$ an **abstraction** and $t[x/u]$ a **closure** where $[x/u]$ is an **explicit substitution**. We write $tt_1 \ldots t_n$ for $(\ldots(tt_1)\ldots t_n)$. The notions of **free** and **bound** variables are defined as usual, in particular, $\mathtt{fv}(t[x/u]) := \mathtt{fv}(t) \setminus \{x\} \cup \mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) := \mathtt{fv}(t) \setminus \{x\}$, $\mathtt{bv}(t[x/u]) := \mathtt{bv}(t) \cup \{x\} \cup \mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) := \mathtt{bv}(t) \cup \{x\}$. We work with the standard notion of α-conversion *i.e.* renaming of bound variables for abstractions and substitutions. We write $\mathtt{C}[t]$ (resp. $\mathtt{L}[t]$) for the term obtained by replacing

the hole of $\mathtt{C}$ (resp. $\mathtt{L}$) by the term t. We write $\mathtt{C}[\![u]\!]$ or $\mathtt{L}[\![u]\!]$ when the free variables of u are not captured by the context, *i.e.* there are no abstractions or explicit substitutions in the context that binds the free variables of u. The set of **positions** of t, written $\mathtt{pos}(t)$, is the finite language over $\{0,1\}$ inductively defined as follows: $\epsilon \in \mathtt{pos}(t)$ for every t; $0p \in \mathtt{pos}(\lambda x.t)$ if $p \in \mathtt{pos}(t)$; $0p \in \mathtt{pos}(tu)$ (resp. $\mathtt{pos}(t[x/u])$) if $p \in \mathtt{pos}(t)$; $1p \in \mathtt{pos}(tu)$ (resp. $\mathtt{pos}(t[x/u])$) if $p \in \mathtt{pos}(u)$. The **subterm of t at position p** is written $t|_p$ and defined as expected. The term u **has an occurrence** in t iff there is $p \in \mathtt{pos}(t)$ such that $t|_p = u$. We write $|t|_x$ to denote the **number of free occurrences** of the variable x in the term t. All these notions are extended to contexts as expected.

Operational Semantics: The **$\mathtt{M}$-calculus** is given by the set of terms $\mathcal{T}_{\mathtt{M}}$ and the **reduction relation** $\to_{\mathtt{dB}\cup\mathtt{c}\cup\mathtt{w}}$, the **union** of $\to_{\mathtt{dB}}$, $\to_{\mathtt{c}}$, and $\to_{\mathtt{w}}$, denoted by $\to_{\mathtt{M}}$, which are, respectively, the closure by term contexts $\mathtt{C}$ of the following rewriting rules:

$$\begin{array}{lll} \mathtt{L}[\lambda x.t]u & \mapsto_{\mathtt{dB}} & \mathtt{L}[t[x/u]] \\ \mathtt{C}[\![x]\!][x/u] & \mapsto_{\mathtt{c}} & \mathtt{C}[\![u]\!][x/u] \\ t[x/u] & \mapsto_{\mathtt{w}} & t \qquad \text{if } |t|_x = 0 \end{array}$$

The names $\mathtt{dB}$, $\mathtt{c}$ and $\mathtt{w}$ stand for d*istant* B*eta*, c*ontraction* and w*eakening*, respectively. Rule $\mapsto_{\mathtt{dB}}$ (resp. $\mapsto_{\mathtt{c}}$) comes from the structural λ-calculus [5] (resp. Milner's calculus [36]), while $\mapsto_{\mathtt{w}}$ belongs to both calculi. By α-conversion we can assume in the rule $\mathtt{dB}$ that x may only be free in t and no variable in the domain of $\mathtt{L}$, defined as expected, has free occurrences in the term u. The *pushed out* list context $\mathtt{L}$ in rule $\mathtt{dB}$ can be obtained by using an equivalence related to Regnier's σ-equivalence [38]: $\mathtt{L}[\lambda x.t]u \sim_\sigma \mathtt{L}[(\lambda x.t)u] \to_{\mathtt{dB}} \mathtt{L}[t[x/u]]$. We will come back on this equivalence in Sec. 4.

The reflexive-transitive (resp. transitive) closure of $\to_{\mathtt{M}}$ is denoted by $\to_{\mathtt{M}}^*$ (resp. $\to_{\mathtt{M}}^+$). Given $t \in \mathcal{T}_{\mathtt{M}}$, t is **in $\mathtt{M}$-normal form**, written $t \in \mathtt{M}\text{-nf}$, if there is no t' s.t. $t \to_{\mathtt{M}} t'$; and t **has an $\mathtt{M}$-nf** iff there is $t' \in \mathtt{M}\text{-nf}$ such that $t \to_{\mathtt{M}}^* t'$. Moreover, t is **weakly $\mathtt{M}$-normalizing**, written $t \in \mathcal{WN}(\mathtt{M})$, iff t has an $\mathtt{M}$-nf, t is **strongly $\mathtt{M}$-normalizing** or $\mathtt{M}$-terminating, written $t \in \mathcal{SN}(\mathtt{M})$, if there is no infinite $\mathtt{M}$-reduction sequence starting at t. Every $\mathtt{M}$-term is $(\mathtt{c},\mathtt{w})$-strongly normalizing [29].

The notion of *redex occurrence* in this calculus is more subtle than the one in standard rewriting because one unique term may give rise to different reduction steps at the root, *e.g.* $(xu)[x/u]\ {}_{\mathtt{c}}\!\leftarrow (xx)[x/u] \to_{\mathtt{c}} (ux)[x/u]$. Thus, given $p \in \mathtt{pos}(t)$, p is said to be a **$\mathtt{dB}$-redex occurrence** of t if $t|_p = \mathtt{L}[\lambda x.t]u$, p is a **$\mathtt{w}$-redex occurrence** of t if $t|_p = v[x/u]$ with $|v|_x = 0$, and p is a **$\mathtt{c}$-redex occurrence** of t if $p = p_1p_2$, $t|_{p_1} = \mathtt{C}[\![x]\!][x/u]$ and $\mathtt{C}|_{p_2} = \Box$. For example 000 and 001 are both $\mathtt{c}$-redex occurrences of the term $\lambda z.(xx)[x/u]$.

The $\mathtt{M}$-calculus enjoys good properties required for calculi with ES (including simulation of β-reduction, preservation of strong normalization, confluence on terms and metaterms and full composition) [29]. It was recently used in different investigations of computer science [4,2,3].

The reduction relation $\to_{\mathtt{M}}$ can be refined in different ways, where the (reflexive-)transitive closures and normal-forms are defined as expected. The **non-erasing** reduction relation $\to_{\mathtt{M}\setminus\mathtt{w}}$ is given by $\to_{\mathtt{dB}\cup\mathtt{c}}$, and plays a key role in the characterization of strongly normalizing terms in Sec. 4. Another key subrelation studied in this paper is *linear-head reduction* [19,35], a strategy related to abstract machines [19] and linear logic [24]. To introduce this notion, we first define the set of **linear-head contexts** that are generated by the following grammar:

$$\mathtt{L_H} ::= \Box \mid \lambda x.\mathtt{L_H} \mid \mathtt{L_H}t \mid \mathtt{L_H}[x/t]$$

Linear-head M-reduction, written $\to_{\mathtt{L_HM}}$, is the closure under *linear-head contexts* of the rewriting rules $\{\mapsto_{\mathtt{dB}}, \mapsto_{\mathtt{c}|_{\mathtt{L_H}}}\}$, where $\mapsto_{\mathtt{c}|_{\mathtt{L_H}}}$ is the following variation of the rewriting rule $\mapsto_{\mathtt{c}}$:

$$\mathtt{L_H}[\![x]\!]x[x/u] \mapsto_{\mathtt{c}|_{\mathtt{L_H}}} \mathtt{L_H}[\![u]\!][x/u]$$

Indeed, the leftmost (*i.e. head*) occurrence of the variable x in $\mathtt{L_H}[\![x]\!]$ is substituted by u and this partial (*i.e. linear*) substitution is only performed on that head occurrence. The notion of $\mathtt{c}|_{\mathtt{L_H}}$**-redex occurrence** is defined as for the c-rule. A term t is **linear-head M-normalizing**, written $t \in \mathcal{L}_{\mathcal{H}}\mathcal{N}(\mathtt{M})$, iff t has an $\mathtt{L_H}\mathtt{M}$-nf. For example, if $t_0 := \lambda x.xy$ and $t_1 := x[y/z](\mathtt{II})$, where $\mathtt{I} := \lambda w.w$, then $t_0 \in \mathtt{M}$-nf, and so also $t_0 \in \mathtt{L_H}\mathtt{M}$-nf, while $t_1 \notin \mathtt{M}$-nf but $t_1 \in \mathtt{L_H}\mathtt{M}$-nf.

Types: We denote finite multisets by brackets, so that $[\,]$ denotes the empty multiset; $[a, a, b]$ denotes a multiset having two occurrences of the element a and one occurrence of b. We use $+$ for multiset union. Given a countable infinite set of *base types* $\alpha, \beta, \gamma, \ldots$ we consider **types** and **multiset types** defined by the following grammars:

(**types**)	$\tau, \sigma, \rho ::= \alpha \mid \mathcal{M} \to \tau$	
(**multiset types**)	$\mathcal{M} ::= [\tau_i]_{i \in I}$	where I is a finite set

Observe that our types are *strict* [16,6], *i.e.* the type on the right hand side of a functional type is never a multiset. They also make use of usual notations for multisets, as in [21], so that $[\sigma, \sigma, \tau]$ must be understood as $\sigma \wedge \sigma \wedge \tau$, where the symbol $\wedge$ is defined to enjoy commutative and associative axioms. When $\wedge$ verifies the axiom $\sigma \wedge \sigma = \sigma$, the underlying type system is **idempotent**, otherwise, like in this paper, it is **non-idempotent**.

Type assignments, written Γ, Δ, are functions from variables to multiset types, assigning the empty multiset to all but a finite set of the variables. The domain of Γ is given by $\mathtt{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\,]\}$. The **intersection of type assignments**, written $\Gamma + \Delta$, is defined by $(\Gamma + \Delta)(x) := \Gamma(x) + \Delta(x)$, where the symbol $+$ denotes multiset union. As a consequence $\mathtt{dom}(\Gamma + \Delta) = \mathtt{dom}(\Gamma) \cup \mathtt{dom}(\Delta)$. When $\mathtt{dom}(\Gamma)$ and $\mathtt{dom}(\Gamma)$ are disjoint we write $\Gamma; \Delta$ instead of $\Gamma + \Delta$. We write $\Gamma \setminus\!\setminus x$ for the assignement $(\Gamma \setminus\!\setminus x)(x) = [\,]$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$.

$$\frac{}{x{:}[\tau] \vdash x{:}\tau}\ (\mathtt{ax}) \qquad \frac{x{:}[\sigma_i]_{i\in I}; \Gamma \vdash t{:}\tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t[x/u]{:}\tau}\ (\mathtt{cut}_{\mathcal{HW}})$$

$$\frac{\Gamma \vdash t{:}\tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t{:}\Gamma(x) \to \tau}\ (\to \mathtt{i}) \qquad \frac{\Gamma \vdash t{:}[\sigma_i]_{i\in I} \to \tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash tu{:}\tau}\ (\to \mathtt{e}_{\mathcal{HW}})$$

Fig. 1. The Type System $\mathcal{HW}$ for the M-Calculus

Typing Rules $\{(\mathtt{ax}), (\to \mathtt{i})\}$ plus

$$\frac{x{:}[\sigma_i]_{i\in I}; \Gamma \vdash t{:}\tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I\cup\{w\}}}{\Gamma +_{i\in I\cup\{w\}} \Delta_j \vdash t[x/u]{:}\tau}\ (\mathtt{cut}_{\mathcal{S}})$$

$$\frac{\Gamma \vdash t{:}[\sigma_i]_{i\in I} \to \tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I\cup\{w\}}}{\Gamma +_{i\in I\cup\{w\}} \Delta_i \vdash tu{:}\tau}\ (\to \mathtt{e}_{\mathcal{S}})$$

Fig. 2. The Type System $\mathcal{S}$ for the M-Calculus

The Type Systems: Type judgments have the form $\Gamma \vdash t{:}\tau$, where Γ is a type assignment, t is a term and τ is a type. The type systems $\mathcal{HW}$, after $\mathcal{H}$ead-$\mathcal{W}$eak, and $\mathcal{S}$, after $\mathcal{S}$trong, for the M-calculus are given respectively in Fig. 1 and 2. A **(typing) derivation** in system X is a tree obtained by applying the (inductive) typing rules of system X. The notation $\Gamma \vdash_X t{:}\tau$ means there is a derivation of the judgment $\Gamma \vdash t{:}\tau$ in system X. The term t is **typable** in system X, or **X-typable**, iff there are Γ and τ s.t. $\Gamma \vdash_X t{:}\tau$. We use the capital Greek letters $\Phi, \Psi, \ldots$ to name type derivations, *e.g.* we write $\Phi \triangleright \Gamma \vdash_X t{:}\tau$. The **size** of a type derivation Φ is a positive natural number $\mathtt{sz}(\Phi)$ defined as expected.

The rules $(\mathtt{ax})$, $(\to \mathtt{i})$ and $(\to \mathtt{e}_{\mathcal{HW}})$ in system $\mathcal{HW}$ come from a relational semantics for linear logic [21]. Remark in particular that the axiom is *relevant* (so there is no weakening) and the rules for application and substitution are *multiplicative*, both characteristics are related to the *resource aware* semantics. A particular case of rule $(\to \mathtt{e}_{\mathcal{HW}})$ is when $I = \emptyset$: the subterm u occuring in the *typed* term tu turns out to be *untyped*. Thus for example, from the derivation $x{:}[\sigma] \vdash_{\mathcal{HW}} \lambda y.x{:}[\,] \to \sigma$ we can construct $x{:}[\sigma] \vdash_{\mathcal{HW}} (\lambda y.x)\Omega{:}\sigma$, where Ω is the non-terminating term $(\lambda z.zz)(\lambda z.zz)$. This is precisely the reason why rules $(\to \mathtt{e}_{\mathcal{S}})$ and $(\mathtt{cut}_{\mathcal{S}})$ in Fig. 2, the system which characterizes strongly-normalizing terms, always asks a **witness** typing derivation for the arguments of applications and substitutions. Indeed, if $I = \emptyset$, then the argument u will be typed with the witness derivation $\Delta_w \vdash u{:}\sigma_w$, whatever the type σ_w is. This witness derivation for u is essential to guarantee strong-normalization of u (and thus of tu and $t[x/u]$). When $I \neq \emptyset$ the rules $(\to \mathtt{e}_{\mathcal{S}})$ and $(\mathtt{cut}_{\mathcal{S}})$ also require a

witness derivation for u, whose use is necessary in order to deal with the $\mathtt{c}$-rule when $|\mathtt{C}[\![x]\!]|_x = 1$ (see discussion after Lem. 4). Last, remark that an alternative definition of rules $(\to \mathtt{e}_{\mathcal{S}})$ and $(\mathtt{cut}_{\mathcal{S}})$ given by adding $I \neq \emptyset$ to rules $(\to \mathtt{e}_{\mathcal{HW}})$ and $(\mathtt{cut}_{\mathcal{HW}})$, respectively, would not be complete: terms like $x[y/z]$ or $(\lambda y.x)z$ become untypable.

Given $\Phi \rhd \Gamma \vdash_{\mathcal{HW}} t{:}\sigma$, not every free variable of t necessarily appears in the domain of Γ, this is for example the case in $x{:}[\sigma] \vdash_{\mathcal{HW}} (\lambda y.x)z{:}\sigma$. More precisely, the systems enjoy the following (weak/strong) relevance properties, that can be easily shown by induction on derivations.

Lemma 1. *If* $\Phi \rhd \Gamma \vdash_{\mathcal{HW}} t{:}\sigma$ *then* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$. *If* $\Phi \rhd \Gamma \vdash_{\mathcal{S}} t{:}\sigma$, *then* $\mathtt{dom}(\Gamma) = \mathtt{fv}(t)$.

In contrast to other intersection type systems for ES in the literature, the typing rules of our systems are syntax oriented, so that generation lemmas are not needed to distinguish particular syntactical forms of derivations.

3 About Linear-Head, Head and Weak M-Normalization

We show in this section two main results. The first one (Sec. 3.1) characterizes linear-head and head $\mathtt{M}$-normalizing terms by means of $\mathcal{HW}$-typability. This result generalizes to calculi with ES the well-known *logical* characterization of *head* β-normalizing λ-terms [17,21]. The $\mathcal{HW}$-type system is known to type also some *non* weakly $\mathtt{M}$-normalizing terms: for instance, if Ω is any non-terminating term, then $x{:}[\,] \to \sigma \vdash_{\mathcal{HW}} x\Omega{:}\sigma$. We then characterize the set of weakly $\mathtt{M}$-normalizing terms, our second result (Sec. 3.2), by restricting the $\mathcal{HW}$-typing derivations to some particular ones. But first, let us develop some key technical tools.

To understand which are the redex occurrences actually constrained by the type system, let us consider a derivation $\Phi \rhd \Gamma \vdash_{\mathcal{HW}} t{:}\tau$. A position $p \in \mathtt{pos}(t)$ is a **typed occurrence** of Φ if either $p = \epsilon$, or $p = ip'$ $(i = 0, 1)$ and $p' \in \mathtt{pos}(t|_i)$ is a typed occurrence of *some* of their corresponding subderivations of Φ. A redex occurrence of t which is also a typed occurrence of Φ is a **redex T-occurrence** of t in Φ. Thus for example, given the following derivations Φ and Φ', we have that ϵ, 0, 1 and 10 are T-occurrences in Φ and Φ', while 11 is a T-occurrence in Φ but not in Φ'.

$$\Phi \rhd \dfrac{\dfrac{\dfrac{}{x{:}[[\tau,\tau]\to\tau] \vdash x{:}[\tau,\tau]\to\tau}}{y{:}[[\,]\to\tau] \vdash y{:}[\,]\to\tau} \qquad \dfrac{}{y{:}[[\tau]\to\tau] \vdash y{:}[\tau]\to\tau} \qquad \dfrac{}{z{:}[\tau] \vdash z{:}\tau}}{\dfrac{y{:}[[\,]\to\tau] \vdash yz{:}\tau}{} \qquad \dfrac{y{:}[[\tau]\to\tau], z{:}[\tau] \vdash yz{:}\tau}{}}{x{:}[[\tau,\tau]\to\tau], y{:}[[\,]\to\tau, [\tau]\to\tau], z{:}[\tau] \vdash x(yz){:}\tau}$$

$$\Phi' \rhd \dfrac{\dfrac{}{x{:}[[\tau,\tau]\to\tau] \vdash x{:}[\tau,\tau]\to\tau} \qquad \dfrac{}{y{:}[[\,]\to\tau] \vdash y{:}[\,]\to\tau} \qquad \dfrac{}{y{:}[[\,]\to\tau] \vdash y{:}[\,]\to\tau}}{\dfrac{}{y{:}[[\,]\to\tau] \vdash yz{:}\tau}}}{x{:}[[\tau,\tau]\to\tau], y{:}[[\,]\to\tau, [\,]\to\tau] \vdash x(yz){:}\tau}$$

The notion of T-occurrence plays a key role in the Subject Reduction (SR) lemma, which is based on a subtle *partial* substitution lemma, a refinement of the standard substitution lemma used in the λ-calculus.

Lemma 2 (SR I). *Let $\Phi \triangleright \Gamma \vdash_{\mathcal{HW}} t{:}\tau$. If $t \to_{\mathtt{M}} t'$ reduces a $(\mathtt{dB}, \mathtt{c}, \mathtt{w})$-redex T-occurrence of t in Φ then $\Phi' \triangleright \Gamma \vdash_{\mathcal{HW}} t'{:}\tau$ and $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

As an example, consider $\Phi'' \triangleright y{:}[[\,]\to[\,]\to\tau] \vdash_{\mathcal{HW}} (xxx)[x/y]{:}\tau$. Then the (typed) reduction step $(xxx)[x/y] \to_{\mathtt{c}} (yxx)[x/y]$ decreases the measure of Φ'' but thereafter $(yxx)[x/y] \to_{\mathtt{c}} (yyx)[x/y] \to_{\mathtt{c}} (yyy)[x/y]$ are not decreasing reduction steps since they act on untyped occurrences.

As a corollary, termination holds for *any* strategy reducing only redexes T-occurrences, an important key point used in Sec. 3.1 and 3.2.

Corollary 1. *If $\Phi \triangleright \Gamma \vdash_{\mathcal{HW}} t{:}\tau$, then any $\mathtt{M}$-reduction sequence contracting only $(\mathtt{dB}, \mathtt{c}, \mathtt{w})$-redex T-occurrences is finite.*

Types of terms can also be recovered by means of Subject Expansion (SE), a property which will be particularly useful in Sec. 3.1 and 3.2.

Lemma 3 (SE I). *If $\Gamma \vdash_{\mathcal{HW}} t'{:}\tau$ and $t \to_{\mathtt{M}} t'$ then $\Gamma \vdash_{\mathcal{HW}} t{:}\tau$.*

3.1 Linear-Head and Head $\mathtt{M}$-Normalization

Linear-head reduction [19,35] comes from a fine notion of evaluation for proof nets [25]. It is a particular reduction strategy of the $\mathtt{M}$-calculus although it is not a strategy of β-reduction. In contrast to *head*-reduction for λ-calculus the reduction relation $\to_{\mathtt{L_H M}}$ for $\mathtt{M}$-terms is non-deterministic: $y[y/w][x/z] \,{}_{\mathtt{L_H M}}\!\leftarrow (\lambda x.y[y/w])z \to_{\mathtt{L_H M}} (\lambda x.w[y/w])z$. This behaviour is however safe since $\to_{\mathtt{L_H M}}$ has the diamond property [7].

Another remarkable property of linear-head reduction is that the hole of the contexts $\mathtt{L_H}$ cannot be duplicated nor erased. This is related to a recent result [3] stating that linear-head reduction is *standard* for the $\mathtt{M}$-calculus, exactly as *left-to-right* reduction is standard for the λ-calculus.

We now refine a known result in the λ-calculus which characterizes *head*-normalizing terms by means of intersection types, either idempotent [17,8][1] or non-idempotent [21]. Indeed, the set of linear-head $\mathtt{M}$-normalizing terms coincides with the set of $\mathcal{HW}$-typable terms.

Lemma 4. *If $\Phi \triangleright \Gamma \vdash_{\mathcal{HW}} t{:}\tau$ and t has no $(\mathtt{dB}, \mathtt{c}|_{\mathtt{L_H}})$-redexes T-occurrences in Φ, then $t \in \mathtt{L_H M}$-nf.*

It is worth noticing that Lem. 4 does not hold for *head*-nfs. Indeed, the term $(yxx)[x/y]$ in the example just after Lem. 2 does not have any redex T-occurrence (the only two $\mathtt{c}$-redexes occurrences are untyped), and is not a *head*-nf. This emphasizes the fact that linear-head reduction is more pertinent for calculi with ES than head reduction. We conclude by

[1] Although idempotency was not explicity mentioned in [17], a remark on pp. 55 points out the meaninglessness of duplication of types in a sequence.

Theorem 1. *Let $t \in \mathcal{T}_{\mathtt{M}}$. Then $t \in \mathcal{L}_{\mathcal{H}}\mathcal{N}(\mathtt{M})$ iff t is $\mathcal{HW}$-typable.*

Proof. Let $t \in \mathcal{L}_{\mathcal{H}}\mathcal{N}(\mathtt{M})$. We proceed by induction on the length of the linear-head $\mathtt{M}$-normalizing reduction using Lem. 3 (see [30] for details).

Let t be $\mathcal{HW}$-typable. By Cor. 1 the strategy consisting in the contraction of $(\mathtt{dB}, \mathtt{c}|_{\mathtt{L_H}})$-redex T-occurrences terminates in a term t' without such redexes. The term t' is typable by Lem. 2 and t' is a $\mathtt{L_H M}$-nf by Lem. 4. Thus, $t \in \mathcal{L}_{\mathcal{H}}\mathcal{N}(\mathtt{M})$.

We can finally characterize head-normalization. A term t is **head-normalizing**, written $t \in \mathcal{HN}(\mathtt{M})$, iff t $\mathtt{M}$-reduces to a term of the form $\lambda x_1 \dots x_n.y u_1 \dots u_m$ for some $n \geq 0, m \geq 0$.

Theorem 2. *Let $t \in \mathcal{T}_{\mathtt{M}}$. Then $t \in \mathcal{HN}(\mathtt{M})$ iff t is $\mathcal{HW}$-typable.*

Proof. For the if implication we have $\mathcal{HN}(\mathtt{M}) \subseteq \mathcal{L}_{\mathcal{H}}\mathcal{N}(\mathtt{M})$ so we conclude by Thm. 1. Otherwise, t $\mathcal{HW}$-typable implies by Thm. 1 that $t \to_{\mathtt{M}}^* t'$, where $t' \in \mathtt{L_H M}$-nf. The (terminating) reduction relation $\to_{(\mathtt{c},\mathtt{w})}$ on t' gives a term of the required form.

3.2 Weak M-Normalization

In this section we use the type system $\mathcal{HW}$ to characterize weakly $\mathtt{M}$-normalizing terms, a result that extends the well-known characterization [17] of weakly β-normalizing in the λ-calculus. As in [17,13], $\mathcal{HW}$-typability alone does not suffice to characterize weak $\mathtt{M}$-normalizing terms (see an example at the beginning of Sec. 3). The type $[\,]$ plays a similar rôle to the universal ω type in [17,13], although it is restricted to occur only in the domain type of a function that accepts any kind of argument. We then restrict the allowed typing derivations in order to recover such a characterization. Indeed, the set of **positive** (resp. **negative**) subtypes of a type is the smallest set satisfying the following conditions (*cf.*[13]).

- $A \in \mathcal{P}(A)$.
- $A \in \mathcal{P}([\sigma_i]_{i\in I})$ if $\exists i\ A \in \mathcal{P}(\sigma_i)$; $A \in \mathcal{N}([\sigma_i]_{i\in I})$ if $\exists i\ A \in \mathcal{N}(\sigma_i)$.
- $A \in \mathcal{P}(\mathcal{M} \to \tau)$ if $A \in \mathcal{N}(\mathcal{M})$ or $A \in \mathcal{P}(\tau)$; $A \in \mathcal{N}(\mathcal{M} \to \tau)$ if $A \in \mathcal{P}(\mathcal{M})$ or $A \in \mathcal{N}(\tau)$.
- $A \in \mathcal{P}(\Gamma)$ if $\exists\ y \in \mathtt{dom}(\Gamma)$ s.t. $A \in \mathcal{N}(\Gamma(y))$; $A \in \mathcal{N}(\Gamma)$ if $\exists\ y \in \mathtt{dom}(\Gamma)$ s.t. $A \in \mathcal{P}(\Gamma(y))$.
- $A \in \mathcal{P}(\langle \Gamma, \tau \rangle)$ if $A \in \mathcal{P}(\Gamma)$ or $A \in \mathcal{P}(\tau)$; $A \in \mathcal{N}(\langle \Gamma, \tau \rangle)$ if $A \in \mathcal{N}(\Gamma)$ or $A \in \mathcal{N}(\tau)$.

As an example, $[\,] \in \mathcal{P}([\,])$, so that $[\,] \in \mathcal{N}([\,] \to \sigma)$, $[\,] \in \mathcal{P}(x{:}[[\,] \to \sigma])$ and $[\,] \in \mathcal{P}(\langle x{:}[[\,] \to \sigma], \sigma \rangle)$.

Lemma 5. *Let $\Phi \rhd \Gamma \vdash_{\mathcal{HW}} t{:}\tau$ s.t. $[\,] \notin \mathcal{P}(\langle \Gamma, \tau \rangle)$. If t has no $(\mathtt{dB}, \mathtt{c}, \mathtt{w})$-redex T-occurrences in Φ, then $t \in \mathtt{M}$-nf.*

Theorem 3. *Let $t \in \mathcal{T}_{\mathtt{M}}$. Then, $t \in \mathcal{WN}(\mathtt{M})$ iff $\Gamma \vdash_{\mathcal{HW}} t{:}\tau$ and $[\,] \notin \mathcal{P}(\langle \Gamma, \tau \rangle)$.*

Proof. If $t \in \mathcal{WN}(\mathtt{M})$, we proceed by induction on the length of the $\mathtt{M}$-normalizing reduction sequence using Lem. 3 (see in [30] for details).

Suppose $\Gamma \vdash_{\mathcal{HW}} t{:}\tau$ and $[\,] \notin \mathcal{P}(\langle \Gamma, \tau \rangle)$. By Cor. 1 the strategy of contracting only redex T-occurrences terminates in a term t' without such redexes. The term t' is typable by Lem. 2 and then t' turns out to be a $\mathtt{M}$-nf by Lem. 5. Thus, $t \in \mathcal{WN}(\mathtt{M})$.

4 About Strong $\mathtt{M}$-Normalization

In this section we show the third main result of the paper which is a characterization of the set of strongly $\mathtt{M}$-normalizing terms by means of $\mathcal{S}$-typability. The proof is done in several steps. The first key point is the characterization of the set of strongly $\mathtt{M}\backslash\mathtt{w}$-normalizing terms (instead of $\mathtt{M}$-normalizing terms). For that, SR and SE lemmas for the $\mathcal{S}$-type system are needed, and an inductive characterization of the set $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$ turns out to be helpful to obtain them. The second key point is the equivalence between strongly $\mathtt{M}$ and $\mathtt{M}\backslash\mathtt{w}$-normalizing terms. While the inclusion $\mathcal{SN}(\mathtt{M}) \subseteq \mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$ is straightforward, the fact that every $\mathtt{w}$-reduction step can be *postponed* w.r.t. any $\mathtt{M}\backslash\mathtt{w}$-step (Lem. 11) turns out to be crucial to show $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w}) \subseteq \mathcal{SN}(\mathtt{M})$.

We first introduce the **graphical equivalence** $\sim$ on $\mathtt{M}$-terms, given by the contextual, transitive, symmetric and reflexive closure of the following three axioms[2]

$$\begin{array}{lll} t[x/u][y/v] \approx_{\mathtt{CS}} t[y/v][x/u] & \text{if } y \notin \mathtt{fv}(u) \;\&\; x \notin \mathtt{fv}(v) \\ (\lambda y.t)[x/u] \approx_{\sigma_1} \lambda y.t[x/u] & \text{if } y \notin \mathtt{fv}(u) \\ (tv)[x/u] \approx_{\sigma_2} t[x/u]v & \text{if } x \notin \mathtt{fv}(v) \end{array}$$

This equivalence, related to Regnier's σ-equivalence [38] on λ-terms (resp. σ-equivalence on terms with ES [5]), preserves types, a property used to perform some *safe* transformations of terms in order to inductively define the set $\mathcal{SN}(\mathtt{M}\backslash\mathtt{w})$ (*cf.* clause (E)). Note that, for any $t \in \mathcal{T}_{\mathtt{M}}$, we have that the set $\{t' \mid t \to_{\mathtt{M}\backslash\mathtt{w}} t'\}$ is finite. Therefore, for any $t \in \mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$, the **depth of** t can be defined as the maximal length of $\mathtt{M}\backslash\mathtt{w}$-reduction sequences starting at t, denoted by $\eta_{\mathtt{M}\backslash\mathtt{w}}(t)$.

Lemma 6 (Invariance for $\sim$). *Let $t, t' \in \mathcal{T}_{\mathtt{M}}$ s.t. $t \sim t'$. Then, 1) $\eta_{\mathtt{M}\backslash\mathtt{w}}(t) = \eta_{\mathtt{M}\backslash\mathtt{w}}(t')$. 2) If $\Phi \triangleright \Gamma \vdash_{\mathcal{S}} t{:}\tau$, then $\Phi' \triangleright \Gamma \vdash_{\mathcal{S}} t'{:}\tau$. Moreover, $\mathtt{sz}(\Phi){=}\mathtt{sz}(\Phi')$.*

In contrast to system $\mathcal{HW}$, whose typing measure $\mathtt{sz}()$ is only decreasing w.r.t. reduction of redex *typed* occurrences, the system $\mathcal{S}$ enjoys a stronger subject reduction property, guaranteeing that *every* reduction decreases the measure $\mathtt{sz}()$ of terms (whose redexes are all typed now).

Lemma 7 (SR II). *Let $\Phi \triangleright \Gamma \vdash_{\mathcal{S}} t{:}\tau$. If $t \to_{\mathtt{M}\backslash\mathtt{w}} t'$ then $\Phi' \triangleright \Gamma \vdash_{\mathcal{S}} t'{:}\tau$ and $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

[2] Eventhough only σ_2 will be used later to give an inductive definition of $\mathcal{SN}(\mathtt{M})$, the equivalence is presented as a whole.

Notice that the previous lemma does not hold if the witness derivation in the rules $(\to \mathtt{e}_{\mathcal{S}})$ and $(\mathtt{cut}_{\mathcal{S}})$ in Fig. 2 is only required for the case $I = \emptyset$. For example, given $x[x/y] \to_{\mathtt{c}} y[x/y]$ and their respective typing derivations Φ and Φ', one would have $\mathtt{sz}(\Phi) = \mathtt{sz}(\Phi_y) + 2 = 2 \cdot \mathtt{sz}(\Phi_y) + 1 = \mathtt{sz}(\Phi')$. One can even have $\mathtt{sz}(\Phi) < \mathtt{sz}(\Phi')$ if y is replaced by an arbitrary bigger term. Notice that an erasing step $v[x/u] \to_{\mathtt{w}} v$ also decreases $\mathtt{sz}(\_)$ but the type assignment for u may change w.r.t. that of $v[x/u]$.

Lemma 8 (SE II). *Let $\Gamma \vdash_{\mathcal{S}} t' : \tau$. If $t \to_{\mathtt{M}\setminus\mathtt{w}} t'$ then $\Gamma \vdash_{\mathcal{S}} t : \tau$.*

Notice that expansion does not hold for $\to_{\mathtt{w}}$-reduction. For example $x : [\sigma] \vdash_{\mathcal{S}} x : \sigma$ and $x[y/\Omega] \to_{\mathtt{w}} x$, but $x : [\sigma] \not\vdash_{\mathcal{S}} x[y/\Omega] : \sigma$.

These technical tools are now used to prove that $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$ coincides exactly with the set of $\mathcal{S}$-typable terms. To close the picture, *i.e.* to show that also $\mathcal{SN}(\mathtt{M})$ coincides with the set of $\mathcal{S}$-typable terms, we establish an equivalence between $\mathcal{SN}(\mathtt{M})$ and $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$. This is done constructively thanks to the use of an inductive definition for $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$. Indeed, the **inductive set of $\mathtt{M}\setminus\mathtt{w}$-strongly-normalizing terms** is the smallest subset of $\mathcal{T}_{\mathtt{M}}$ that satisfies the following properties:

(V) If $t_1, \ldots, t_n \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$, then $x t_1 \ldots t_n \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.
(L) If $t \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$, then $\lambda x.t \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.
(W) If $t, s \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$ and $|t|_x = 0$, then $t[x/s] \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.
(B) If $u[x/v]t_1, \ldots, t_n \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$, then $(\lambda x.u)v t_1, \ldots, t_n \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.
(C) If $\mathtt{C}[\![u]\!][x/u] \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$, then $\mathtt{C}[\![x]\!][x/u] \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.
(E) If $(tu)[x/s] \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$ and $|u|_x = 0$, then $t[x/s]u \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.

Note the use of the σ_2-axiom in the last clause of the definition. It is not surprising that $\mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$ turns out to be equivalent to $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$, a property which considerably simplifies the proof of Lemma 10.

Lemma 9. $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w}) = \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$

Proof. Given $o \in \mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$, we show $o \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$ by induction on $\langle \eta_{\mathtt{M}\setminus\mathtt{w}}(o), |o| \rangle$. The converse uses induction on the definition of $\mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$.

Lemma 10. *Let $t \in \mathcal{T}_{\mathtt{M}}$. If $t \in \mathcal{SN}(\mathtt{M}\setminus\mathtt{w})$ then t is $\mathcal{S}$-typable.*

Proof. Use the equality $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w}) =_{L.\ 9} \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$ to reason by induction on $t \in \mathcal{ISN}(\mathtt{M}\setminus\mathtt{w})$. The proof also uses Lem. 6 and 8 (see [30] for details).

In order to infer $\mathcal{SN}(\mathtt{M}\setminus\mathtt{w}) \subseteq \mathcal{SN}(\mathtt{M})$, the following postponement property is crucial.

Lemma 11 (Postponement). *Let $v \in \mathcal{T}_{\mathtt{M}}$. If $v \to_{\mathtt{w}}^{+} \to_{\mathtt{M}\setminus\mathtt{w}} v'$ then $v \to_{\mathtt{M}\setminus\mathtt{w}} \to_{\mathtt{w}}^{+} v'$.*

Proof. We first show by cases $v \to_{\mathtt{w}} \to_{\mathtt{M}\setminus\mathtt{w}} v'$ implies $v \to_{\mathtt{M}\setminus\mathtt{w}} \to_{\mathtt{w}}^{+} v'$. Then, the statement holds by induction on the number of $\mathtt{w}$-steps from v.

Lemma 12 (From $\mathtt{M}\backslash\mathtt{w}$ to $\mathtt{M}$). *Let $t\in\mathcal{T}_{\mathtt{M}}$. If $t\in\mathcal{SN}(\mathtt{M}\backslash\mathtt{w})$, then $t\in\mathcal{SN}(\mathtt{M})$.*

Proof. We show that any reduction sequence $\rho : t \to_{\mathtt{M}} \dots$ is finite by induction on the pair $\langle t, n\rangle$, where n is the maximal number such that ρ can be decomposed as $\rho : t \to_{\mathtt{w}}^{n} t' \to_{\mathtt{M}\backslash\mathtt{w}} t'' \to \dots$ (this is well-defined since $\to_{\mathtt{w}}$ is trivially terminating). We compare the pair $\langle t, n\rangle$ using $\to_{\mathtt{M}\backslash\mathtt{w}}$ for the first component (this is well-founded since $t \in \mathcal{SN}(\mathtt{M}\backslash\mathtt{w})$ by hyp.) and the standard order on natural numbers for the second one. When the reduction sequence starts with at least one $\mathtt{w}$-step we conclude by Lem. 11. All the other cases are straightforward.

We conclude this section with the third main theorem for $\mathtt{M}$-calculus:

Theorem 4. *Let $t \in \mathcal{T}_{\mathtt{M}}$. Then t is $\mathcal{S}$-typable iff $t \in \mathcal{SN}(\mathtt{M})$.*

Proof. Let $\Phi \triangleright \Gamma \vdash_{\mathcal{S}} t{:}\tau$. Assume $t \notin \mathcal{SN}(\mathtt{M}\backslash\mathtt{w})$ so that $\exists\infty$ sequence $t = t_0 \to_{\mathtt{M}\backslash\mathtt{w}} t_1 \to_{\mathtt{M}\backslash\mathtt{w}} t_2 \to_{\mathtt{M}\backslash\mathtt{w}} \cdots$. By Lem. 7 $\Phi_i \triangleright \Gamma \vdash t_i{:}\tau$ for every i, and $\exists\infty$ sequence $\mathtt{sz}(\Phi_0) > \mathtt{sz}(\Phi_1) > \mathtt{sz}(\Phi_2) > \dots$, which leads to a contradiction. Therefore, $t \in \mathcal{SN}(\mathtt{M}\backslash\mathtt{w}) \subseteq_{\text{Lem. 12}} \mathcal{SN}(\mathtt{M})$.

For the converse, $t \in \mathcal{SN}(\mathtt{M}) \subseteq \mathcal{SN}(\mathtt{M}\backslash\mathtt{w})$ because $\to_{\mathtt{M}\backslash\mathtt{w}} \subseteq \to_{\mathtt{M}}$. We conclude by Lem. 10.

A corollary of this result is that $\mathtt{M}$-calculus enjoys the **(IE)** Property [26], namely, if $t\{x/u\}$ and u are in $\mathcal{SN}(\mathtt{M})$, then $t[x/u]$ is also in $\mathcal{SN}(\mathtt{M})$. Indeed, Thm. 4 gives $t\{x/u\}$ and u typable, then Lem. 30 in [30] gives the exact premises to type $t[x/u]$, which belongs to $\mathcal{SN}(\mathtt{M})$ by Thm. 4.

5 Conclusion

This paper studies quantitative types for the linear substitution calculus for which we characterized linear-head, head, weak and strongly normalizing sets of terms. In particular, the correspondence between head β-normalization for λ-terms and linear-head $\mathtt{M}$-normalization for terms with ES can now be obtained by means of an *indirect* logical reasoning (*i.e.* the $\mathcal{HW}$-system), in contrast to the operational result given in [4].

The type systems are given by simple formalisms: intersection is represented by multisets, the typing rules are syntax-oriented and no subtyping relation is used. Similar ideas can be applied [30] in the framework of intuitionistic sequent style, giving rise to a reformulation of Herbelin's calculus which is interesting in its own. The $\mathcal{HW}$-system also enjoys the inhabitation property for λ-calculus [12], which is a proper sub-calculus of the linear substitution calculus.

Our strong normalization characterization implies that the number of steps of the longest reduction sequences of terminating $\mathtt{M}$-terms is bounded by the the size of typing derivations. But despite the use of quantitative types, we did not give an exact upper bound, as done for example in [9,20]. This remains as future work.

Although type inference is undecidable for any system characterizing termination properties, semi-decidable restrictions are expected to hold. Principal

typing is a property (*cf.* [21]) which allows to obtain partial typing inference algorithms [40,39,31] and exact bounds for termination (*cf.*[10]). Moreover, relevance in the sense of [18] is a key property to obtain principal typings. Therefore semi-decidable typing inference algorithms are also expected to hold for our two non-idempotent type systems.

Neergard *et al.* [37] proved that type inference and execution of typed programs are in different (resp. the same) classes of complexity in the idempotent (resp. non-idempotent) case. However, the system introduced by Carlier *et al.* [14] allows to relax the notion of type linearity. An interesting challenge would be relax the notion of linear types in order to gain expressivity while staying in a different class.

Last but not least, the inhabitation problem for idempotent intersection types in the λ-calculus is known to be undecidable [41], while the problem was recently shown to be decidable in the non-idempotent case [12]. An interesting question concerns the inhabitation problems for our non-idempotent type systems.

Acknowledgment. This work was partially funded by the international project **DeCOPA STIC-AmSud 146/2012** (second author partially supported by a PDJ grant from the Brazilian agency CAPES) and the French-Argentinian Laboratory in Computer Science **INFINIS**.

References

1. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.-J.: Explicit substitutions. JFP 1(4), 375–416 (1991)
2. Accattoli, B.: Evaluating functions as processes. In: TERMGRAPH. EPTCS, vol. 110, pp. 41–55 (2013)
3. Accattoli, B., Bonelli, E., Kesner, D., Lombardi, C.: A nonstandard standardization theorem. In: POPL. ACM (2014)
4. Accattoli, B., Dal Lago, U.: On the invariance of the unitary cost model for head reduction. RTA, LIPIcs 15, 22–37 (2012)
5. Accattoli, B., Kesner, D.: The structural λ-calculus. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 381–395. Springer, Heidelberg (2010)
6. van Bakel, S.: Complete restrictions of the intersection type discipline. TCS 102(1), 135–163 (1992), version with a corrected proof at, `http://www.doc.ic.ac.uk/~svb/Research/Papers/TCS92corrected.pdf`
7. Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics, revised edn. Elsevier Science, Amsterdan (1984)
8. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. Bulletin of Symbolic Logic 48, 931–940 (1983)
9. Bernadet, A., Lengrand, S.: Complexity of strongly normalising λ-terms via non-idempotent intersection types. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 88–107. Springer, Heidelberg (2011)
10. Bernadet, A., Lengrand, S.: Non-idempotent intersection types and strong normalisation. LMCS 9(4) (2013)
11. Boudol, G., Curien, P.-L., Lavatelli, C.: A semantics for lambda calculi with resources. MSCS 9(4), 437–482 (1999)

12. Bucciarelli, A., Kesner, D., Ronchi Della Rocca, S.: The inhabitation problem for non-idempotent intersection types (submitted)
13. Cardone, F., Coppo, M.: Two extension of Curry's type inference system. In: Logic and Computer Science. APIC Series, vol. 31, pp. 19–75. Academic Press (1990)
14. Carlier, S., Polakow, J., Wells, J.B., Kfoury, A.J.: System E: Expansion variables for flexible typing with linear and non-linear types and intersection types. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 294–309. Springer, Heidelberg (2004)
15. Coppo, M., Dezani-Ciancaglini, M.: A new type-assignment for lambda terms. Archiv für Mathematische Logik und Grundlagenforschung 19, 139–156 (1978)
16. Coppo, M., Dezani-Ciancaglini, M.: An extension of the basic functionality theory for the lambda-calculus. Notre Dame J. of Form. Log. 21(4), 685–693 (1980)
17. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Functional characters of solvable terms. Mathematical Logic Quarterly 27(2-6), 45–58 (1981)
18. Damiani, F., Giannini, P.: A decidable intersection type system based on relevance. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 707–725. Springer, Heidelberg (1994)
19. Danos, V., Regnier, L.: Head linear reduction (2003), `iml.univ-mrs.fr/~regnier/articles/pam.ps.gz`
20. De Benedetti, E., Ronchi Della Rocca, S.: Bounding normalization time through intersection types. ITRS, EPTCS 121, 48–57 (2013)
21. de Carvalho, D.: Sémantiques de la logique linéaire et temps de calcul. PhD Univ. Aix-Marseille II (2007)
22. Di Cosmo, R., Kesner, D., Polonovski, E.: Proof nets and explicit substitutions. MSCS 13(3), 409–450 (2003)
23. Gabbay, D., de Queiroz, R.: Extending the Curry-Howard interpretation to linear, relevant and other resource logics. JSL 57(4), 1319–1365 (1992)
24. Girard, J.-Y.: Linear logic. TCS 50, 1–102 (1987)
25. Girard, J.-Y.: Proof-nets: The parallel syntax for proof-theory. In: Logic and Algebra, pp. 97–124 (1996)
26. Kesner, D.: The theory of calculi with explicit substitutions revisited. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 238–252. Springer, Heidelberg (2007)
27. Kesner, D.: A theory of explicit substitutions with safe and full composition. LMCS 5(3) (2009)
28. Kesner, D., Lengrand, S.: Resource operators for lambda-calculus. IandC 205(4), 419–473 (2007)
29. Kesner, D., Ó Conchúir, S.: Milner's lambda calculus with partial substitutions (2008), `http://www.pps.univ-paris-diderot.fr/~kesner/papers`
30. Kesner, D., Ventura, D.: Quantitative Types for Intuitionistic Calculi, `http://hal.archives-ouvertes.fr/hal-00980868/`
31. Kfoury, A.J., Wells, J.B.: Principality and type inference for intersection types using expansion variables. TCS 311(1-3), 1–70 (2004)
32. Krivine, J.-L.: A Call-by-Name Lambda-Calculus Machine. HOSC 20(3), 199–207 (2007)
33. Krivine, J.-L.: Lambda-calculus, types and models. Ellis, Horwood (1993)
34. Lengrand, S., Lescanne, P., Dougherty, D., Dezani-Ciancaglini, M., van Bakel, S.: Intersection types for explicit substitutions. IandC 189, 17–42 (2004)
35. Mascari, G., Pedicini, M.: Head linear reduction and pure proof net extraction. TCS 135(1), 111–137 (1994)

36. Milner, R.: Local bigraphs and confluence: Two conjectures (extended abstract). ENTCS 175(3), 65–73 (2007)
37. Neergaard, P., Mairson, H.: Types, Potency, and Idempotency: Why Nonlinearity and Amnesia Make a Type System Work. In: ICFP 2004, SIGPLAN Not., vol. 39(9), pp. 138–149 (2004)
38. Regnier, L.: Une équivalence e sur les lambda-termes. TCS 2(126), 281–292 (1994)
39. Ronchi Della Rocca, S.: Principal Type scheme and unification for intersection type discipline. TCS 59, 1–29 (1988)
40. Ronchi Della Rocca, S., Venneri, B.: Principal Type Scheme for an extended type theory. TCS 28, 151–169 (1984)
41. Urzyczyn, P.: The emptiness problem for intersection types. Journal of Symbolic Logic 64(3), 1195–1215 (1999)

Generic Partition Refinement Algorithms for Coalgebras and an Instantiation to Weighted Automata

Barbara König and Sebastian Küpper

Universität Duisburg-Essen, Germany

Abstract. Coalgebra offers a general framework for modelling different types of state-based systems. Our aim is to present generic algorithms to decide behavioural equivalence for coalgebras which generalize partition refinement. The underlying idea of the algorithms is to work on the final chain and to factor out redundant information. If the algorithm terminates, the result of the construction is a representative of the given coalgebra that is not necessarily unique and that allows to precisely answer questions about behavioural equivalence. We apply the algorithm to weighted automata over semirings in order to obtain a procedure for checking language equivalence for a large number of semirings.

1 Introduction

Coalgebra [Rut00] offers a unifying theory in which we can model and reason about various types of transition systems and automata. Given a category $\mathbf{C}$ and an endofunctor $F\colon \mathbf{C} \to \mathbf{C}$, an F-coalgebra is an arrow $\alpha\colon X \to FX$. In the case of $\mathbf{Set}$, X can be seen as the set of states of the transition system, F represents the branching type and α is the transition function. Depending on the choice of X we can specify different kinds of systems: for instance, the (finite) powerset functor $FX = \mathcal{P}_{fin}(X)$ specifies non-deterministic branching, whereas the functor $FX = X^A$ describes labels. Such functors can be combined to construct more complex types of transition systems.

Coalgebra comes equipped with a canonical notion of behavioural equivalence [Sta09]. We believe that an important contribution of coalgebra should be the provision of generic algorithms for checking behavioural equivalence, independent of the type of transition system. Such generic algorithms would be useful for two reasons: for classifying and comparing existing algorithms and for obtaining prototype algorithms (that might be further optimized) when studying a new class of systems. One example of such generic methods that have recently been studied are up-to techniques [RBB$^+$14,BPPR14].

Here we are interested in generic algorithms for checking behavioural equivalence, akin to minimization or partition refinement techniques. A rather general account for such minimization techniques has been presented in [ABH$^+$12], by using factorization structures and factoring the arrows into the final chain. For coalgebras over $\mathbf{Set}$ this boils down to classical partition refinement, encompassing minimization of deterministic automata [HU79] or the computation of bisimulation equivalence classes for probabilistic transition systems [LS89,Bai96]. In [ABH$^+$12] we have also shown how to handle some coalgebras in categories different from $\mathbf{Set}$, especially in Kleisli categories, which allow

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 311–325, 2014.

to specify side-effects and hence trace equivalence (where non-determinism is abstracted away as a side-effect).

However, some examples do not fall into this framework, most notably weighted automata over semirings, for which the underlying category does not possess suitable factorization structures. We found that a different notion of factorization can be used to capture behavioural equivalence. Furthermore it is unnecessary to look for a unique minimization or canonical representative, rather it is sufficient to compute *some* representative coalgebra and use it to precisely answer questions about behavioural equivalence. For weighted automata over semirings this yields an algorithm that we did not find in the literature as such. For probabilistic automata there is a related procedure for checking language equivalence [KMO+11], for fields a method is discussed in [Bor09] and a result for rings, based on results by Schützenberger, is given in [DK13]. The notion of coalgebra homomorphism is related to conjugacy described in [BLS06].

We will present a generic algorithm, based on the notion of equivalence classes of arrows. We will compare this algorithm to the algorithm of [ABH+12] that uses factorization structures [AHS90]. An important special case that we will discuss in detail and with many examples is the setting of weighted automata.

2 Preliminaries

We presuppose basic knowledge of category theory. We assume the reader is familiar with the notions of categories and functors. We are considering coalgebras in categories possibly different from $\mathbf{Set}$. However, in order to be able to speak about behavioural equivalence of states, we need to restrict to concrete categories, i.e. categories $\mathbf{C}$ with a faithful functor $U\colon \mathbf{C} \to \mathbf{Set}$, called concretization functor.

A coalgebra is an arrow $\alpha\colon X \to FX$ where the endofunctor F describes the branching type of the system under consideration, X plays the role of the set of states and α specifies transitions.

Definition 2.1 (Coalgebra). *Let $F : \mathbf{C} \to \mathbf{C}$ be an endofunctor on a category $\mathbf{C}$. An F-coalgebra is a pair $(X, \alpha : X \to FX)$, where X is an object of $\mathbf{C}$ and α is an arrow in $\mathbf{C}$. Given two F-coalgebras (X, α), (Y, β), a coalgebra homomorphism from (X, α) to (Y, β) is a $\mathbf{C}$-arrow $f : X \to Y$, so that $Ff \circ \alpha = \beta \circ f$.*

Coalgebra homomorphisms can be considered as structure-preserving maps between coalgebras, they correspond to functional bisimulations.

Definition 2.2 (Behavioural Equivalence). *Let $(\mathbf{C}, U)$ be a concrete category and $(X, \alpha : X \to FX)$ be an F-coalgebra. We call the elements $x \in UX$ the states of (X, α). Two states $x, y \in UX$ are* behaviourally equivalent *($x \sim y$) if and only if there exists an F-coalgebra (Y, β) and a coalgebra-homomorphism $f : (X, \alpha) \to (Y, \beta)$ such that $Uf(x) = Uf(y)$.*

We will now introduce some relations on objects and arrows.

Definition 2.3 (Relations on Objects and Arrows). *Let X, Y be two objects of a category $\mathbf{C}$. We write $X \leq Y$ whenever there is an arrow $f\colon X \to Y$. Furthermore we write $X \equiv Y$ whenever $X \leq Y$, $Y \leq X$.*

Let $a\colon X \to A$, $b\colon X \to B$ be two arrows in $\mathbf{C}$ with the same domain. We write $a \leq^X b$ whenever there exists an arrow $d\colon A \to B$ with $d \circ a = b$. Similarly we write $a \equiv^X b$.

If the objects of a category formed a set, $\leq$ would be a preorder (or quasi-order) and $\equiv$ would be an equivalence relation. That is, reflexivity, transitivity and symmetry hold (for $\equiv$). Note that if a category has a final object 1, then $X \leq 1$ holds for any other object X. Furthermore if $f\colon X \to 1$ is an arrow into the final object, we have that $g \leq^X f$ for any other arrow $g\colon X \to Y$.

Proposition 2.4. *Let X, Y be objects of a category $\mathbf{C}$ and let $a\colon X \to A$, $b\colon X \to B$ be arrows in $\mathbf{C}$. Furthermore let $F\colon \mathbf{C} \to \mathbf{C}$ be an endofunctor. Then (i) $X \leq Y$ implies $FX \leq FY$, (ii) $a \leq^X b$ implies $Fa \leq^{FX} Fb$, (iii) $a \leq^X b$ implies $a \circ c \leq^Y b \circ c$ for any arrow $c\colon Y \to X$.*

Example 2.5. Let $f\colon X \to A$, $g\colon X \to B$ where $X \neq \emptyset$ be two functions in $\mathbf{Set}$. It holds that $f \equiv^X g$ if and only if both functions induce the same partition on X, i.e., for all $x, y \in X$ it holds that $f(x) = f(y) \iff g(x) = g(y)$. Similarly $f \leq^X g$ means for all $x, y \in X$ that $f(x) = f(y) \Rightarrow g(x) = g(y)$.

Hence if $a \equiv^X b$ holds for two arrows $a\colon X \to A$, $b\colon X \to B$ in a concrete category $(\mathbf{C}, U)$, we can conclude that Ua, Ub induce the same partition on UX.

The notion of equivalent arrows is connected to the notion of split-mono. An arrow $m\colon X \to Y$ is called split-mono if it has a left inverse, i.e., if there exists an arrow $m^{\leftarrow}\colon Y \to X$ such that $m^{\leftarrow} \circ m = id_X$. Now, assume two arrows $a\colon X \to Y$, $b\colon X \to Z$ are equivalent ($a \equiv^X b$). Then there is an arrow $m\colon Y \to Z$ and an arrow $m^{\leftarrow}\colon Z \to Y$ such that $m \circ a = b$ and $m^{\leftarrow} \circ b = a$. Then, $m^{\leftarrow} \circ m \circ a = a$, hence m behaves like a split-mono, relative to a. More concretely, split-monos $m\colon X \to Y$ are exactly the arrows that are equivalent to id_X.

In Section 4 we will show that equivalence on arrows boils down to a very natural notion in the setting of weighted automata: the fact that two sets of vectors (with entries from a semiring) generate the same semimodule.

Finally, we need the notion of factorization structures [AHS90].

Definition 2.6 (Factorization Structures). *Let $\mathbf{C}$ be a category and let $\mathcal{E}$, $\mathcal{M}$ be classes of morphisms in $\mathbf{C}$. The pair $(\mathcal{E}, \mathcal{M})$ is called a* factorization structure *for $\mathbf{C}$ whenever*

- *$\mathcal{E}$ and $\mathcal{M}$ are closed under composition with isos.*
- *$\mathbf{C}$ has* $(\mathcal{E}, \mathcal{M})$-factorizations *of morphisms, i.e., each morphism f of $\mathbf{C}$ has a factorization $f = m \circ e$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$.*
- *$\mathbf{C}$ has the unique* $(\mathcal{E}, \mathcal{M})$-diagonalization property*: for each commutative square $g \circ e = m \circ f$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$ there exists a unique* diagonal*, i.e., a morphism d such that $d \circ e = f$ and $m \circ d = g$.*

3 Generic Algorithms

For our algorithms we assume in the following that the category under consideration has a final object 1. Before introducing the algorithms, based on the construction of the

final chain [AK95], we will first discuss how behavioural equivalence can be expressed as a post-fixpoint using the terminology of the previous section.

Proposition 3.1. *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra on $\mathbf{C}$. Furthermore let $f\colon X \to Y$ be an arrow. It holds that $f \leq^X Ff \circ \alpha$ (f is a post-fixpoint) if and only if there exists a coalgebra $\beta\colon Y \to FY$ such that f is a coalgebra homomorphism from (X, α) to (Y, β).*

For every such post-fixpoint f we have that $x, y \in UX$ and $Uf(x) = Uf(y)$ implies $x \sim y$. If, in addition, it holds for every other post-fixpoint $g\colon X \to Z$ that $g \leq^X f$ (f is the largest post-fixpoint), we can conclude that f induces behavioural equivalence, i.e., $Uf(x) = Uf(y) \iff x \sim y$.

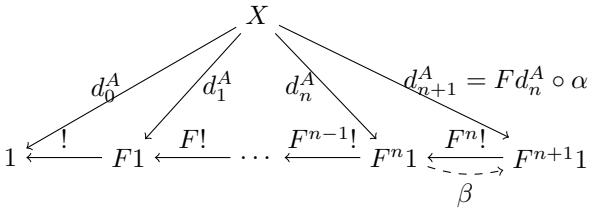
Proof. The first statement is almost trivial, since $f \leq^X Ff \circ \alpha$ means the existence of an arrow $\beta\colon X \to FX$ with $\beta \circ f = Ff \circ \alpha$, which is exactly the condition that β is a coalgebra homomorphism. Hence, by definition of behavioural equivalence, $Uf(x) = Uf(y)$ implies $x \sim y$. It is left to show that $x \sim y$ implies $Uf(x) = Uf(y)$ if f is the largest fixpoint. Since $x \sim y$, there must be some coalgebra $\gamma\colon Z \to FZ$ and a coalgebra homomorphism $g\colon X \to Z$ such that $\gamma \circ g = Fg \circ \alpha$ and $Ug(x) = Ug(y)$. This implies that $g \leq^X Fg \circ \alpha$ and hence $g \leq^X f$. Finally, we obtain $Uf(x) = Uf(y)$. □

In $\mathbf{Set}$ one can imagine the largest fixpoint $f\colon X \to Y$ as a function that maps every state into its equivalence class. We will now discuss three algorithms that expect a coalgebra α as input and yield, if they terminate, a coalgebra homomorphism (from α to some target coalgebra) that induces behavioural equivalence.

Algorithm 3.2 (Final Chain Algorithm A). *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. We define the following algorithm.*

Step 0**:** *Take the (unique) arrow $d_0^A\colon X \to 1$.*
Step $i+1$**:** *Compute $d_{i+1}^A = Fd_i^A \circ \alpha : X \to F^{i+1}1$.*
If there exists an arrow $\beta : F^i1 \to F^{i+1}1$ such that $\beta \circ d_i^A = d_{i+1}^A$, i.e., if $d_i^A \leq^X d_{i+1}^A$, the algorithm terminates and returns d_i^A and (F^i1, β) as its result.

Algorithm A is well-known iteration along the final chain, however, to our knowledge, the termination condition is new.



The algorithm is easy to analyse using the terminology introduced earlier. Specifically, it yields a sequence of arrows $d_0^A \geq^X d_1^A \geq^X d_2^A \geq^X \ldots$ that approximates behavioural equivalence from above. It terminates whenever this sequence becomes stationary, i.e., $d_n^A \equiv^X d_{n+1}^A$.

Lemma 3.3. *Let $g : X \to Z$ be any post-fixpoint, i.e. $g \leq^X Fg \circ \alpha$, then for all d_i^A obtained in Algorithm A we have $d_i^A \geq^X g$.*

Proof. Clearly $d_0^A \geq^X g$, because d_0^A is the arrow into the final object of the category. By induction, using Proposition 2.4, we can show that $d_i^A \geq^X g$ implies $d_{i+1}^A = Fd_i^A \circ \alpha \geq^X Fg \circ \alpha \geq^X g$. □

Proposition 3.4. *If Algorithm A terminates in step n, its result d_n^A induces behavioural equivalence, i.e. $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ud_n^A(x) = Ud_n^A(y)$.*

Proof. First, $d_0^A \geq^X d_1^A$ since d_0^A is an arrow into the final object. Since $\leq^X$ is preserved by functors and by right composition with arrows, we obtain by induction, using Proposition 2.4, that $d_i^A \geq^X d_{i+1}^A$. The termination condition says that $d_n^A \leq^X d_{n+1}^A = Fd_n^A \circ \alpha$. In order to conclude with Theorem 3.1 that the resulting arrow d_n^A induces behavioural equivalence, we have to show that it is the largest post-fixpoint. Assume another post-fixpoint $g\colon X \to Z$ with $g \leq^X Fg \circ \alpha$. Then, Theorem 3.3 shows that $d_n^A \geq^X g$ and thus d_n^A is the largest post-fixpoint. □

Furthermore whenever $d_n^A \equiv^X d_{n+1}^A$ we have that $d_n^A \equiv^X d_m^A$ for every $m \geq n$. Hence every arrow d_m^A obtained at a later step induces behavioural equivalence as well. We can show that d_n^A is equivalent to the arrow into the final coalgebra (if it exists).

Lemma 3.5. *Let (Z, κ) be the final coalgebra, i.e., a coalgebra into which there exists a unique coalgebra homomorphism from every other coalgebra. Take a coalgebra (X, α) and the unique homomorphism $f\colon (X, \alpha) \to (Z, \kappa)$ and assume that Algorithm A terminates in step n. Then $f \equiv^X d_n^A$.*

Note that after each step of the algorithm, it is permissible to replace d_i^A with any representative e_i^A of the equivalence class of d_i^A, i.e., any e_i^A with $d_i^A \equiv^X e_i^A$. This holds because $d_i^A \equiv^X e_i^A$ implies $d_{i+1}^A = Fd_i^A \circ \alpha \equiv^X Fe_i^A \circ \alpha$ and checking the termination condition $d_n^A \leq^X d_{n+1}^A$ can instead be done for any representatives of d_n^A, d_{n+1}^A. This gives rise to the following algorithm:

Algorithm 3.6 (Final Chain Algorithm B). *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Morover, let $\mathcal{R}$, the class of representatives, be a class of arrows of $\mathbf{C}$ such that for any arrow d in $\mathbf{C}$ we have an arrow $e \in \mathcal{R}$ that is equivalent to d, i.e. $d \equiv^X e$. We define the following algorithm:*

Step 0: *Take the (unique) arrow $d_0^B\colon X \to 1$.*

Step $i+1$: *Find a representative $e_i^B \in \mathcal{R}$, for d_i^B, i.e. factor $d_i^B = m_i^B \circ e_i^B$ such that $(e_i^B\colon X \to Y_i) \in \mathcal{R}$, $m_i^B\colon Y_i \to FY_{i-1}$. Determine $d_{i+1}^B = Fe_i^B \circ \alpha\colon X \to FY_i$. If there exists an arrow $\gamma\colon Y_i \to FY_i$ such that $\gamma \circ e_i^B = d_{i+1}^B$, i.e., $d_{i+1}^B \geq^X e_i^B$, the algorithm terminates and returns e_i^B and (Y_i, γ) as its result.*

Choosing good and compact representatives can substantially mitigate state space explosion. Hence, Algorithm B is an optimization of Algorithm A that (potentially) reduces the number of computations needed in every step. Moreover, Algorithm B terminates in exactly as many steps as Algorithm A (if it terminates).

Proposition 3.7. *Algorithm B terminates in n steps if and only if Algorithm A terminates in n steps. Furthermore two states $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ue_n^B(x) = Ue_n^B(y)$.*

Termination for Algorithm B is independent of the choice of the representatives e_i^B.

In [ABH+12] an algorithm similar to ours is being discussed. The algorithm also works on the final chain and uses a factorization system to trim the intermediate results with the aim of finding a so-called minimization, i.e., a unique and minimal representative of the given coalgebra α. We will discuss a variation of said algorithm.

Algorithm 3.8 (Final Chain Algorithm C). *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Morover, let $(\mathcal{E}, \mathcal{M})$ be a factorization structure such that U maps $\mathcal{M}$-morphisms to injections. We define the following algorithm:*

Step 0**:** *Take the (unique) arrow $d_0^C \colon X \to 1$.*
Step $i+1$**:** *Factor $d_i^C = m_i^C \circ e_i^C$ via the factorization structure, i.e. $(e_i^C \colon X \to Y_i) \in \mathcal{E}$, $(m_i^C \colon Y_i \to FY_{i-1}) \in \mathcal{M}$. Determine $d_{i+1}^C = Fe_i^C \circ \alpha \colon X \to FY_i$.*
If there exists an arrow $\delta \colon Y_i \to FY_i$ such that $\delta \circ e_i^C = d_{i+1}^C$, the algorithm terminates and returns e_i^C and (Y_i, δ) as its result.

Remark 3.9. Note that for Algorithms A and B we have to assume that the preorders are decidable, for Algorithms B and C we have to assume that the factorizations are computable. Naturally, in concrete applications, one has to choose a suitable class of representatives and a strategy for factoring. For the case of weighted automata we will discuss how such a strategy looks like (see Section 4).

If a category has a suitable factorization structure that ensures compact intermediate results, Algorithm C might terminate faster than Algorithm B. However, not every category has a suitable factorization structure (see our examples in Section 4), which drove us to investigate alternatives such as Algorithms A and B introduced earlier. For the trivial factorization structure where $\mathcal{E}$ is the class of all arrows and $\mathcal{M}$ are just the isomorphisms, we obtain the unoptimized Algorithm A. Algorithm C is a variation of the algorithm in [ABH+12] with a relaxed (but correct) termination condition: in the earlier version the algorithm terminated only if one of the m_i^C was an isomorphism.

Under certain conditions, stated below, Algorithm C terminates in the same number of steps as Algorithm B.

Proposition 3.10. *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Moreover, let $(\mathcal{E}, \mathcal{M})$ be a factorization structure for $\mathbf{C}$.*

1. *Assume that F preserves $\mathcal{M}$-arrows. If Algorithm A terminates in n steps, then Algorithm C terminates in n steps as well.*
2. *If Algorithm C terminates in n steps and for each arrow d_i^A, $i = 0, \ldots, n$ there exists an arrow $e_i \in \mathcal{E}$ with $e_i \equiv^X d_i^A$ (i.e., there is an arrow from $\mathcal{E}$ in all relevant equivalence classes), then Algorithm B terminates in n steps as well.*

Furthermore two states $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ue_n^C(x) = Ue_n^C(y)$.

In $\mathbf{Set}$ every equivalence class of arrows, except the equivalence classes of $\equiv^{\emptyset}$, contains a surjection. Hence if we choose surjections and injections as a factorization structure in $\mathbf{Set}$, we obtain exactly the same termination behaviour as the other two algorithms, provided that the state set is not empty. In this case, a suitable class $\mathcal{R}$ of representatives for Algorithm B would also be the surjections. This yields classical partition refinement algorithms.

Termination: The algorithm terminates whenever $\equiv^X$ has only finitely many equivalence classes. In $\mathbf{Set}$ this is the case for finite sets X. However, there are other categories where this condition holds for certain objects X: in $\mathbf{Vect}$, the category of vector spaces and linear maps, $\equiv^X$ is of finite index, whenever X is a finite-dimensional vector space.

Note that it is not sufficient to iterate until the partition induced by Ud_i is not refined from one step to the next. We will discuss weighted automata in the next section and the tropical semiring provides an example where behavioural equivalence is undecidable, but all steps of Algorithm B are computable and the partition induced by Ud_i can only be refined finitely often, rendering this criterion invalid.

4 Weighted Automata

We will show that we can apply our approach to weighted automata with weights taken from a semiring. Weighted automata [DKV09] are a versatile tool to specify and analyse systems equipped with quantitative information. They are a generalization of nondeterministic automata (NFA), but instead of just accepting or rejecting words over a given alphabet, weighted automata assign values taken from a semiring to words.

Coalgebraic treatment for weighted automata was already discussed in [BMS13], but for different categories. Here, we follow the ideas of [HJS07], which shows how to obtain trace or language equivalence by working in a Kleisli category.

We mainly consider Algorithm B. The algorithm will not terminate for every possible choice of the semiring (for example it is known that language equivalence is undecidable for tropical semirings). However, we will characterize the cases for which it terminates.

Definition 4.1 (Semiring). *Let S be a set. A* semiring *is a tuple $\mathbb{S} = (S, +, \cdot, 0, 1)$, where $0 \in S$, $1 \in S$ and $1 \neq 0$, $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, $0 \cdot a = a \cdot 0 = 0$ for all $a \in S$ and the distributive laws $(a + b) \cdot c = a \cdot c + b \cdot c$ and $c \cdot (a + b) = c \cdot a + c \cdot b$ hold for all $a, b, c \in S$. We will in the sequel identify $\mathbb{S}$ with the set S it is defined on.*

Hence a semiring is a ring that need not have additive inverses. All rings and therefore all fields are semirings. Whenever we are talking about a semiring throughout this paper, we assume $\mathbb{S}$ is a decidable set and the operations $\oplus$ and $\otimes$ can be computed effectively.

Example 4.2. The semiring $\mathbb{S} = (\mathbb{N} \cup \{\infty\}, min, +, \infty, 0)$, where $\mathbb{S}$-addition is the minimum and $\mathbb{S}$-multiplication is addition, is called the tropical semiring.

Take a partially ordered set $(\mathbb{L}, \leq)$ such that for $a, b \in \mathbf{L}$ we have a supremum $a \sqcup b$ and an infimum $a \sqcap b$. If $\mathbf{L}$ is a distributive lattice and we have top and bottom elements $\top$, $\bot$, we obtain a semiring $(\mathbb{L}, \sqcup, \sqcap, \bot, \top)$.

The usual notions of linear algebra (vector spaces, matrices, linear maps) can be extended from fields to semirings.

Definition 4.3 (Semimodule, Matrix). *Let X be an index set, let $\mathbb{S}$ be a semiring and let $\mathbb{S}^X$ the set of all functions $s\colon X \to \mathbb{S}$. Then $\mathbb{S}^X$ is a* semimodule*, i.e., a set closed under pointwise addition and multiplication with a constant. Every subset of $\mathbb{S}^X$ closed under these operations is called a* subsemimodule *of $\mathbb{S}^X$.*

An $X \times Y$-matrix a with weights over $\mathbb{S}$ is given by a function $a\colon X \times Y \to \mathbb{S}$, i.e., it is an element of $\mathbb{S}^{X \times Y}$, each row $x \in X$ there are only finitely many entries different from 0. We can multiply an $X \times Y$-matrix a and a $Y \times Z$-matrix b in the usual way, obtaining an $X \times Z$-matrix $a \cdot b$.

By $a_x\colon Y \to \mathbb{S}$, where $x \in X$, we denote the i-th row of a, i.e., $a_x(y) = a(x, y)$.

Similarly, for a generating set $G \subseteq \mathbb{S}^Y$ of vectors we denote by $\langle G \rangle$ the subsemimodule spanned by G, i.e., the set that contains all linear combinations of vectors of G. Given an $X \times Y$-matrix we denote by $\langle a \rangle \subseteq \mathbb{S}^Y$ the subsemimodule of $\mathbb{S}^Y$ that is spanned by the rows of a, i.e., $\langle a \rangle = \langle \{a_x \mid x \in X\} \rangle$.

Since matrices are arrows in the category, we represent them by lower case letters.

Definition 4.4 (Category of Matrices, Linear Maps). *We consider a category $\mathbf{M}(\mathbb{S})$ of $\mathbb{S}$-matrices where objects are sets and an arrow $a\colon Y \to X$ is an $X \times Y$-matrix a as defined above. Arrow composition is performed by matrix multiplication, i.e., for $a\colon X \to Y$, $b\colon Y \to Z$ we have $b \circ a = b \cdot a$. The identity arrow $id_X\colon X \to X$ is the $X \times X$ unit matrix.*

There exists a concretization functor U where $UX = \mathbb{S}^X$ and for $a\colon X \to Y$ $Ua : UX \to UY$ is the linear map from $\mathbb{S}^X$ to $\mathbb{S}^Y$ represented by the matrix a, i.e., $Ua(x) = a \cdot x$ for all $x \in \mathbb{S}^X$. (Note that x is considered as a vector and the multiplication of a matrix with a vector is defined as usual.)

If the semiring is a field, there exists a factorization structure which factors every matrix into a matrix of full row rank and a matrix of full column rank. This factorization is unique up-to isomorphism. However, for generic semirings, semimodule theory does not provide a similarly elegant notion of basis as in vector spaces, and such unique factorizations are not possible in general. Hence Algorithm C is usually not applicable.

We can now define weighted automata in a coalgebraic notation. Note that, different from the weighted automata in [DKV09], our automata do not have initial states, as it is customary in coalgebra, and hence no initial weights, but only final weights. However, for language equivalence we can easily simulate initial weights by adding a new state to the automaton with edges going to each state of the automaton, carrying the initial weights of these states and labelled with some (new) symbol.

Definition 4.5 (Weighted Automaton). *Let $\mathbf{M}(\mathbb{S})$ be the category defined above. Let A be a finite set of alphabet symbols. We define an endofunctor $F\colon \mathbf{M}(\mathbb{S}) \to \mathbf{M}(\mathbb{S})$ as*

follows: on objects[1] $FX = A \times X + 1$ *for a set* X*. For an arrow* $f\colon Y \to X$ *we have* $Ff\colon A \times Y + 1 \to A \times X + 1$ *where* $Ff((a,x),(a,y)) = f(x,y)$ *for* $a \in A$*,* $x \in X$*,* $y \in Y$*,* $Ff(\bullet,\bullet) = 1$ *and* $Ff(c,d) = 0$ *for all remaining* $c \in A \times X + 1, d \in A \times Y + 1$*.*

A weighted automaton *is an* F*-coalgebra, i.e., an arrow* $\alpha\colon X \to FX$ *in the category* $\mathbf{M}(\mathbb{S})$ *or, alternatively, an* $FX \times X$*-matrix with entries taken from* $\mathbb{S}$*.*

For a weighted automaton α, $\alpha(\bullet, x)$ denotes the final weight of state $x \in X$ and $\alpha((a,y),x)$ denotes the weight of the a-transition from x to y. We are mainly interested in weighted automata where the state set X is finite.

Definition 4.6 (Language of a Weighted Automaton). *Let* (X,α) *be a weighted automaton over alphabet* A*, a semiring* $\mathbb{S}$ *and a finite state set* X*. The language* $L_\alpha : A^* \to \mathbb{S}^X$ *of* α *is recursively defined as*

- $L_\alpha(\varepsilon)(x) = \alpha(\bullet, x)$
- $L_\alpha(aw)(x) = \sum_{x' \in X} \alpha((a,x'),x) \cdot L_\alpha(w)(x')$ *for* $a \in A$*,* $w \in A^*$

We will call $L_\alpha(w)(x)$ *the weight that state* x *assigns to the word* w*.*

Remark 4.7. Note that the language of a weighted automaton need not be defined if X is not finite, because coalgebras in $\mathbf{M}(\mathbb{S})$ need not be finitely branching and therefore computing the language of a state may depend on computing an infinite sum.

It can be shown that two states $x, y \in X$ of a finite weighted automaton (X,α) are behaviourally equivalent in the coalgebraic sense ($x \sim y$) if and only if they assign the same weight to all words, i.e. $L_\alpha(w)(x) = L_\alpha(w)(y)$ for all $w \in A^*$.

Proposition 4.8. *Let* $(X, \alpha\colon X \to FX)$*, where* X *is a finite set, be a weighted automaton over the finite alphabet* A *and the semiring* $\mathbb{S}$*. Then, two states* $x, y \in X$ *are language equivalent if and only if they are behaviourally equivalent.*

Due to the change of category (from $\mathbf{Set}$ to $\mathbf{M}(\mathbb{S})$) we obtain language equivalence instead of a notion of bisimilarity. Note that $\mathbf{M}(\mathbb{S})$ could also be considered as a Kleisli category over the semiring monad (see also [HJS07] which explains the effects that the implicit branching or side-effects of the monad have on behavioural equivalence).

From now onwards we consider only finite index sets. We will study the category $\mathbf{M}(\mathbb{S})$ and show what the preorder on arrows (see Definition 2.3) means in this setting.

Proposition 4.9. *Let* $a\colon X \to Y$*,* $b\colon X \to Z$ *be two arrows in* $\mathbf{M}(\mathbb{S})$*, i.e.,* a *is a* $Y \times X$*-matrix and* b *is a* $Z \times X$*-matrix. It holds that* $a \leq^X b \iff \langle a \rangle \subseteq \langle b \rangle$*. That is, two matrices* a, b *are ordered if and only if the subsemimodule spanned by* a *is included in the subsemimodule spanned by* b*. Hence also* $a \equiv^X b \iff \langle a \rangle = \langle b \rangle$*.*

We will now describe how to choose representatives in this category. Given an arrow $a\colon X \to Y$, i.e., a $Y \times X$-matrix, choose any set of vectors $G \subseteq \mathbb{S}^X$ that generates $\langle a \rangle$, i.e., $\langle a \rangle = \langle G \rangle$, and take any matrix a' with G as row vectors. Then $a \equiv^X a'$ and a' can be taken as a representative of a.

[1] Here $X + Y$ denotes the disjoint union of two sets X, Y and 1 stands for the singleton set $\{\bullet\}$.

Finding such a set G of generators that is minimal in size will be quite difficult for a general semiring, since semimodules do not have a notion of basis as elegant as vector spaces. However, for our optimization purposes, it is enough to make a' reasonably small. This can be done by going through the row vectors of a in any order, eliminating every vector that can be written as a linear combination of the remaining row vectors. We will use this strategy to determine representatives in $\mathcal{R}$ for Algorithm B, which are those matrices where no further eliminations are possible.

Definition 4.10 (Class of Representatives). *We define $\mathcal{R}$ as the class of all matrices a that do not contain a row that is a linear combination of the other rows of a.*

Our algorithm need not terminate though. Termination depends on the semiring and possibly on the automaton we investigate. It was already shown that language equivalence is not decidable for weighted automata with weights over the *tropical semiring* as shown in [Kro92,ABK11]. To state a termination condition, we define the following generating sets for a given weighted automaton (X, α): $S^n = \{L_\alpha(w) \mid w \in A^*, |w| \leq n\}$ and $S^* = \{L_\alpha(w) \mid w \in A^*\} = \bigcup_{n=0}^{\infty} S^n$. It can be shown that $S^* \subseteq \mathbb{S}^X$ is finitely generated if and only if there exists an index n such that $\langle S^n \rangle = \langle S^* \rangle$. (Note that $\langle S^* \rangle$ consists of all linear combinations of vectors of S^*. Hence if it has a finite generator set G, the vectors of G can again be composed of finitely many vectors of S^*. And these must be contained in some S^n.) We will show that Algorithm B terminates whenever $\langle S^* \rangle$ is finitely generated (see also [DK13,BMS13]). We start with the following lemma:

Lemma 4.11. *Let $d_i \colon X \to F^i 1$, $i \in \mathbb{N}_0$ be the sequence of arrows resp. matrices generated by the Algorithm A, where 1 (the final object) is the empty set. Hence $F^i 1$ contains tuples of the form $(a_1, a_2, \ldots, a_{i-1}, \bullet)$, where $a_j \in A$. We will identify such a tuple with the word $w = a_1 a_2 \ldots a_{i-1}$. Then, for an index $i \in \mathbb{N}_0$ and a word w with $|w| < i$, we have $(d_i)_w = L_\alpha(w)$. This means that $\langle d_{i+1} \rangle = \langle S^i \rangle$.*

Theorem 4.12. *If $\langle S^* \rangle$ for a weighted automaton (X, α) is finitely generated, Algorithm B terminates.*

Proof. From Theorem 4.11 it follows that $\langle d^A_{i+1} \rangle = \langle S^i \rangle$. Since $\langle S^* \rangle$ is finitely generated, there must be an index n with $\langle S^n \rangle = \langle S^{n+1} \rangle$, hence $\langle d^A_{n+1} \rangle = \langle d^A_{n+2} \rangle$. Theorem 4.9 then implies that $d^A_{n+1} \equiv^X d^A_{n+2}$ and hence $d^A_{n+1} \leq^X d^A_{n+2}$, which is exactly the termination condition. Note that this is not impaired when we consider representatives e_i in Algorithm B, since $e^B_{n+1} \leq^X e^B_{n+2}$ holds as well. In fact, a representative e^B_i generates the same subsemimodule as d^A_i. □

If $\mathbb{S}$ is a *field*, we basically obtain the algorithm for fields from [Bor09,ABH+12]. Note that for fields, minimal generating sets of semimodules, i.e. bases for vector spaces, always have the same size, so each intermediate result will be of the smallest size possible. Using a result by Droste and Kuske, [DK13], we can see that the algorithm also terminates for skew-fields, so structures that are almost fields, but where multiplication need not be commutative. In both cases, for a finite set X there are only $|X|$ different non-isomorphic semimodules, i.e., vector spaces. Thus any increasing chain has to become stationary and we can guarantee termination.

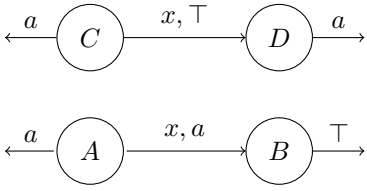
If $\mathbb{S}$ is a *finite semiring*, the algorithm terminates since there are only finitely many different row vectors of a fixed dimension $|X|$. If $\mathbb{S}$ is a *distributive complete lattice* or any other *commutative idempotent semiring*, the algorithm will terminate as well (because from the, obviously, finitely many weights contained in a finite automaton, only finitely many elements of $\mathbb{S}$ can be obtained via meet and join and thus there are again only finitely many different row vectors of a fixed dimension).

Note that termination does not necessarily imply decidability of language equivalence. For this it is also necessary to be able to decide whether two matrices are equivalent, i.e., whether a vector is generated by a given set of generators. This need not be decidable in general, but it is decidable in all the cases above.

We now consider several examples.

Example 4.13. We use as a semiring the complete distributive lattice $\mathbb{L} = \{\top, \bot, a, b\}$ where $\bot \leq a \leq \top$, $\bot \leq b \leq \top$ and consider the labelled transition system (X, α), $X = \{A, B, C, D\}$ with weights over $\mathbb{L}$ and labels from $A = \{x\}$ represented by the transition matrix and automaton (transitions with weight $\bot$ are omitted):

$$\alpha = \begin{array}{c} \\ (x,A) \\ (x,B) \\ (x,C) \\ (x,D) \\ \bullet \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} \bot & \bot & \bot & \bot \\ a & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \top & \bot \\ a & \top & a & a \end{pmatrix} \end{array}$$



We apply Algorithm B. Below e_i denotes the chosen representative in $\mathcal{R}$.

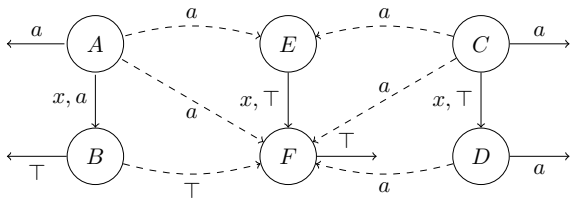
- We start with $d_0 = e_0$, a 0×4-matrix
- $Fe_0 = \begin{pmatrix} \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_1 = Fe_0 \cdot \alpha = \begin{pmatrix} a & \top & a & a \end{pmatrix}$. This is in $\mathcal{R}$, so $e_1 = d_1$.
- $Fe_1 = \begin{pmatrix} a & \top & a & a & \bot \\ \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_2 = Fe_1 \cdot \alpha = \begin{pmatrix} a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$. This is in $\mathcal{R}$, so $e_2 = d_2$.
- $Fe_2 = \begin{pmatrix} a & \bot & a & \bot & \bot \\ a & \top & a & a & \bot \\ \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_3 = Fe_2 \cdot \alpha = \begin{pmatrix} \bot & \bot & \bot & \bot \\ a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$.

  This is not in $\mathcal{R}$, we can for example see that the first row equals $\bot$ times the second plus $\bot$ times the third row. So we factorize:

$$d_3 = \gamma \cdot e_3 = \begin{pmatrix} \bot & \bot \\ \top & \bot \\ \bot & \top \end{pmatrix} \cdot \begin{pmatrix} a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$$

  Now $e_3 = e_2$ and we can stop our computation.

The resulting automaton is a two-state automaton, where the states will be called E, F. Looking at e_3 we can see that the states A and C of X are equivalent, since their columns coincide (in fact, both accept ε and x with weight a). States B and D on the other hand are not equivalent to any other state. We see that $e_2 : X \to Y$, where $Y = \{E, F\}$, is a coalgebra homomorphism from (X, α) to (Y, γ). The coalgebra (Y, γ) can be considered as a minimal representative of (X, α). The following diagram depicts automata (X, α) and (Y, γ) where the coalgebra homomorphism e_2 is drawn with dashed lines.



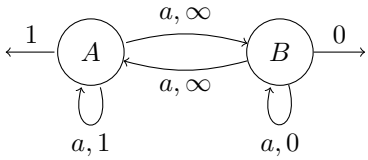
$$\gamma = \begin{matrix} \\ (x,E) \\ (x,F) \\ \bullet \end{matrix} \begin{matrix} E & F \end{matrix} \begin{pmatrix} \bot & \bot \\ \top & \bot \\ \bot & \top \end{pmatrix}$$

We can also see that our method of factoring is not unique, because in γ we could have chosen $\gamma((x,E),E) = b$ (and all other entries as before). In this case, there would be an x,b-loop on state E in the diagram above. Since all dashed arrows going into E carry weight a and $a \sqcap b = \bot$, this loop would not have any effect and the equivalence one obtains is the same.

In the next example, we will investigate the tropical semiring (cf. Theorem 4.2). We will write $\oplus$ and $\otimes$ for the $\mathbb{S}$-addition respectively the $\mathbb{S}$-multiplication to avoid confusion. Language equivalence is in general undecidable, hence the algorithm will not terminate in general.

Example 4.14. Consider the (rather simple) transition system over the one-letter alphabet $\{a\}$, given by the matrix α:

$$\begin{matrix} \\ (a,A) \\ (a,B) \\ \bullet \end{matrix} \begin{matrix} A & B \end{matrix} \begin{pmatrix} 1 & \infty \\ \infty & 0 \\ 1 & 0 \end{pmatrix}$$



Applying Algorithm B to α, we obtain the following (intermediate) results:

- $d_0 = e_0$ is the 0×2-matrix
- $Fe_0 = \begin{pmatrix} \infty & \infty & 0 \end{pmatrix}$, $d_1 = Fe_0 \cdot \alpha = \begin{pmatrix} 1 & 0 \end{pmatrix} = e_1$
- $d_2 = \begin{pmatrix} 2 & 0 \\ 1 & 0 \end{pmatrix} \in \mathcal{R}$, so we choose $e_2 = d_2$.
- $d_3 = \begin{pmatrix} 3 & 0 \\ 2 & 0 \\ 1 & 0 \end{pmatrix} \notin \mathcal{R}$, because we can obtain the second row as a linear combination of the first and the third row: $1 \otimes \begin{pmatrix} 1 & 0 \end{pmatrix} \oplus 0 \otimes \begin{pmatrix} 3 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \end{pmatrix} \oplus \begin{pmatrix} 3 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \end{pmatrix}$ However, we cannot obtain the first row via linear combination of the other two rows, so the algorithm cannot stop in the third iteration.
  So we can choose: $m_3 = \begin{pmatrix} 0 & \infty \\ 0 & 1 \\ \infty & 0 \end{pmatrix}$ and $e_3 = \begin{pmatrix} 3 & 0 \\ 1 & 0 \end{pmatrix}$
- From now on, each step is analogous to the third step, we obtain $e_i = \begin{pmatrix} i & 0 \\ 1 & 0 \end{pmatrix}$ in each iteration i, but we will never reach a d_{i+1} such that $d_{i+1} \equiv^X e_i$.

Algorithm B therefore does not terminate for α. However, since the two states are already separated from the first step onwards, we can at least conclude that they are not behaviourally equivalent. There are other example automata over this semiring for which Algorithm B does terminate.

5 Conclusion

Related Work: Our work is closely related to [ABH+12] which uses factorization structures in order to obtain generic algorithms for partition refinement. However, the algorithm in [ABH+12] could not handle general weighted automata over semirings, due to the absence of suitable factorization structures.

[Sta09] also discusses several coalgebraic methods to check behavioural equivalence, however the paper focusses more on the relation-based view with fixpoint iteration to obtain bisimulations. Staton compares with the final chain and can prove that whenever the arrow $F^i!$ in the final chain is a mono, then the relation refinement sequence converges at the latest. In our examples, the algorithm usually terminates earlier, since we only need a relative inverse β of $F^i!$ wrt. d_i^A.

For weighted automata, we are only aware of minimization algorithms for deterministic automata that do not work in the non-deterministic case. These algorithms are based on the idea of redistributing the weights of transitions in a canonical way and applying the minimization algorithm for deterministic automata (without weights) to the resulting automaton, where label and weight of each transition form a new transition label in the deterministic weight-free automaton. Mohri's algorithm [Moh97,Moh09] is based on weight pushing, and is applicable whenever $\mathbb{S}$ is zero-sum-free and weakly divisible. Eisner's algorithm [Eis03] works whenever $\mathbb{S}$ has multiplicative inverses. He remarks that his variation of weight pushing can also be applied to some semirings that do not have inverses, if they can be extended to ones that do have inverses. However, then, the minimal automaton might carry weights outside of $\mathbb{S}$.

Kiefer et al. [KMO+11] have investigated optimizations for the case of weighted automata with weights over the field $\mathbb{R}$, with applications to probabilistic automata. Their algorithm is a probabilistic optimization of an algorithm that enumerates words of length at most n, where n is the number of states, and their weights. Instead of checking every such word they use probabilistic methods to determine which words to check. As far as we know this method can not easily be generalized to arbitrary semirings.

Language equivalence is not decidable for every semiring. For instance, it was shown that language equivalence is undecidable for the tropical semiring [Kro92,ABK11].

Droste and Kuske [DK13] describe when equivalence of two weighted automata is decidable, based on earlier results by Schützenberger. Their decidability result is close to our own, only they work with weighted automata with initial weights. They show that whenever $\mathbb{S}$ is a ring such that every subsemimodule generated by an automaton is finitely generated, language equivalence is decidable. While their notation, using linear presentations, is different from ours and they concentrate only on rings, the underlying ideas are related (cf. Section 4 where we show termination if the subsemimodule generated by an automaton is finitely generated).

In [BLS06], which uses results for rational power series presented in [BR88], it is shown that for subsemirings of a field (such as $\mathbb{Z}$ or $\mathbb{N}$) weighted language equivalence is decidable. The paper uses the notion of conjugacy, which is strongly related to the notion of coalgebra homomorphism.

Ésik and Maletti have investigated proper semirings in [ÉM10] and have proven that language equivalence for weighted automata is decidable whenever the corresponding semiring is proper and effectively presentable. Furthermore they investigated

Noetherian semirings, i.e. semirings where every subsemimodule of a finitely generated $\mathbb{S}$-semimodule is finitely generated. However, they do not give a concrete algorithm.

Bonsangue, Milius and Silva [BMS13] have also investigated language equivalence of weighted automata in a coalgebraic setting, however they use Eilenberg-Moore instead of Kleisli categories. They present an axiomatization of weighted automata and give a set of sound and complete equational laws axiomatizing language equivalence.

Recently, in [UH14], Urabe and Hasuo studied simulation and language inclusion for weighted automata, were the coalgebraic theory provides the basis for an efficient implementation.

We did not find our procedure for checking language equivalence for weighted automata in the literature, but it may not be too surprising, given the related work. However, we believe that our contribution goes beyond providing a procedure for weighted automata: we give a uniform presentation of generic algorithms that can be applied to various kinds of transition systems. Furthermore we attempted to bridge the gap between coalgebraic methods and methods for weighted automata.

Future Work: For future work we plan to further investigate the issue of termination: are there more general criteria which guarantee termination for our algorithms? For this we will need a suitable notion of "finiteness" in a general categorical setting, for instance the notion of *finitely generated* or *locally presentable* [AR94].

Furthermore, when working with equivalence classes of arrows, it is necessary to find good representatives, in order to discard redundant information and make the representation more compact.

So far we have studied coalgebras over $\mathbf{Set}$ and weighted automata in this setting. Furthermore our theory also applies to other examples in [ABH$^+$12] such as non-deterministic automata and conditional transition systems. Naturally, we plan to investigate additional case studies.

In [FMT05], a procedure similar to our algorithm is used to minimize HD-automata. We plan to check whether this algorithm is an instance of our Algorithm B.

Acknowledgements. We would like to thank Alexandra Silva, Filippo Bonchi and Marcello Bonsangue for several interesting discussions on this topic.

References

ABH$^+$12. Adámek, J., Bonchi, F., Hülsbusch, M., König, B., Milius, S., Silva, A.: A coalgebraic perspective on minimization and determinization. In: Birkedal, L. (ed.) FOSSACS 2012. LNCS, vol. 7213, pp. 58–73. Springer, Heidelberg (2012)

ABK11. Almagor, S., Boker, U., Kupferman, O.: What's decidable about weighted automata? In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 482–491. Springer, Heidelberg (2011)

AHS90. Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories - The Joy of Cats. Wiley (1990)

AK95. Adámek, J., Koubek, V.: On the greatest fixed point of a set functor. Theoretical Computer Science 150, 57–75 (1995)

AR94. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. London Mathematical Society Lecture Note Series, vol. 189. Cambridge University Press (1994)

Bai96. Baier, C.: Polynomial time algorithms for testing probabilistic bisimulation and simulation. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 50–61. Springer, Heidelberg (1996)

BLS06. Béal, M.-P., Lombardy, S., Sakarovitch, J.: Conjugacy and equivalence of weighted automata and functional transducers. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 58–69. Springer, Heidelberg (2006)

BMS13. Bonsangue, M., Milius, S., Silva, A.: Sound and complete axiomatizations of coalgebraic language equivalence. ACM Transactions on Computational Logic 14(1) (2013)

Bor09. Boreale, M.: Weighted bisimulation in linear algebraic form. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 163–177. Springer, Heidelberg (2009)

BPPR14. Bonchi, F., Petrisan, D., Pous, D., Rot, J.: Coinduction up to in a fibrational setting. CoRR abs/1401.6675 (2014)

BR88. Berstel, J., Reutenauer, C.: Rational Series and their Languages. Springer (1988)

DK13. Droste, M., Kuske, D.: Weighted automata. To appear in Automata: from Mathematics to Applications. European Mathematical Society (2013)

DKV09. Droste, M., Werner, K., Vogler, H.: Handbook of Weighted Automata. Springer (2009)

Eis03. Mohri, M.: Weighted automata algorithms. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, pp. 213–254. Springer (2009)

ÉM10. Ésik, Z., Maletti, A.: Simulation vs. equivalence. In: Proc. of FCS 2010, pp. 119–124 (2010)

FMT05. Ferrari, G., Montanari, U., Tuosto, E.: Coalgebraic minimization of HD-automata for the π-calculus using polymorphic types. Theor. Comput. Sci. 331(2-3), 325–365 (2005)

HJS07. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Computer Science 3(4:11), 1–36 (2007)

HU79. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages and computation. Addison Wesley, Reading (1979)

KMO$^+$11. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: Language equivalence for probabilistic automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 526–540. Springer, Heidelberg (2011)

Kro92. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 101–112. Springer, Heidelberg (1992)

LS89. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing (preliminary report). In: Proc. of POPL 1989, pp. 344–352. ACM (1989)

Moh97. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics 23, 269–311 (1997)

Moh09. Mohri, M.: Weighted automata algorithms. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, pp. 213–254. Springer (2009)

RBB$^+$14. Rot, J., Bonchi, F., Bonsangue, M., Rutten, J., Pous, D., Silva, A.: Enhanced coalgebraic bisimulation. In: Mathematical Structures in Computer Science (to appear, 2014)

Rut00. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)

Sta09. Staton, S.: Relating coalgebraic notions of bisimulation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 191–205. Springer, Heidelberg (2009)

UH14. Urabe, N., Hasuo, I.: Generic forward and backward simulations III: Quantitative simulations by matrices. In: Proc. of CONCUR 2014. LNCS/ARCoSS. Springer (to appear, 2014)

Looking at Separation Algebras with Boolean BI-eyes*

Dominique Larchey-Wendling[1] and Didier Galmiche[2]

[1] LORIA – CNRS, Nancy, France
dominique.larchey-wendling@loria.fr
[2] LORIA – Université de Lorraine, Nancy, France
didier.galmiche@loria.fr

Abstract. In this paper, we show that the formulæ of Boolean BI cannot distinguish between some of the different notions of separation algebra found in the literature: partial commutative monoids, either cancellative or not, with a single unit or not, all define the same notion of validity. We obtain this result by the careful study of the specific properties of the counter-models that are generated by tableaux proof-search in Boolean BI.

1 Introduction

Separation logic [18] is a well established logical formalism for reasoning about heaps of memory and programs that manipulate them. The purely propositional part of the logic is usually given by Boolean BI (also denoted BBI) which is a particular *bunched logic* obtained by freely combining the Boolean connectives of classical propositional logic with those of multiplicative intuitionistic linear logic [11]. Provability in BBI is defined by a Hilbert system [17] and corresponds to validity in the class of non-deterministic (or relational) monoids [8]. Restricting that class to e.g. partial monoids gives another notion of validity [14] for which the Hilbert system is not complete anymore.

Separation logic is defined by a particular kind of partial monoids built for instance from memory heaps that are composed by disjoint union; see [3,13,15] for a survey of the different models either abstract or concrete that are usually considered in the literature. These models verify some additional properties that may be invalid in non-deterministic models or even in partial monoidal models. Some of these properties are the foundation of *separation algebras* [5,6,7]. For instance, the existence of *multiple units* for the composition of heaps, or the property that the composition of heaps is a *cancellative* operation, the main focus of this paper. This last property does not hold in an arbitrary partial monoid.

Let us discuss some motivations behind the study of these specific properties of separation algebras. Abstract separation logics and variants of BBI are usually undecidable [3,2,14,15]. But still, being able to prove statements expressed in BBI is required in the framework of Hoare logic. Hence the idea is to try narrowing down the logic and the separation model through the logical

* Work partially supported by the ANR grant DynRes (project No. ANR-11-BS02-011).

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 326–340, 2014.

or proof-theoretical representations of the specific properties of separation algebras. We notice the lively interest in proof-search for relational BBI [1,10,16], partial monoidal BBI [12,13] and propositional abstract separation logic [9].

In [4], Brotherston and Villard show that cancellativity cannot be axiomatized within BBI: no formula of BBI is able to distinguish cancellative from non-cancellative monoids. Let us note that even though an axiomatization is proposed in some hybrid extension of BBI [4], proof-search in such extensions of BBI is a largely unexplored track of research. In the current paper, we show the stronger result that any BBI formula that is valid in partial and cancellative models is also valid in any partial model: validity of BBI formulæ is the very same predicate if you add cancellativity as a requirement for your models.

In [9], Hóu *et al.* present a labelled sequent calculus for proof-search in *propositional abstract separation logic* extending their work on relational BBI [10] by introducing model specific proof-rules, in particular one for partiality and one for cancellativity. A noticeable consequence of our result is that their rule for cancellativity is redundant when searching for proofs of BBI-formulæ: one may find shorter proofs using that rule but it does not reinforce provability. As another consequence, extending the older labelled tableaux calculus for partial monoidal BBI of Larchey-Wendling and Galmiche [13] to cover cancellativity is trivial: simply do nothing. The difficulty does not lie in the extension of the system but in the proof of the redundancy of cancellativity.

The results obtained in this paper emphasize the importance of the *strong completeness theorem* for partial monoidal BBI [12] from which they derive. The counter-models generated by the labelled tableaux proof-search calculus contain information about the logic itself that, when carefully extracted, can be used to obtain completeness for additional properties of abstract models.

Let us give an overview of the paper. In Section 2, we recall the syntax and Kripke semantics of Boolean BI and we present *non-deterministic monoids* which are the models of BBI, and some sub-classes of monoids related to separation algebras and abstract separation logic models, e.g. cancellative monoids. In Section 3, we study the links between *single unit* and *multi-unit monoids* and give a quick semantic overview of why they are equivalent w.r.t. BBI validity. In Section 4, we define the notion of *partial monoidal equivalence* (or PME for short) to syntactically represent partial monoids with a single unit. We define basic and simple PMEs which are the monoids that are generated by labelled tableaux proof-search [12]. In Section 5, we use the strong completeness result for simple PMEs to derive an equivalence theorem for some separation algebras. It is based on our core result: *basic/simple* PME*s are cancellative and have invertible squares*. We discuss the proof of this result in the following sections. In Section 6, we introduce the notion of invertibility in the context of PMEs. In Section 7, we argue that even though basic PMEs are defined inductively, it is not possible to give a direct inductive proof of cancellativity or of the invertibility of squares for basic PMEs. In Section 8, we show that basic PMEs can be transformed into primary PMEs and that primary PMEs are cancellative with invertible squares. *Omitted proofs can be found in the appendices.*

2 Boolean BI and Its Non-deterministic Kripke Semantics

In this section, we introduce a "compact" syntax for BBI: conjunction $\wedge$ and negation $\neg$ are the only Boolean connectives.[1] Then, we present the Kripke semantics of BBI based on the notion of non-deterministic monoid.

Definition 1. *The* formulæ of BBI *are freely built using logical variables in* Var*, the logical constant* $\mathbb{I}$*, the unary connective* $\neg$ *or binary connectives in* $\{*, -\!\!*, \wedge\}$*. The formal grammar is* $F ::= v \mid \mathbb{I} \mid \neg F \mid F \wedge F \mid F * F \mid F -\!\!* F$ *with* $v \in \mathsf{Var}$*.*

We introduce the semantic foundations of BBI. Let us consider a set M.[2] We denote by $\mathcal{P}(M)$ the power-set of M, i.e. its set of subsets. A binary function $\circ : M \times M \longrightarrow \mathcal{P}(M)$ is naturally extended to a binary operator on $\mathcal{P}(M)$ by $X \circ Y = \bigcup\{x \circ y \mid x \in X, y \in Y\}$ for any subsets X, Y of M. Using this extension, we can view an element m of M as the singleton set $\{m\}$ and derive equations like $m \circ X = \{m\} \circ X$, $a \circ b = \{a\} \circ \{b\}$ or $\emptyset \circ X = \emptyset$.

Definition 2. *A* non-deterministic monoid *(*ND*-monoid for short) is a triple* $\mathfrak{M} = (M, \circ, U)$ *where* $U \subseteq M$ *is the* set of units *and* $\circ : M \times M \longrightarrow \mathcal{P}(M)$ *is a* composition *for which the axioms of* (neutrality) $\forall x \in M\ x \circ U = \{x\}$, (commutativity) $\forall x, y \in M\ x \circ y = y \circ x$, *and* (associativity)[3] $\forall x, y, z \in M\ (x \circ y) \circ z = x \circ (y \circ z)$ *hold.*

The extension of $\circ$ to $\mathcal{P}(M)$ thus induces a (usual) commutative monoidal structure with unit U on $\mathcal{P}(M)$. The term *non-deterministic* was introduced in [8] in order to emphasize the fact that the composition $a \circ b$ may yield not only one but an arbitrary number of results including the possible incompatibility of a and b in which case $a \circ b = \emptyset$. Notice that $\mathfrak{M}$ is called a BBI-*model* in [4].

Given $\mathfrak{M} = (M, \circ, U)$ and an interpretation $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(M)$ of variables, we define the Kripke forcing relation by induction on the structure of formulæ:

$$\mathfrak{M}, x \Vdash_\delta v \text{ iff } x \in \delta(v) \qquad \mathfrak{M}, x \Vdash_\delta \mathbb{I} \text{ iff } x \in U \qquad \mathfrak{M}, x \Vdash_\delta \neg A \text{ iff } \mathfrak{M}, x \nVdash_\delta A$$
$$\mathfrak{M}, x \Vdash_\delta A \wedge B \text{ iff } \mathfrak{M}, x \Vdash_\delta A \text{ and } \mathfrak{M}, x \Vdash_\delta B$$
$$\mathfrak{M}, x \Vdash_\delta A * B \text{ iff } \exists a, b,\ x \in a \circ b \text{ and } \mathfrak{M}, a \Vdash_\delta A \text{ and } \mathfrak{M}, b \Vdash_\delta B$$
$$\mathfrak{M}, x \Vdash_\delta A -\!\!* B \text{ iff } \forall a, b,\ (b \in x \circ a \text{ and } \mathfrak{M}, a \Vdash_\delta A) \Rightarrow \mathfrak{M}, b \Vdash_\delta B$$

Definition 3 (BBI-validity, Counter-Models). *A formula* F *of* BBI *is* valid *in* $\mathfrak{M} = (M, \circ, U)$ *if for any interpretation* $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(M)$ *the relation* $\mathfrak{M}, m \Vdash_\delta F$ *holds for any* $m \in M$*. A* counter-model *of the formula* F *is given by a* ND*-monoid* $\mathfrak{M}$*, an interpretation* $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(M)$*, and an element* $m \in M$ *such that* $\mathfrak{M}, m \nVdash_\delta F$*.*

In some papers, you might find BBI defined by non-deterministic monoidal Kripke semantics [1,4,8,10], in other papers it is defined by partial deterministic monoidal Kripke semantics [12,13] and generally separation logic models are particular instances of partial (deterministic) monoids [3,4,9]. See [13] for a general discussion about these issues.

[1] The other Boolean connectives can be obtained by De Morgan's laws.
[2] The case $M = \emptyset$ is *allowed* but arguably not very interesting in the case of BBI.
[3] Associativity should be understood using the extension of $\circ$ to $\mathcal{P}(M)$.

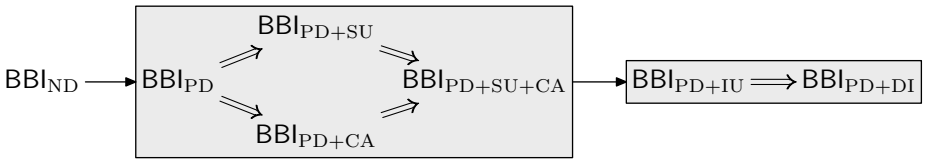


Fig. 1. Inclusions between BBI-validity in some sub-classes of ND-monoids

Definition 4. *For any* ND*-monoid* $(M, \circ, U)$*, we name some properties as follows:*

(PD)	*Partial deterministic*	$\forall x, y, a, b \ \{x, y\} \subseteq a \circ b \Rightarrow x = y$
(SU)	*Single unit*	$\exists u \ U = \{u\}$
(CA)	*Cancellativity*	$\forall k, a, b \ (k \circ a) \cap (k \circ b) \neq \emptyset \Rightarrow a = b$
(IU)	*Indivisible units*	$\forall x, y \ x \circ y \cap U \neq \emptyset \Rightarrow x \in U$
(DI)	*Disjointness*	$\forall x \ x \circ x \neq \emptyset \Rightarrow x \in U$

These properties allow us to consider sub-classes of the full class of ND-monoids. Other properties like *divisibility* or *cross-split* are considered as well in [4] but in this paper, we focus on the properties of Definition 4.

We denote by ND the full class of non-deterministic monoids. We identify the property X with the sub-class $X \subseteq \mathrm{ND}$ of monoids which satisfy property X. If X and Y are two properties, we read $X + Y$ as the sub-class of monoids of ND that satisfy the *conjunction of* X *and* Y. This is the meaning of the equation $X + Y = X \cap Y$ which might look strange at first. As an example, $\mathrm{PD} + \mathrm{SU} + \mathrm{CA} + \mathrm{IU}$ is both the conjunction of those four properties and the sub-class of cancellative partial deterministic monoids with a single and indivisible unit.

Proposition 1. *The two strict inclusions* $\mathrm{DI} \subsetneq \mathrm{IU}$ *and* $\mathrm{PD} + \mathrm{DI} \subsetneq \mathrm{PD} + \mathrm{IU}$ *hold.*

The sub-class HM of *heap monoids* verifies all the properties of Definition 4. However, it is not defined by a property but it is described by the concrete models of Separation Logic [15].

Various notions of *separation algebra* can be found in the literature: for instance the "original" notion of separation algebra is defined in [5] as the elements of the sub-class $\mathrm{PD} + \mathrm{SU} + \mathrm{CA}$; in the "views" framework of [6], a separation algebra is an element of sub-class PD; while it is of sub-class $\mathrm{PD} + \mathrm{CA}$ in [7]. To finish, in [13], though not called separation algebra, a BBI-model is an element of sub-class $\mathrm{PD} + \mathrm{SU}$.

In general the sub-classes of ND define different notions of validity on the formulæ of BBI [14]. However, it was proved recently that theses properties are not axiomatizable in BBI [4], *with the exception of* IU.[4] We define a notation to express the relations between those potentially different notions of validity.

Definition 5 (BBI_X). *For any sub-class* $X \subseteq \mathrm{ND}$*, we denote by* BBI_X *the set of formulæ of* BBI *which are valid in any* ND*-monoid of the sub-class* X*.*

[4] In [4], $\mathbb{I} \to (A * B) \to A$ is used as a BBI-axiom for IU but we favor $\neg(\mathbb{I} \wedge (\neg\mathbb{I} * \neg\mathbb{I}))$.

Obviously, if the inclusion $X \subseteq Y$ holds between the sub-classes X and Y of ND-monoids then inclusion $\mathsf{BBI}_Y \subseteq \mathsf{BBI}_X$ holds between the sets of valid formulæ. The sets BBI_X are usually not recursive (at least for the sub-classes we consider here) because of the undecidability of BBI [3,2,14,15]. The identity $\mathsf{BBI}_X = \mathsf{BBI}_Y$ implies for instance that a semi-decision algorithm for validity (of formulæ) in sub-class X can be replaced by some semi-decision algorithm for validity in sub-class Y. It also "suggests" that there might exist some kind of relation (like a map [4] or a bisimulation [15]) between the models of sub-class X and those of sub-class Y.[5]

To the best of our knowledge, the graph of Figure 1 summarizes what was known about the inclusion relations between the formulæ valid in the previously mentioned sub-classes of ND-monoids, the single arrow $\rightarrow$ representing strict inclusion, the double arrow $\Rightarrow$ representing non-strict inclusion. In fact, besides trivial inclusion results derived from the obvious inclusions of sub-classes of monoids, not very much was known except the strict inclusion $\mathsf{BBI}_{\mathrm{ND}} \subsetneq \mathsf{BBI}_{\mathrm{PD}}$ proved[6] in [14] and the strict inclusions $\mathsf{BBI}_{\mathrm{ND}} \subsetneq \mathsf{BBI}_{\mathrm{IU}}$ and $\mathsf{BBI}_{\mathrm{PD}} \subsetneq \mathsf{BBI}_{\mathrm{PD+IU}}$ which are trivial consequences of the stronger result that IU can be axiomatized in BBI. Beware that PD *cannot* be axiomatized in BBI [4].

The left gray box in Figure 1 is the main motivation behind the current paper. It contains the four different definitions of separation algebras mentioned earlier: PD, PD + SU, PD + CA and PD + SU + CA. In this paper, we show that these four sub-classes of ND-monoids define the same set of valid formulæ, i.e. the double arrows are in fact identities. To obtain these results, we first give a simple proof that $\mathsf{BBI}_{\mathrm{PD+SU}} \subseteq \mathsf{BBI}_{\mathrm{PD}}$ in Section 3, and then a much more involved proof that $\mathsf{BBI}_{\mathrm{PD+SU+CA}} \subseteq \mathsf{BBI}_{\mathrm{PD+SU}}$ in the latest sections of the paper. This proof is based on a careful study of the properties of the counter-models generated by proof-search, which are complete for $\mathsf{BBI}_{\mathrm{PD+SU}}$ [12].

The right gray box in Figure 1 is a secondary focus of our paper. We prove the identities $\mathsf{BBI}_{\mathrm{PD+IU}} = \mathsf{BBI}_{\mathrm{PD+DI}} = \mathsf{BBI}_{\mathrm{PD+SU+CA+IU+DI}}$ by exploiting the fact that the counter-models generated by proof-search which satisfy property IU also satisfy property DI.

3 Single Units in Non-deterministic Monoids

We give a quick overview of the relations between the multi-unit semantics and the single unit semantics. We recall that they define the same notion of validity for BBI and we give a model-theoretic account of this equivalence. Soundness/completeness for the single unit semantics w.r.t. the Hilbert proof system for BBI were already established in [8].[7]

Definition 6 (The unit of x). *Let $(M, \circ, U)$ be a* ND*-monoid. For any $x \in M$, there exists a unique $u_x \in U$ such that $x \circ u_x = \{x\}$. It is called* the unit of x.

[5] Relation from which a constructive proof of $\mathsf{BBI}_X = \mathsf{BBI}_Y$ could be derived.

[6] In fact only $\mathsf{BBI}_{\mathrm{SU}} \subsetneq \mathsf{BBI}_{\mathrm{PD+SU}}$ is proved in [14] but the *same argument* will do.

[7] The *same proof* works for the more general multi-unit semantics, as assumed for instance in Theorem 2.5 of [4]. Hence the identity $\mathsf{BBI}_{\mathrm{ND}} = \mathsf{BBI}_{\mathrm{SU}}$ was known since [8].

Definition 7 (Slice Monoid at x). *Let $\mathfrak{M} = (M, \circ, U)$ be a* ND*-monoid and let $x \in M$. Then the triple $\mathfrak{M}_x = (M_x, \circ', \{u_x\})$ is a* ND*-monoid of sub-class* SU *where $M_x = \{k \in M \mid u_k = u_x\}$ and $\circ'$ is the restriction of $\circ$ to M_x which is defined on $M_x \times M_x$ by $u \circ' v = u \circ v$. The triple $\mathfrak{M}_x$ is called the* slice monoid at x.

Lemma 1. *Let $\mathfrak{M} = (M, \circ, U)$ be a* ND*-monoid, $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(M)$ and $x \in M$. Let us consider $\mathfrak{M}_x$, the slice monoid at x and let $\delta' : \mathsf{Var} \longrightarrow \mathcal{P}(M_x)$ be defined by $\delta'(z) = \delta(z) \cap M_x$ for any $z \in M_x$. For any formula F of* BBI *and any $z \in M_x$, we have $\mathfrak{M}, z \Vdash_\delta F$ iff $\mathfrak{M}_x, z \Vdash_{\delta'} F$.*

Theorem 1. *If $K \subseteq$ ND is a sub-class of* ND*-monoids closed under slicing, then $\mathsf{BBI}_K = \mathsf{BBI}_{K+\mathrm{SU}}$ holds. In particular, $\mathsf{BBI}_{\mathrm{ND}} = \mathsf{BBI}_{\mathrm{SU}}$ and $\mathsf{BBI}_{\mathrm{PD}} = \mathsf{BBI}_{\mathrm{PD}+\mathrm{SU}}$.*

Remark: the property SU cannot be axiomatized in BBI [4]. The identity $\mathsf{BBI}_{\mathrm{ND}} = \mathsf{BBI}_{\mathrm{SU}}$ gives another proof argument for this result.

4 Partial Monoidal Equivalences

We recall the framework of labels and constraints that is used to syntactically represent partial monoids of sub-class PD + SU which form the semantic basis of partial monoidal Boolean BI. The section is a short reminder of the theory developed in [13] where a labelled tableaux system is introduced and its soundness w.r.t. the sub-class PD + SU is established. Moreover, the (strong) completeness of this tableaux system is proved in [12] and this crucial (albeit non-constructive) result is restated here as Theorem 2.

4.1 Words, Constraints, PMEs and the Sub-class PD + SU

Let $L^\star$ be the set of *finite multisets of letters* of the alphabet L. We call the elements of $L^\star$ *words*; they do not account for the order of letters. The composition of words is denoted *multiplicatively*[8] and the *empty word* is denoted ϵ. Hence $(L^\star, \cdot, \epsilon)$ is the (usual) commutative monoid freely generated by L.

We view the alphabet L or any of its subsets $X \subseteq L$ as a subset $X \subsetneq L^\star$, i.e. we assume letters as one-letter words. We denote $x \prec y$ when x is a *sub-word* of y (i.e. $\exists k, xk = y$). If $x \prec y$, the unique k such that $xk = y$ is denoted y/x and we have $y = x(y/x)$. The *carrier alphabet* of a word m is $\mathcal{A}_m = \{\mathsf{c} \in L \mid \mathsf{c} \prec m\}$.

A *constraint* is an ordered pair of words in $L^\star \times L^\star$ denoted $m \rightharpoonup n$. A binary relation $R \subseteq L^\star \times L^\star$ between words of $L^\star$ is a set of constraints, hence $x \; R \; y$ is a shortcut for $x \rightharpoonup y \in R$. The *language* of a binary relation $R \subseteq L^\star \times L^\star$ denoted $\mathcal{L}_R$ is defined by $\mathcal{L}_R = \{x \in L^\star \mid \exists m, n \in L^\star \text{ s.t. } xm \; R \; n \text{ or } m \; R \; xn\}$. The *carrier alphabet* of R is $\mathcal{A}_R = \bigcup\{\mathcal{A}_m \cup \mathcal{A}_n \mid m \; R \; n\}$.

A word $m \in L^\star$ is said to be *defined in* R if $m \in \mathcal{L}_R$ and is *undefined in* R otherwise. A letter $\mathsf{c} \in L$ is *new to* R if $\mathsf{c} \notin \mathcal{A}_R$. The language $\mathcal{L}_R$ is downward closed w.r.t. the sub-word order $\prec$. The inclusion $\mathcal{L}_R \subseteq \mathcal{A}_R^\star$ and the identity

[8] The additive notation + would conflict with the $\rightharpoonup$ sign later used for constraints.

$\mathcal{A}_R = \mathcal{L}_R \cap L$ hold. If R_1 and R_2 are two relations such that $R_1 \subseteq R_2$ then the inclusions $\mathcal{A}_{R_1} \subseteq \mathcal{A}_{R_2}$ and $\mathcal{L}_{R_1} \subseteq \mathcal{L}_{R_2}$ hold. Let us define the particular sets of constraints/relations we are interested in.

Definition 8 (PME). *A* partial monoidal equivalence *(*PME *for short) over the alphabet L is a binary relation $\sim \subseteq L^\star \times L^\star$ which is closed under the rules $\langle \epsilon, s, c, d, t \rangle$:*

$$\frac{}{\epsilon \rightharpoonup \epsilon}\,\langle\epsilon\rangle \quad \frac{x \rightharpoonup y}{y \rightharpoonup x}\,\langle s\rangle \quad \frac{ky \rightharpoonup ky \quad x \rightharpoonup y}{kx \rightharpoonup ky}\,\langle c\rangle \quad \frac{xy \rightharpoonup xy}{x \rightharpoonup x}\,\langle d\rangle \quad \frac{x \rightharpoonup y \quad y \rightharpoonup z}{x \rightharpoonup z}\,\langle t\rangle$$

Proposition 2. *Any* PME $\sim$ *is also closed under the (derived) rules $\langle p_l, p_r, e_l, e_r \rangle$:*

$$\frac{kx \rightharpoonup y}{x \rightharpoonup x}\,\langle p_l\rangle \quad \frac{x \rightharpoonup ky}{y \rightharpoonup y}\,\langle p_r\rangle \quad \frac{x \rightharpoonup y \quad yk \rightharpoonup m}{xk \rightharpoonup m}\,\langle e_l\rangle \quad \frac{x \rightharpoonup y \quad m \rightharpoonup yk}{m \rightharpoonup xk}\,\langle e_r\rangle$$

and the identities $\mathcal{L}_\sim = \{x \in L^\star \mid x \sim x\}$ and $\mathcal{A}_\sim = \{\mathsf{c} \in L \mid \mathsf{c} \sim \mathsf{c}\}$ hold.

See [13] for a proof of Proposition 2. These derived rules will be more suitable for proving properties of PMEs throughout this paper. Rule $\langle p_l \rangle$ (resp. $\langle p_r \rangle$) is a left (resp. right) projection rule. Rules $\langle e_l \rangle$ and $\langle e_r \rangle$ express the possibility to exchange related sub-words inside the PME $\sim$, either on the left or on the right.

Definition 9. *A* PME is cancellative *(resp.* has indivisible units*, resp.* has disjointness*) if it is closed under rule $\langle ca \rangle$ (resp. rule $\langle iu \rangle$, resp. rule $\langle di \rangle$).*[9]

$$\frac{kx \rightharpoonup ky}{x \rightharpoonup y}\,\langle ca\rangle \qquad \frac{\epsilon \rightharpoonup xy}{\epsilon \rightharpoonup x}\,\langle iu\rangle \qquad \frac{xx \rightharpoonup xx}{\epsilon \rightharpoonup x}\,\langle di\rangle$$

Let us see how the rules $\langle ca \rangle$, $\langle iu \rangle$ and $\langle di \rangle$ relate to sub-classes CA, IU and DI. Let $\sim$ be a PME over L. The relation $\sim$ is a partial equivalence on $L^\star$ by rules $\langle s \rangle$ and $\langle t \rangle$. The partial equivalence class of a word x is $[x] = \{y \mid x \sim y\}$. The partial quotient $L^\star/\!\sim$ is the set of non-empty classes $L^\star/\!\sim = \{[x] \mid x \sim x\}$. We define a non-deterministic composition on $L^\star/\!\sim$ by $[z] \in [x] \bullet [y]$ iff $z \sim xy$.

Proposition 3. *The triple $\mathfrak{M}_\sim = (L^\star/\!\sim, \bullet, \{[\epsilon]\})$ is a* ND*-monoid of sub-class* PD + SU*. $\mathfrak{M}_\sim$ is of sub-class* CA *(resp. sub-class* IU*, resp. sub-class* DI*) if and only if $\sim$ is closed under rule $\langle ca \rangle$ (resp. rule $\langle iu \rangle$, resp. rule $\langle di \rangle$).*

4.2 Generated PME, Basic PME Extensions and Simple PMEs

Defined by closure under some deduction rules, the class of PMEs over an alphabet L is thus closed under arbitrary intersections. Let $\mathcal{C}$ be a set of constraints over the alphabet L. The PME *generated by* $\mathcal{C}$ is the least PME containing $\mathcal{C}$. It is either denoted by $\sim_{\mathcal{C}}$ or $\overline{\mathcal{C}}$ and the notations $m \sim_{\mathcal{C}} n$ and $m \rightharpoonup n \in \overline{\mathcal{C}}$ are synonymous. The operator $\mathcal{C} \mapsto \overline{\mathcal{C}}$ is a closure operator on sets of constraints, i.e. it is extensive ($\mathcal{C} \subseteq \overline{\mathcal{C}}$), monotonic ($\mathcal{C} \subseteq \mathcal{D}$ implies $\overline{\mathcal{C}} \subseteq \overline{\mathcal{D}}$) and idempotent ($\overline{\overline{\mathcal{C}}} \subseteq \overline{\mathcal{C}}$). The identity $\mathcal{A}_{\mathcal{C}} = \mathcal{A}_{\overline{\mathcal{C}}}$ holds (see [13] Proposition 3.16) but the identity $\mathcal{L}_{\mathcal{C}} = \mathcal{L}_{\overline{\mathcal{C}}}$ *does not hold* in general, only the inclusion $\mathcal{L}_{\mathcal{C}} \subseteq \mathcal{L}_{\overline{\mathcal{C}}}$ holds.

[9] Not every PME is cancellative; e.g. $\sim = \{\epsilon \rightharpoonup \epsilon, \mathsf{x} \rightharpoonup \mathsf{x}, \mathsf{y} \rightharpoonup \mathsf{y}, \mathsf{k} \rightharpoonup \mathsf{k}, \mathsf{kx} \rightharpoonup \mathsf{kx}, \mathsf{ky} \rightharpoonup \mathsf{ky}, \mathsf{kx} \rightharpoonup \mathsf{ky}, \mathsf{ky} \rightharpoonup \mathsf{kx}\}$ is a non-cancellative PME over $L = \{\mathsf{x}, \mathsf{y}, \mathsf{k}\}$.

Proposition 4 (Compactness). *Let $\mathcal{C}$ be a set of constraints over the alphabet L and $m, n \in L^\star$ be s.t. $m \sim_{\mathcal{C}} n$ holds. There exists a finite subset $\mathcal{C}_f \subseteq \mathcal{C}$ such that $m \sim_{\mathcal{C}_f} n$.*

This *compactness property* (proved in [13] Proposition 3.17) is not related to the particular nature of rules defining PMEs but solely to the fact that the rules $\langle \epsilon, s, c, d, t \rangle$ only have a *finite number of premises*.

Definition 10 (PME Extension). *Let $\sim$ be a* PME *and $\mathcal{C}$ be a set of constraints, both over L. We denote by $\sim + \mathcal{C} = \overline{(\sim \cup \mathcal{C})}$ the* extension of $\sim$ *by the constraints of $\mathcal{C}$.*

The extension $\sim + \mathcal{C}$ is the least PME containing both $\sim$ and $\mathcal{C}$. Let $\sim$ be a PME and $\mathcal{C}_1, \mathcal{C}_2$ be two sets of constraints. The identities $(\sim + \mathcal{C}_1) + \mathcal{C}_2 = (\sim + \mathcal{C}_2) + \mathcal{C}_1 = \sim + (\mathcal{C}_1 \cup \mathcal{C}_2)$ hold. Moreover, for any $m, n \in L^\star$, the relation $m \sim n$ holds if and only if the identity $\sim + \{m \twoheadrightarrow n\} = \sim$ holds.

We single out PME extensions of the forms $\sim + \{\mathsf{ab} \twoheadrightarrow m\}$, $\sim + \{\mathsf{a}m \twoheadrightarrow \mathsf{b}\}$ or $\sim + \{\epsilon \twoheadrightarrow m\}$ where m is defined in $\sim$ and $\mathsf{a} \neq \mathsf{b}$ are two letters new to $\sim$. These extensions are generated by proof-search in the tableau method for BBI [12].

Definition 11 (Basic Extension). *Given a* PME *$\sim$ over the alphabet L, a constraint is* basic w.r.t. *$\sim$ when it is of one of the three forms $\mathsf{ab} \twoheadrightarrow m$, $\mathsf{a}m \twoheadrightarrow \mathsf{b}$ or $\epsilon \twoheadrightarrow m$ with $m \sim m$ and $\mathsf{a} \neq \mathsf{b} \in L \backslash \mathcal{A}_\sim$. When $x \twoheadrightarrow y$ is basic w.r.t. $\sim$, we say that $\sim + \{x \twoheadrightarrow y\}$ is a* basic extension of the PME *$\sim$.*

Let $k \in \mathbb{N} \cup \{\infty\}$ and $(x_i \twoheadrightarrow y_i)_{i<k}$ be a sequence of constraints. Let $\mathcal{C}_p = \{x_i \twoheadrightarrow y_i \mid i < p\}$ for $p < k$. We suppose that each extension $\sim_{\mathcal{C}_p} + \{x_p \twoheadrightarrow y_p\}$ is basic for any $p < k$. If $k < \infty$ (resp. $k = \infty$) then the sequence $(x_i \twoheadrightarrow y_i)_{i<k}$ is called *basic* (resp. *simple*). The empty sequence of constraints is basic.

Definition 12. *A* basic (resp. simple) PME *is of the form $\sim_{\mathcal{C}}$ where $\mathcal{C} = \{x_i \twoheadrightarrow y_i \mid i < k\}$ and $(x_i \twoheadrightarrow y_i)_{i<k}$ is a basic (resp. simple) sequence of constraints.*

Any basic PME is simple: indeed, by rule $\langle \epsilon \rangle$ we have $\sim + \{\epsilon \twoheadrightarrow \epsilon\} = \sim$ for any PME $\sim$. Thus, using case $\epsilon \twoheadrightarrow m$ of Definition 11 with $m = \epsilon$, we can complete any basic sequence into a simple sequence by looping on $\epsilon \twoheadrightarrow \epsilon$. The converse does not hold: simple PMEs with infinite alphabets are not basic.

Remark: we point out that in the set of constraints $\mathcal{C}$, the order of appearance of constraints does not impact the closure $\sim_{\mathcal{C}}$. However, in a basic (or simple) sequence of constraints, the order is important because the newness of letters depends on the previous constraints in the sequence. Moreover, to prove that a PME is not basic, it is not sufficient to show that the sequence that defines it is not basic: maybe there exists another defining sequence which is basic.

5 Equivalence Results for Some Separation Algebras

In this section, we show our main result: many of the different classes of separation algebra found in the literature (see discussion of Section 2) cannot be distinguished by any formula of Boolean BI. This is a stronger result than the

impossibility to axiomatize those classes in BBI [4]. Our result relies in an essential way on the (non-constructive) strong completeness theorem for partial monoidal BBI [12].[10] "Strong" means that $\mathsf{BBI}_{\mathrm{PD+SU}}$ is complete for the specific monoids that are generated by tableaux proof-search, i.e. simple PMEs.

Theorem 2 (Strong Completeness for Partial Monoidal BBI). *Let F be a* BBI*-formula that is invalid in some partial deterministic monoid with single unit, i.e. $F \notin \mathsf{BBI}_{\mathrm{PD+SU}}$. There exists a countable alphabet L, a simple* PME *$\sim$ over L, a valuation $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(L^\star/\!\sim)$ and a letter $\mathsf{a} \in L$ such that $\mathsf{a} \sim \mathsf{a}$ and $\mathfrak{M}_\sim, [\mathsf{a}] \nVdash_\delta F$.*

We will exploit the following properties of simple PMEs to derive our equivalence results for some separation algebras / abstract separation logics.

Theorem 3. *Simple* PME*s are closed under rule $\langle ca \rangle$. Simple* PME*s which are closed under rule $\langle iu \rangle$ are also closed under rule $\langle di \rangle$.*

Theorem 3 is the core result of the current paper. In Section 6, we introduce the tools used in its proof. In Section 7, we show that this proof cannot be done by direct induction on the sequence of constraints. In Section 8, we develop the argumentation using a detour via primary PMEs. The result is formally obtained as a conjunction of Corollaries 2 and 3.

Theorem 4. *The following notions of separation algebras found in the literature collapse to the same validity on* BBI *formulæ. Formally, we have the identities:*

(a) $\mathsf{BBI}_{\mathrm{PD}} = \mathsf{BBI}_{\mathrm{PD+SU}} = \mathsf{BBI}_{\mathrm{PD+CA}} = \mathsf{BBI}_{\mathrm{PD+SU+CA}}$;
(b) $\mathsf{BBI}_{\mathrm{PD+IU}} = \mathsf{BBI}_{\mathrm{PD+DI}} = \mathsf{BBI}_{\mathrm{PD+SU+CA+IU+DI}}$.

Proof. Let Q and K be the two following sub-classes $Q = \mathrm{PD} + \mathrm{SU} + \mathrm{CA}$ and $K = Q + \mathrm{IU} + \mathrm{DI}$ of ND-monoids. For (a), we prove the inclusions $\mathsf{BBI}_Q \subseteq \mathsf{BBI}_{\mathrm{PD+SU}} \subseteq \mathsf{BBI}_{\mathrm{PD}} \subseteq \mathsf{BBI}_{\mathrm{PD+CA}} \subseteq \mathsf{BBI}_Q$. We have $\mathsf{BBI}_{\mathrm{PD}} \subseteq \mathsf{BBI}_{\mathrm{PD+CA}} \subseteq \mathsf{BBI}_Q$ by sub-class inclusion in ND-monoids. By Theorem 1, we have $\mathsf{BBI}_{\mathrm{PD}} = \mathsf{BBI}_{\mathrm{PD+SU}}$. Hence, to obtain (a), it is sufficient to prove $\mathsf{BBI}_Q \subseteq \mathsf{BBI}_{\mathrm{PD+SU}}$. For (b), we show the inclusions $\mathsf{BBI}_K \subseteq \mathsf{BBI}_{\mathrm{PD+IU}} \subseteq \mathsf{BBI}_{\mathrm{PD+DI}} \subseteq \mathsf{BBI}_K$. Since we have $K \subseteq \mathrm{PD} + \mathrm{DI}$, the inclusion $\mathsf{BBI}_{\mathrm{PD+DI}} \subseteq \mathsf{BBI}_K$ is immediate. Then the inclusion $\mathsf{BBI}_{\mathrm{PD+IU}} \subseteq \mathsf{BBI}_{\mathrm{PD+DI}}$ is a direct consequence of Proposition 1. Hence, to obtain (b), it is sufficient to prove $\mathsf{BBI}_K \subseteq \mathsf{BBI}_{\mathrm{PD+IU}}$.

Let us prove the contrapositive of the inclusion $\mathsf{BBI}_Q \subseteq \mathsf{BBI}_{\mathrm{PD+SU}}$. Let us consider $F \notin \mathsf{BBI}_{\mathrm{PD+SU}}$ and let us show $F \notin \mathsf{BBI}_Q$. By Theorem 2, we obtain a simple PME $\sim$, a valuation $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(L^\star/\!\sim)$ and a letter $\mathsf{a} \in L$ such that $\mathsf{a} \sim \mathsf{a}$ and $\mathfrak{M}_\sim, [\mathsf{a}] \nVdash_\delta F$. By Theorem 3, the simple PME $\sim$ is closed under rule $\langle ca \rangle$ and thus, by Propositions 3, the partial quotient monoid $\mathfrak{M}_\sim$ belongs to the sub-class $\mathrm{PD} + \mathrm{SU} + \mathrm{CA}$. We deduce $F \notin \mathsf{BBI}_Q$.

Before we prove the inclusion $\mathsf{BBI}_K \subseteq \mathsf{BBI}_{\mathrm{PD+IU}}$, let us make a remark on the formula $\mathbb{U} = \neg(\neg\mathbb{I} * \neg\mathbb{I})$ and the scheme $(\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} (\cdot)$. Let $\mathfrak{M} = (M, \circ, \{e\})$ be a ND-monoid of sub-class SU and let $\delta : \mathsf{Var} \longrightarrow \mathcal{P}(M)$. Then we have $\mathfrak{M}, e \Vdash_\delta \mathbb{U}$

[10] The proof in Coq is available at `http://www.loria.fr/~larchey/BBI`.

if and only if $\mathfrak{M}$ is of sub-class IU. Let F be a BBI-formula. Then for any $x \in M$, we have $\mathfrak{M}, x \nVdash_\delta (\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} F$ if and only if $\mathfrak{M}$ is of sub-class IU and $\mathfrak{M}, x \nVdash_\delta F$.

Let us now prove the contrapositive of the inclusion $\mathsf{BBI}_K \subseteq \mathsf{BBI}_{\mathrm{PD+IU}}$. Let us consider a formula F such that $F \notin \mathsf{BBI}_{\mathrm{PD+IU}}$ and let us show $F \notin \mathsf{BBI}_K$. Let us first establish $(\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} F \notin \mathsf{BBI}_{\mathrm{PD+SU}}$. Since the sub-class PD + IU is closed under slicing, by Theorem 1 we have $F \notin \mathsf{BBI}_{\mathrm{PD+SU+IU}}$. Hence there exists a counter-model $\mathfrak{M}$ of F in sub-class PD + SU + IU. From the previous remark on $\mathbb{U}$, we deduce that $\mathfrak{M}$ is also a counter-model of $(\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} F$. As $\mathfrak{M}$ also belongs to sub-class PD + SU, we deduce $(\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} F \notin \mathsf{BBI}_{\mathrm{PD+SU}}$.

We apply Theorem 2 and we obtain a counter-model of $(\mathbb{I} \wedge \mathbb{U}) \mathrel{-\!\!*} F$ of the form $\mathfrak{M}_\sim$ where $\sim$ is a simple PME. Since $\mathfrak{M}_\sim$ is of subclass SU, we deduce that $\mathfrak{M}_\sim$ is of subclass IU and $\mathfrak{M}_\sim$ is a counter-model of F (see previous remark on $\mathbb{U}$). Hence $\mathfrak{M}_\sim$ is of sub-class PD + SU + IU. Thus by Proposition 3, $\sim$ is closed under rule $\langle iu \rangle$. Hence by Theorem 3, the simple PME $\sim$ is closed under rules $\langle ca \rangle$ and $\langle di \rangle$. By Proposition 3, $\mathfrak{M}_\sim$ is a counter-model of F of sub-class PD + SU + CA + IU + DI and we conclude $F \notin \mathsf{BBI}_K$.

Remark: unlike IU, DI is not axiomatizable in BBI [4] thus we cannot have $\mathsf{BBI}_{\mathrm{DI}} = \mathsf{BBI}_{\mathrm{IU}}$. Hence the strict inclusion $\mathsf{BBI}_{\mathrm{IU}} \subsetneq \mathsf{BBI}_{\mathrm{DI}}$ by Proposition 1. Let us now discuss and develop the proof of Theorem 3, our core result.

6 Invertibility, Group-PMEs and Squares

In this section, we study the properties of the extension $\sim + \{\epsilon \nsim m\}$ and how they impact invertible letters/words. We introduce the notion of group-PME.

Definition 13. *A* group-PME *over L is a* PME $\sim$ *such that $\mathcal{A}_\sim = \mathcal{I}_\sim$ where $\mathcal{I}_\sim = \{\mathsf{c} \in L \mid \epsilon \sim \mathsf{c}\beta$ holds for some $\beta \in L^\star\}$ is the set of* invertible letters *of $\sim$.*

The operator $\sim \mapsto \mathcal{I}_\sim$ is monotonic. By rule $\langle p_r \rangle$, the inclusion $\mathcal{I}_\sim \subseteq \mathcal{A}_\sim$ holds for any PME. We may write $\mathcal{I}_\mathcal{C}$ for $\mathcal{I}_{\sim_\mathcal{C}}$; this should not lead to any ambiguity. We introduce a set of derived rules related to invertible words (in $\mathcal{I}^\star_\sim$) and we analyze the relations between $\sim$ and invertible words. Appart from the letter α which serves as a parameter for (primary) extensions, we ease the reading by denoting invertible words with greek letters β, γ, ... in place of x, y, ...

Definition 14 (Squares and Invertible Squares). *We say that a word $\alpha \in L^\star$ is* square-free *if $\forall \mathsf{c} \in L, \mathsf{cc} \not\sim \alpha$. We say that the* PME $\sim$ *be over L* has invertible squares *if $\forall \mathsf{c} \in L, \mathsf{cc} \sim \mathsf{cc} \Rightarrow \mathsf{c} \in \mathcal{I}_\sim$ (i.e. any squarable letter is invertible).*

Proposition 5. *Let $\sim$ be a* PME *over L. If $\sim$ has invertible squares then for any word $k \in L^\star$, if $kk \sim kk$ holds then $k \in \mathcal{I}^\star_\sim$ holds.*

Proposition 6. PME*s are closed under rules $\langle \epsilon_c, i_\uparrow, i_c, i_s, i_\leftarrow, i_\rightarrow \rangle$:*

$$\frac{\epsilon \nsim \gamma \quad \epsilon \nsim \beta}{\epsilon \nsim \gamma\beta}\,\langle \epsilon_c \rangle \qquad \frac{x \nsim y \quad \epsilon \nsim \gamma\beta}{\gamma x \nsim \gamma y}\,\langle i_c \rangle \qquad \frac{x \nsim \beta y \quad \epsilon \nsim \gamma\beta}{\gamma x \nsim y}\,\langle i_\leftarrow \rangle$$

$$\frac{\epsilon \nsim \gamma\beta \quad \epsilon \nsim \gamma\beta'}{\beta \nsim \beta'}\,\langle i_\uparrow \rangle \qquad \frac{\gamma x \nsim \gamma y \quad \epsilon \nsim \gamma\beta}{x \nsim y}\,\langle i_s \rangle \qquad \frac{\gamma x \nsim y \quad \epsilon \nsim \gamma\beta}{x \nsim \beta y}\,\langle i_\rightarrow \rangle$$

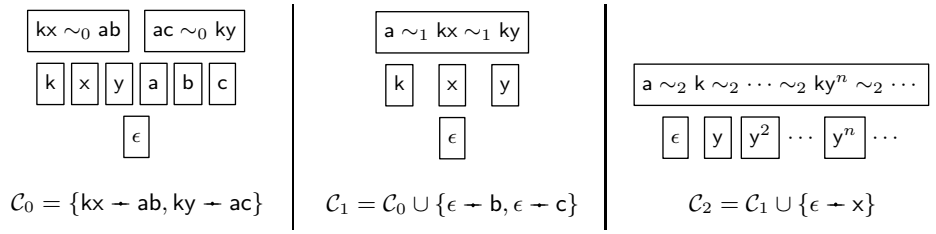


Fig. 2. The partial equivalence classes of $\sim_0 = \overline{\mathcal{C}_0}$, $\sim_1 = \overline{\mathcal{C}_1}$ and $\sim_2 = \overline{\mathcal{C}_2}$

Proposition 7. *Let $\sim$ be a* PME *over L and $x, y \in L^\star$ and $\gamma \in \mathcal{I}^\star_\sim$. We have: (a) $x \in \mathcal{I}^\star_\sim$ iff $\exists \beta\ \epsilon \sim x\beta$; (b) $x \sim y$ iff $\gamma x \sim \gamma y$; (c) the inclusion $\mathcal{I}^\star_\sim \subseteq \mathcal{L}_\sim$ holds; (d) if $x \sim y$ then $x \in \mathcal{I}^\star_\sim \Leftrightarrow y \in \mathcal{I}^\star_\sim$.*

In any group-PME $\sim$, every defined letter is invertible and from Proposition 7 (c), we obtain the identity $\mathcal{L}_\sim = \mathcal{I}^\star_\sim$.[11] Proposition 8 makes explicit a sufficient condition under which extensions do not change invertible letters: no new invertible letter appears in $\sim + \{x \rightharpoonup y\}$ unless either $x \in \mathcal{I}^\star_\sim$ or $y \in \mathcal{I}^\star_\sim$.

Proposition 8. *Let $\sim$ be a* PME *and $\mathcal{C}$ be a set of constraints such that for any $x \rightharpoonup y \in \mathcal{C}$ the identity $\{x, y\} \cap \mathcal{I}^\star_\sim = \emptyset$ holds. Then the identity $\mathcal{I}_{\sim+\mathcal{C}} = \mathcal{I}_\sim$ holds.*

7 No Direct Inductive Proof of Cancellativity for Basic PMEs

We argue that it is not possible to prove cancellativity of basic PMEs by a direct induction on the length of the sequence defining them. This justifies the involved development that lies ahead. We present an example where the extensions $\sim + \{\epsilon \rightharpoonup m\}$ break cancellativity and introduce non-invertible squares.[12]

Let $\mathsf{k}, \mathsf{x}, \mathsf{y}, \mathsf{a}, \mathsf{b}, \mathsf{c} \in L$ be six different letters. Let us consider the following PME $\sim_0 = \sim_{\mathcal{C}_0}$ where $\mathcal{C}_0 = \{\mathsf{kx} \rightharpoonup \mathsf{ab}, \mathsf{ky} \rightharpoonup \mathsf{ac}\}$. In Figure 2, we represent the corresponding set of partial equivalence classes of $\sim_0$. It is left to the reader to check that these are indeed the partial equivalence classes of the closure of $\mathcal{C}_0$: we have $L^\star/\sim_0 = \{[\epsilon], [\mathsf{k}], [\mathsf{x}], [\mathsf{y}], [\mathsf{a}], [\mathsf{b}], [\mathsf{c}], [\mathsf{kx}], [\mathsf{ky}]\}$ with $[\alpha] = \{\alpha\}$ for $\alpha \in \{\epsilon, \mathsf{k}, \mathsf{x}, \mathsf{y}, \mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $[\mathsf{kx}] = \{\mathsf{kx}, \mathsf{ab}\}$ and $[\mathsf{ky}] = \{\mathsf{ky}, \mathsf{ac}\}$. We check that $\sim_0$ is cancellative and has invertible squares (it contains no square except ϵ).

Now we consider the extension $\mathcal{C}_1 = \mathcal{C}_0 \cup \{\epsilon \rightharpoonup \mathsf{b}, \epsilon \rightharpoonup \mathsf{c}\}$ and $\sim_1 = \sim_0 + \{\epsilon \rightharpoonup \mathsf{b}, \epsilon \rightharpoonup \mathsf{c}\}$. Let us denote $E = \mathsf{b}^\star\mathsf{c}^\star = \{\mathsf{b}^i\mathsf{c}^j \mid i, j \in \mathbb{N}\}$. Then $L^\star/\sim_1 = \{[\epsilon], [\mathsf{k}], [\mathsf{x}], [\mathsf{y}], [\mathsf{a}]\}$ where $[\alpha] = \alpha E$ for $\alpha \in \{\epsilon, \mathsf{k}, \mathsf{x}, \mathsf{y}\}$ and $[\mathsf{a}] = (\mathsf{a} \mid \mathsf{kx} \mid \mathsf{ky})E$. The PME $\sim_1$ is not cancellative anymore. Indeed, $\mathsf{kx} \sim_1 \mathsf{ky}$ but $\mathsf{x} \not\sim_1 \mathsf{y}$. Hence we have an example

[11] In that case, $\sim$ is a congruence over $\mathcal{I}^\star_\sim$ and the quotient $\mathcal{I}^\star_\sim/\sim$ is an Abelian group.
[12] i.e. some kk with $kk \sim kk$ and $k \notin \mathcal{I}^\star_\sim$; see Definition 14.

that shows that the extension $\sim + \{\epsilon \rightarrowtail m\}$ does not preserve cancellativity. But still $\sim_1$ has invertible squares; check that $\mathcal{I}_{\sim_1} = \{\mathsf{b}, \mathsf{c}\}$.

Finally we consider the extension $\mathcal{C}_2 = \mathcal{C}_1 \cup \{\epsilon \rightarrowtail \mathsf{x}\}$ and $\sim_2 = \sim_1 + \{\epsilon \rightarrowtail \mathsf{x}\}$. Let us denote $E = \mathsf{b}^\star\mathsf{c}^\star\mathsf{x}^\star$. Then $L^\star/\sim_2 = \{[\mathsf{y}^n] \mid n \geqslant 0\} \cup \{[\mathsf{a}]\}$ with $[\mathsf{y}^n] = \mathsf{y}^n E$ and $[\mathsf{a}] = (\mathsf{a} \mid \mathsf{k})\mathsf{y}^\star E$. Like $\sim_1$, the PME $\sim_2$ is not cancellative. Moreover it has squares like y^2 where y is not an invertible letter; check $\mathcal{I}_{\sim_2} = \{\mathsf{b}, \mathsf{c}, \mathsf{x}\}$. Hence $\sim_2$ contains non-invertible squares.

We see that the extension $\sim + \{\epsilon \rightarrowtail m\}$ preserves neither cancellativity nor the invertibility of squares. Therefore it is not possible to show that basic PMEs have these properties by direct induction on the basic sequence.

8 Basic PMEs are Primary Extensions of Group-PMEs

We define the notion of primary extension and use the equations in Lemma 3 to show that cancellativity and invertible squares are preserved by primary extensions. We then prove that basic PMEs are primary extensions of group-PMEs.

Definition 15 (Primary PME). *Let $\sim$ be a* PME *over L and $\alpha, m \in L^\star$ be two words such that $m \sim m$, $\alpha \neq \epsilon$, $\mathcal{A}_\sim \cap \mathcal{A}_\alpha = \emptyset$ and α is square-free. A* type-1 extension of $\sim$ *is of the form $\sim + \{\alpha \rightarrowtail m\}$; A* type-2 extension of $\sim$ *is of the form $\sim + \{\alpha m \rightarrowtail \mathsf{b}\}$ with $\mathsf{b} \in L \setminus (\mathcal{A}_\sim \cup \mathcal{A}_\alpha)$. A* primary extension of $\sim$ *is a type-1 or a type-2 extension of $\sim$. The class of* primary PMEs *is the least class containing group-*PMEs *and closed under primary extensions.*

We show that the properties of "cancellativity" and "invertible squares" hold for group-PMEs and are preserved by primary extensions.

Lemma 2. *Every group-*PME *is cancellative and has invertible squares.*

Lemma 3. *Let $\sim$ be a* PME *over L and $m, \alpha \in L^\star$ be such that $m \sim m$, $\alpha \neq \epsilon$ and $\mathcal{A}_\alpha \cap \mathcal{A}_\sim = \emptyset$. Then the two following identities hold:*

$$\sim + \{\alpha \rightarrowtail m\} = \{\delta\alpha^u x \rightarrowtail \delta\alpha^v y \mid \exists i,\ m^u x \sim m^v y, m^{i+u} x \sim m^{i+v} y \text{ and } \delta \prec \alpha^i\}$$
$$\sim + \{\alpha m \rightarrowtail \alpha m\} = \sim \cup \{\delta x \rightarrowtail \delta y \mid x \sim y, \epsilon \neq \delta \prec \alpha \text{ and } \exists q\ xq \sim m\}$$

Moreover, if $\sim$ is cancellative then both $\sim + \{\alpha \rightarrowtail m\}$ and $\sim + \{\alpha m \rightarrowtail \alpha m\}$ are cancellative; and if $\sim$ has invertible squares and α is square-free then both $\sim + \{\alpha \rightarrowtail m\}$ and $\sim + \{\alpha m \rightarrowtail \alpha m\}$ have invertible squares.

Corollary 1. *Primary* PMEs *are cancellative and have invertible squares.*

The proof of Lemma 3 is long/technical but not too difficult (once you have the equations). We now prove our core result: basic PMEs are primary PMEs; in particular, they are cancellative and have invertible squares.

Theorem 5. *Basic* PMEs *are primary* PMEs.

Proof. Let us consider a basic PME $\sim$. By Definition 12, there exists a basic sequence of constraints $(x_i \rightharpoonup y_i)_{i<k}$ such that $\sim \; = \; \sim_{\mathcal{H}}$, with $k < \infty$ and $\mathcal{H} = \{x_0 \rightharpoonup y_0, \ldots, x_{k-1} \rightharpoonup y_{k-1}\}$. For any $q \leqslant k$, we denote $\mathcal{H}_q = \{x_i \rightharpoonup y_i \mid i < q\}$. The extension $\sim_{\mathcal{H}_q} + \{x_q \rightharpoonup y_q\}$ is basic for any $q < k$. We recall the notation $\mathcal{I}_\sim = \mathcal{I}_{\mathcal{H}}$ for the set of invertible letters of $\sim \; = \; \sim_{\mathcal{H}}$.

From $x_i \rightharpoonup y_i \in \mathcal{H}$, we deduce $x_i \sim y_i$ and by Proposition 7 (d), we have $x_i \in \mathcal{I}_\sim^\star$ iff $y_i \in \mathcal{I}_\sim^\star$ for any $i < k$. Hence we obtain a partition $[0, k[\; = C \uplus D$ with $C = \{i < k \mid \{x_i, y_i\} \subseteq \mathcal{I}_\sim^\star\}$ and $D = \{i < k \mid \{x_i, y_i\} \cap \mathcal{I}_\sim^\star = \emptyset\}$. Let us denote $\mathcal{C} = \{x_i \rightharpoonup y_i \mid i \in C\}$ and $\mathcal{D} = \{x_i \rightharpoonup y_i \mid i \in D\}$.

Let us enumerate $D = \{\sigma_0 < \cdots < \sigma_{d-1}\}$ in strictly increasing order with $d = \mathrm{card}(D) \leqslant k$ and $\sigma : [0, d[\longrightarrow [0, k[$. For $q \leqslant d$, let us denote $D_q = \{\sigma_i \mid i < q\}$. We show the inclusion $[0, \sigma_q[\subseteq C \cup D_q$: indeed, let us consider $j < \sigma_q$ and let us prove $j \in C \cup D_q$. From $\sigma_q < k$, we deduce $j \in [0, k[\; = C \uplus D$. In case $j \in C$, we have finished. In case $j \in D = \{\sigma_0 < \cdots < \sigma_{d-1}\}$, then $j = \sigma_r$ for some $r < d$. If $q \leqslant r$ then $\sigma_q \leqslant \sigma_r = j$ which contradicts $j < \sigma_q$. Hence we must have $r < q$ and we conclude $j = \sigma_r \in D_q$. Let us denote $\mathcal{D}_q = \{x_{\sigma_i} \rightharpoonup y_{\sigma_i} \mid i < q\}$ for $q \leqslant d$. From $D_q \subseteq [0, \sigma_q[$ we derive $\mathcal{D}_q \subseteq \mathcal{H}_{\sigma_q}$.

Let us prove the identities $\mathcal{A}_\mathcal{C} = \mathcal{I}_\mathcal{C} = \mathcal{I}_\sim$. Since $\mathcal{H} = \mathcal{C} \cup \mathcal{D}$, we get $\sim_{\mathcal{H}} \; = \; \sim_{\mathcal{C}+\mathcal{D}}$. Moreover, every constraint of $\mathcal{D}$ is of the form $x \rightharpoonup y$ with $\{x, y\} \cap \mathcal{I}_\sim^\star = \emptyset$. As $\mathcal{I}_\mathcal{C} \subseteq \mathcal{I}_\mathcal{H} = \mathcal{I}_\sim$ we deduce $\{x, y\} \cap \mathcal{I}_\mathcal{C}^\star = \emptyset$ for every constraint $x \rightharpoonup y \in \mathcal{D}$. Thus, by Proposition 8, we have $\mathcal{I}_{\sim_{\mathcal{C}+\mathcal{D}}} = \mathcal{I}_{\sim_\mathcal{C}}$ and thus $\mathcal{I}_\mathcal{C} = \mathcal{I}_{\sim_\mathcal{C}} = \mathcal{I}_{\sim_{\mathcal{C}+\mathcal{D}}} = \mathcal{I}_{\sim_\mathcal{H}} = \mathcal{I}_\sim$. Also, for any $x \rightharpoonup y \in \mathcal{C}$ we have $\{x, y\} \subseteq \mathcal{I}_\sim^\star$ and thus $\mathcal{A}_\mathcal{C} \subseteq \mathcal{I}_\sim$. We conclude $\mathcal{A}_\mathcal{C} = \mathcal{I}_\mathcal{C} = \mathcal{I}_\sim$. In particular, $\sim_\mathcal{C}$ is a group-PME.

Let us define $\mathcal{E}_q = \mathcal{C} \cup \mathcal{D}_q$ for $q \leqslant d$. As $\mathcal{C} \subseteq \mathcal{E}_q \subseteq \mathcal{E}_d = \mathcal{C} \cup \mathcal{D}_d = \mathcal{C} \cup \mathcal{D} = \mathcal{H}$, we deduce $\mathcal{I}_\sim = \mathcal{I}_\mathcal{C} \subseteq \mathcal{I}_{\mathcal{E}_q} \subseteq \mathcal{I}_\mathcal{H} = \mathcal{I}_\sim$ and thus $\mathcal{I}_{\mathcal{E}_q} = \mathcal{I}_\sim$ for any $q \leqslant d$. Let us establish the inclusions $\mathcal{H}_{\sigma_q} \subseteq \mathcal{E}_q$ and $\mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_\sim$. The first inclusion follows from $[0, \sigma_q[\subseteq C \cup D_q$ and the definitions of $\mathcal{H}_{\sigma_q}$ and $\mathcal{E}_q$. For the second inclusion, starting with $\mathcal{D}_q \subseteq \mathcal{H}_{\sigma_q}$ we derive $\mathcal{E}_q = \mathcal{C} \cup \mathcal{D}_q \subseteq \mathcal{C} \cup \mathcal{H}_{\sigma_q}$ and thus $\mathcal{A}_{\mathcal{E}_q} \subseteq \mathcal{A}_\mathcal{C} \cup \mathcal{A}_{\mathcal{H}_{\sigma_q}} = \mathcal{I}_\sim \cup \mathcal{A}_{\mathcal{H}_{\sigma_q}}$. Hence the inclusion $\mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_\sim$.

Let us show by induction on $q \leqslant d$ that $\sim_{\mathcal{E}_q}$ is a primary PME. First the ground case. We have $\mathcal{D}_0 = \emptyset$ and thus the identity $\sim_{\mathcal{E}_0} \; = \; \sim_\mathcal{C}$ holds. As a consequence, $\sim_{\mathcal{E}_0}$ is a group-PME and thus is a primary PME. Then the induction step. We assume that $\sim_{\mathcal{E}_q}$ is a primary PME and we show that $\sim_{\mathcal{E}_{q+1}} \; = \; \sim_{\mathcal{E}_q} + \{x_{\sigma_q} \rightharpoonup y_{\sigma_q}\}$ is also a primary PME. For this aim, we show that $\sim_{\mathcal{E}_q} + \{x_{\sigma_q} \rightharpoonup y_{\sigma_q}\}$ is identical to a primary extension of $\sim_{\mathcal{E}_q}$. We remind that the constraint $x_{\sigma_q} \rightharpoonup y_{\sigma_q}$ is basic w.r.t. $\sim_{\mathcal{H}_{\sigma_q}}$. We proceed by case analysis on that fact (see Definition 11):

- if $x_{\sigma_q} \rightharpoonup y_{\sigma_q} = \mathsf{ab} \rightharpoonup m$ with $m \sim_{\mathcal{H}_{\sigma_q}} m$ and $\mathsf{a} \neq \mathsf{b} \in L \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}}$. From $\mathcal{H}_{\sigma_q} \subseteq \mathcal{E}_q$ we deduce $m \sim_{\mathcal{E}_q} m$. We establish the relation $\{\mathsf{a}, \mathsf{b}\} \not\subseteq \mathcal{A}_{\mathcal{E}_q}$: if $\{\mathsf{a}, \mathsf{b}\} \subseteq \mathcal{A}_{\mathcal{E}_q}$ holds then we have $\{\mathsf{a}, \mathsf{b}\} \subseteq \mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_\sim$ and as a consequence $\mathsf{ab} \in \mathcal{I}_\sim^\star$. But from $\sigma_q \in D$, we get $\mathsf{ab} = x_{\sigma_q} \notin \mathcal{I}_\sim^\star$ which leads to a contradiction.

  In case $\{\mathsf{a}, \mathsf{b}\} \cap \mathcal{A}_{\mathcal{E}_q} = \emptyset$ then $\mathcal{A}_\mathsf{ab} \cap \mathcal{A}_{\mathcal{E}_q} = \emptyset$, $\mathsf{ab} \neq \epsilon$ is square-free and $m \sim_{\mathcal{E}_q} m$. Hence, $\sim_{\mathcal{E}_q} + \{\mathsf{ab} \rightharpoonup m\}$ is a type-1 primary extension of $\sim_{\mathcal{E}_q}$.

  In case $\mathsf{a} \in \mathcal{A}_{\mathcal{E}_q}$ and $\mathsf{b} \notin \mathcal{A}_{\mathcal{E}_q}$ then $\mathsf{a} \in \mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_\sim = \mathcal{I}_{\mathcal{E}_q}$ and hence we have $\epsilon \sim_{\mathcal{E}_q} \mathsf{a}\beta$ for some β. The identity $\sim_{\mathcal{E}_q} + \{\mathsf{ab} \rightharpoonup m\} = \sim_{\mathcal{E}_q} +$

$\{\mathsf{b} \twoheadrightarrow m\beta\}$ holds by direct application of rules $\langle i_{\leftarrow}\rangle$ and $\langle i_{\rightarrow}\rangle$. We verify that $\sim_{\mathcal{E}_q} + \{\mathsf{b} \twoheadrightarrow m\beta\}$ is a type-1 primary extension of $\sim_{\mathcal{E}_q}$: $\mathsf{b} \neq \epsilon$ is square-free, $\mathcal{A}_{\mathsf{b}} \cap \mathcal{A}_{\mathcal{E}_q} = \emptyset$, $m\beta \sim_{\mathcal{E}_q} m\beta$ (because $m \sim_{\mathcal{E}_q} m$, $\epsilon \sim_{\mathcal{E}_q} \mathsf{a}\beta$ and rule $\langle i_c\rangle$). Hence $\sim_{\mathcal{E}_q} + \{\mathsf{ab} \twoheadrightarrow m\}$ is identical to a type-1 primary extension of $\sim_{\mathcal{E}_q}$.

The case $\mathsf{b} \in \mathcal{A}_{\mathcal{E}_q}$ and $\mathsf{a} \notin \mathcal{A}_{\mathcal{E}_q}$ can be treated in a symmetric way. In any of these three cases, we have proved that the PME $\sim_{\mathcal{E}_q} + \{\mathsf{ab} \twoheadrightarrow m\}$ can be expressed as a type-1 primary extension of $\sim_{\mathcal{E}_q}$;

– if $x_{\sigma_q} \twoheadrightarrow y_{\sigma_q} = \mathsf{a}m \twoheadrightarrow \mathsf{b}$ with $m \sim_{\mathcal{H}_{\sigma_q}} m$ and $\mathsf{a} \neq \mathsf{b} \in L \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}}$. From $\sigma_q \in D$, we have $\mathsf{b} = y_{\sigma_q} \notin \mathcal{I}^{\star}_{\sim}$ and thus $\mathsf{b} \notin \mathcal{I}_{\sim}$. From the inclusion $\mathcal{H}_{\sigma_q} \subseteq \mathcal{E}_q$, we deduce $m \sim_{\mathcal{E}_q} m$. We further have $\mathsf{b} \notin \mathcal{A}_{\mathcal{E}_q}$ (otherwise we would have $\mathsf{b} \in \mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_{\sim}$ contradicting $\mathsf{b} \notin \mathcal{I}_{\sim}$). We consider the two cases $\mathsf{a} \notin \mathcal{A}_{\mathcal{E}_q}$ and $\mathsf{a} \in \mathcal{A}_{\mathcal{E}_q}$.

In case $\mathsf{a} \notin \mathcal{A}_{\mathcal{E}_q}$ then we check that $\sim_{\mathcal{E}_q} + \{\mathsf{a}m \twoheadrightarrow \mathsf{b}\}$ is a type-2 primary extension: $\mathsf{a} \neq \epsilon$ is square-free, $\mathcal{A}_{\mathsf{a}} \cap \mathcal{A}_{\mathcal{E}_q} = \emptyset$, $m \sim_{\mathcal{E}_q} m$ and $\mathsf{b} \notin \mathcal{A}_{\mathcal{E}_q} \cup \mathcal{A}_{\mathsf{a}}$.

In case $\mathsf{a} \in \mathcal{A}_{\mathcal{E}_q}$ then $\mathsf{a} \in \mathcal{A}_{\mathcal{E}_q} \setminus \mathcal{A}_{\mathcal{H}_{\sigma_q}} \subseteq \mathcal{I}_{\sim} = \mathcal{I}_{\mathcal{E}_q}$. Hence there exists β such that $\epsilon \sim_{\mathcal{E}_q} \mathsf{a}\beta$. The identity $\sim_{\mathcal{E}_q} + \{\mathsf{a}m \twoheadrightarrow \mathsf{b}\} = \sim_{\mathcal{E}_q} + \{\mathsf{b} \twoheadrightarrow \mathsf{a}m\}$ holds by rule $\langle s\rangle$. Let us check that $\sim_{\mathcal{E}_q} + \{\mathsf{b} \twoheadrightarrow \mathsf{a}m\}$ is a type-1 primary extension of $\sim_{\mathcal{E}_q}$: $\mathsf{b} \neq \epsilon$ is square-free and $\mathcal{A}_{\mathsf{b}} \cap \mathcal{A}_{\mathcal{E}_q} = \emptyset$ holds. $\mathsf{a}m \sim_{\mathcal{E}_q} \mathsf{a}m$ is the last remaining condition, obtained from $m \sim_{\mathcal{E}_q} m$ and $\epsilon \sim_{\mathcal{E}_q} \mathsf{a}\beta$ using rule $\langle i_c\rangle$.

In any of these two cases, we have proved that the PME $\sim_{\mathcal{E}_q} + \{\mathsf{a}m \twoheadrightarrow \mathsf{b}\}$ can be expressed as a type-1 or as type-2 primary extension of $\sim_{\mathcal{E}_q}$;

– if $x_{\sigma_q} \twoheadrightarrow y_{\sigma_q} = \epsilon \twoheadrightarrow m$ with $m \sim_{\mathcal{H}_{\sigma_q}} m$. Then we have $x_{\sigma_q} = \epsilon \in \mathcal{I}^{\star}_{\sim}$ which directly contradicts $\{x_{\sigma_q}, y_{\sigma_q}\} \cap \mathcal{I}^{\star}_{\sim} = \emptyset$. Hence this case is not possible.

Hence, by induction on $q \leqslant d$, the PME $\sim_{\mathcal{E}_q}$ is primary. In particular $\sim_{\mathcal{E}_d} = \sim_{\mathcal{H}} = \sim$ is a primary PME.

Corollary 2. *Basic and simple* PME*s are cancellative and have invertible squares.*

Corollary 3. *Simple* PME*s closed under rule* $\langle iu\rangle$ *are also closed under rule* $\langle di\rangle$.

9 Conclusion

In this paper, we prove that validity in Boolean BI does not distinguish between some of the different notions of separation algebras commonly found in the literature. This result is obtained by an in-depth examination of the syntactic properties of basic/simple PMEs which are the counter-models that are generated by tableaux proof-search. We show that these models are cancellative and that the only squares they allow are composed of invertible letters using a detour via the notion of primary PME. From the cancellativity of simple PMEs and the strong completeness theorem, we derive equivalence results for cancellative partial monoids. We relate indivisibility of units to the disjointness property.

We propose some perspectives. First, we could investigate more properties of basic/simple PMEs to enrich the graph of known relations between the family $(\mathsf{BBI}_X)_X$. In particular, we expect a full characterization of basic PMEs that could lead to finer properties of simple PMEs. Another track of research

would be to find a constructive proof of the results of this paper. There is little hope to succeed by using the strong completeness which is inescapably non-constructive; but we could for instance approach the problem by eliminating the cancellativity rule in the proofs of the sequent calculus [9]. Another way to tackle the problem would be to design bisimulations or at least Kripke semantics preserving relations between cancellative and non-cancellative models.

References

1. Brotherston, J.: Bunched Logics Displayed. Studia Logica 100(6), 1223–1254 (2012)
2. Brotherston, J., Kanovich, M.: Undecidability of Propositional Separation Logic and Its Neighbours. In: LICS, pp. 130–139. IEEE Computer Society (2010)
3. Brotherston, J., Kanovich, M.: Undecidability of Propositional Separation Logic and its Neighbours. Journal of the ACM 61(2) (2014)
4. Brotherston, J., Villard, J.: Parametric Completeness for Separation Theories. In: POPL, pp. 453–464. ACM (2014)
5. Calcagno, C., O'Hearn, P.W., Yang, H.: Local Action and Abstract Separation Logic. In: LICS, pp. 366–378. IEEE Computer Society (2007)
6. Dinsdale-Young, T., Birkedal, L., Gardner, P., Parkinson, M.J., Yang, H.: Views: compositional reasoning for concurrent programs. In: POPL, pp. 287–300. ACM (2013)
7. Dockins, R., Hobor, A., Appel, A.W.: A Fresh Look at Separation Algebras and Share Accounting. In: Hu, Z. (ed.) APLAS 2009. LNCS, vol. 5904, pp. 161–177. Springer, Heidelberg (2009)
8. Galmiche, D., Larchey-Wendling, D.: Expressivity properties of Boolean BI through Relational Models. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 357–368. Springer, Heidelberg (2006)
9. Hóu, Z., Clouston, R., Goré, R., Tiu, A.: Proof search for propositional abstract separation logics via labelled sequents. In: POPL, pp. 465–476. ACM (2014)
10. Hóu, Z., Tiu, A., Goré, R.: A Labelled Sequent Calculus for BBI: Proof Theory and Proof Search. In: Galmiche, D., Larchey-Wendling, D. (eds.) TABLEAUX 2013. LNCS, vol. 8123, pp. 172–187. Springer, Heidelberg (2013)
11. Ishtiaq, S.S., O'Hearn, P.W.: BI as an Assertion Language for Mutable Data Structures. In: POPL, London, UK, pp. 14–26 (2001)
12. Larchey-Wendling, D.: The formal strong completeness of partial monoidal Boolean BI. J. Logic. Comput. (2014), http://dx.doi.org/10.1093/logcom/exu031, doi:10.1093/logcom/exu031
13. Larchey-Wendling, D., Galmiche, D.: Exploring the relation between Intuitionistic BI and Boolean BI: an unexpected embedding. Mathematical Structures in Computer Science 19(3), 435–500 (2009)
14. Larchey-Wendling, D., Galmiche, D.: The Undecidability of Boolean BI through Phase Semantics. In: LICS, pp. 140–149. IEEE Computer Society (2010)
15. Larchey-Wendling, D., Galmiche, D.: Nondeterministic Phase Semantics and the Undecidability of Boolean BI. ACM ToCL 14(1), 6 (2013)
16. Park, J., Seo, J., Park, S.: A theorem prover for Boolean BI. In: POPL, pp. 219–232. ACM (2013)
17. Pym, D.: The Semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series, vol. 26. Kluwer Academic Publishers (2002)
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74. IEEE Computer Society (2002)

The Inhabitation Problem for Non-idempotent Intersection Types

Antonio Bucciarelli[1], Delia Kesner[1], and Simona Ronchi Della Rocca[2]

[1] Univ Paris Diderot, Sorbonne Paris Cit, PPS, UMR 7126, CNRS, Paris, France
[2] Dipartimento di Informatica, Università di Torino, Italy

Abstract. The inhabitation problem for intersection types is known to be undecidable. We study the problem in the case of non-idempotent intersection, and we prove decidability through a sound and complete algorithm. We then consider the inhabitation problem for an extended system typing the λ-calculus with pairs, and we prove the decidability in this case too. The extended system is interesting in its own, since it allows to characterize solvable terms in the λ-calculus with pairs.

1 Introduction

Intersection types have been presented in the literature in many variants. Historically, one of the first versions is the one characterizing solvable terms, that we call system $\mathcal{C}$, shown in Fig. 1 [6,15]. Intersection enjoys associativity, commutativity, and in particular idempotency ($\mathtt{A} \wedge \mathtt{A} = \mathtt{A}$). Given a type $\mathtt{A}$ and a typing environment Γ, the problem of deciding whether there exists a term $\mathtt{t}$ such that $\Gamma \vdash \mathtt{t} : \mathtt{A}$ is provable, is known in the literature both as *emptiness problem* and as *inhabitation problem*. The inhabitation problem for system $\mathcal{C}$ has been proved to be undecidable by Urzyczyn [21]. Van Bakel [23] simplified the system, using strict types, where intersection is not allowed on the right side of the arrow; his system $\mathcal{S}$ is presented on the left part of Fig. 2, where intersection is naturally represented through set formation, and the universal type ω by the empty set. The right part of the figure presents the relevant version of $\mathcal{S}$, $\mathcal{S}_r$, a system being relevant if and only if, in its provable judgments, typing environments only contain the consumed premises. The systems $\mathcal{C}$, $\mathcal{S}$ and $\mathcal{S}_r$ are equivalent with respect to the typability power (neglecting the universal type ω). Urzyczyn's proof of undecidability of the inhabitation problem for system $\mathcal{C}$ can be easily adapted to system $\mathcal{S}$. Moreover, the inhabitation problem for $\mathcal{C}$ seems to reduce to that for $\mathcal{S}_r$, proving that the latter is undecidable too [22], so that relevance has nothing to do with the hardness of the inhabitation problem.

In this paper we consider the type assignment system $\mathcal{M}$, which is a variant of $\mathcal{S}_r$, where idempotency of intersection has been removed, and we prove that its inhabitation is decidable, by exploiting the fact that in this case types keep track faithfully of the different uses of variables in terms. System $\mathcal{M}$ characterizes terms having head normal form, so we design a sound and complete algorithm, that, given a typing environment Γ and a type σ, builds a set of approximate

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 341–354, 2014.

$$\frac{\mathtt{x} : \mathtt{A} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{A}}(\mathtt{var}) \qquad \frac{}{\Gamma \vdash \mathtt{t} : \omega}(\omega) \qquad \frac{\Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{t} : \mathtt{B}}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{t} : \mathtt{A} \to \mathtt{B}}(\to \mathtt{I})$$

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{A} \to \mathtt{B} \quad \Gamma \vdash \mathtt{u} : \mathtt{A}}{\Gamma \vdash \mathtt{tu} : \mathtt{B}}(\to \mathtt{E}) \quad \frac{\Gamma \vdash \mathtt{t} : \mathtt{A} \quad \Gamma \vdash \mathtt{t} : \mathtt{B}}{\Gamma \vdash \mathtt{t} : \mathtt{A} \wedge \mathtt{B}}(\wedge \mathtt{I}) \quad \frac{\Gamma \vdash \mathtt{t} : \mathtt{A}_1 \wedge \mathtt{A}_2}{\Gamma \vdash \mathtt{t} : \mathtt{A}_i \ (i = 1, 2)}(\wedge \mathtt{E})$$

Types: $\mathtt{A} ::= \alpha \mid \omega \mid \mathtt{A} \to \mathtt{A} \mid \mathtt{A} \wedge \mathtt{A}$ Typing environments: $\Gamma ::= \emptyset \mid \Gamma, \mathtt{x} : \mathtt{A} \quad \mathtt{x} \notin \mathtt{dom}(\Gamma)$

Fig. 1. System $\mathcal{C}$

$$\frac{\mathtt{x} : \mathtt{A} \in \Gamma \qquad \sigma \in \mathtt{A}}{\Gamma \vdash \mathtt{x} : \sigma}(\mathtt{var}) \qquad \frac{}{\mathtt{x} : \{\sigma\} \vdash \mathtt{x} : \sigma}(\mathtt{var_r})$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{t} : \tau}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{t} : \mathtt{A} \to \tau}(\to \mathtt{I}) \qquad \frac{\Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{t} : \tau}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{t} : \mathtt{A} \to \tau}(\to \mathtt{I})$$

$$\frac{\Gamma \vdash \mathtt{t} : \{\sigma_i\}_{i \in I} \to \tau \quad (\Gamma \vdash \mathtt{u} : \sigma_i)_{i \in I}}{\Gamma \vdash \mathtt{tu} : \tau}(\to \mathtt{E}) \quad \frac{\Gamma \vdash \mathtt{t} : \{\sigma_i\}_{i \in I} \to \tau \quad (\Delta_i \vdash \mathtt{u} : \sigma_i)_{i \in I}}{\Gamma \cup_{i \in I} \Delta_i \vdash \mathtt{tu} : \tau}(\to \mathtt{E_r})$$

Types:
$\sigma ::= a \mid \mathtt{A} \to \sigma$ (strict types)
$\mathtt{A} ::= \emptyset \mid \{\sigma\} \mid \mathtt{A} \cup \mathtt{A}$ (set types)

Typing environments:
$\Gamma ::= \emptyset \mid \Gamma, \mathtt{x} : \mathtt{A} \quad \mathtt{x} \notin \mathtt{dom}(\Gamma)$
$(\Gamma \cup \Delta)(\mathtt{x}) = \Gamma(\mathtt{x}) \cup \Delta(\mathtt{x})$

Fig. 2. Systems $\mathcal{S}$ and $\mathcal{S}_r$

normal forms from which all and only the head normal forms $\mathtt{t}$ such that $\Gamma \vdash \mathtt{t} : \sigma$ can be generated. Then we extend the system, and consequently the language, in order to consider pairs and projections. We obtain a new system $\mathcal{P}$ which characterizes the (suitably defined) solvability in the extended calculus Λ_π, and we prove that inhabitation is decidable also for this extension.

In the last years, growing interest has been devoted to non idempotent intersection types, since they allow to reason about quantitative properties of terms, both from a syntactical and a semantic point of view. In fact, system $\mathcal{M}$ is not new: it is the well known system R of De Carvalho [5], and it is an instance of the class of the essential λ-models defined in [20], which supplies a logical description of the strongly linear relational λ-models. Some other type assignment systems with non-idempotent intersection have been studied in the literature, for various purposes: to compute a bound for the normalization time of terms [7]), to supply new characterizations of strong normalization [3,12], to study type inference [14,17], to study linearity [13], to characterize solvability in the resource λ-calculus [19,18]. Moreover intersection without idempotency, commutativity nor associativity, has been used to study the game semantics of a typed λ-calculus [8]. A unified model-theoretic approach covering both the relevant and non-relevant cases, and unveiling the relations between them, is presented in [9]. Returning to the inhabitation problem, various restrictions of the classical intersection types system have been shown to have decidable inhabitation [16,4]. The approach is substantially different from that used in this

work, since in all cases intersection is idempotent, and the decidability is obtained by restricting the use of rules ($\wedge$I) and ($\wedge$E), so that the corresponding type assignment does not characterize interesting classes of terms, anymore.

2 The Type Assignment System $\mathcal{M}$

In this section we consider a relevant type system for the λ-calculus having strict intersection types that enjoy associativity and commutativity, but not idempotency. In order to emphasize this last property we represent intersections as multisets of types.

We recall that terms and contexts of the λ-calculus are generated by the following grammars, respectively:

$$\mathtt{t}, \mathtt{u}, \mathtt{v} ::= \mathtt{x} \mid \lambda\mathtt{x}.\mathtt{t} \mid \mathtt{tu} \qquad\qquad \mathtt{C} ::= \Box \mid \lambda\mathtt{x}.\mathtt{C} \mid \mathtt{Ct} \mid \mathtt{tC}$$

Given a context $\mathtt{C}$ and a term $\mathtt{t}$, $\mathtt{C}[\mathtt{t}]$ denotes the term obtained by replacing the unique occurrence of $\Box$ in $\mathtt{C}$ by $\mathtt{t}$, allowing the capture of free variables of $\mathtt{t}$. The β-reduction is given by the rule $(\lambda\mathtt{x}.\mathtt{t})\mathtt{u} \to_\beta \mathtt{t}\{\mathtt{u}/\mathtt{x}\}$ where $\mathtt{t}\{\mathtt{u}/\mathtt{x}\}$ denotes the capture free replacement of $\mathtt{x}$ by $\mathtt{u}$ in $\mathtt{t}$.

Let us recall the notion of head-normal forms (hnf), which is the syntactical counter part of the well known notion of solvability for the λ-calculus. A λ-term *is in* hnf if it is generated by the following grammar $\mathcal{J}$, it *has* hnf if it β-reduces to a hnf.

$$\mathcal{J} ::= \lambda\mathtt{x}.\mathcal{J} \mid \mathcal{K} \qquad\qquad \mathcal{K} ::= \mathtt{x} \mid \mathcal{K}\mathtt{t}$$

Definition 1

1. *The set $\mathcal{TM}$ of types is defined by the following grammar:*

$$\begin{array}{ll} \sigma, \tau, \rho ::= \alpha \mid \mathtt{A} \to \tau & (\textit{types}) \\ \mathtt{A} \quad ::= [\sigma_i]_{i\in I} & (\textit{multiset types}) \end{array}$$

   where α ranges over a countable set of base types and I is a finite, possibly empty, set of indices.
2. *Typing environements, or simply environments, written Γ, Δ, are functions from variables to multiset types, assigning the empty multiset to almost all the variables. The domain of Γ, written $\mathtt{dom}(\Gamma)$, is the set of variables whose image is different from $[\,]$.*
3. *A typing judgement is a triple of the form $\Gamma \vdash \mathtt{t} : \mathtt{A}$. The type system $\mathcal{M}$ is given in Fig. 3. If Π is derivation with conclusion $\Gamma \vdash \mathtt{t} : \sigma$ we write $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$, and call $\mathtt{t}$ the subject of Π. The measure of a derivation Π, written $\mathtt{meas}(\Pi)$, is the number of rule applications in Π. By abuse of notation, $\Gamma \vdash \mathtt{t} : \sigma$ also denotes the existence of some derivation with conclusion $\Gamma \vdash \mathtt{t} : \sigma$.*
4. *A derivation Π is a left-subtree of a derivation Σ if either $\Pi = \Sigma$ or $\Pi \triangleright \Delta \vdash \mathtt{u} : \sigma$ is the major premise of $\Sigma' \triangleright \Delta +_{i\in I} \Delta_i \vdash \mathtt{uv} : \tau$ and Σ' is a left-subtree of Σ.*

Given $\{\Gamma_i\}_{i\in I}$, $+_{i\in I}\Gamma_i$ is the environment mapping $\mathtt{x}$ to $\uplus_{i\in I}\Gamma_i(\mathtt{x})$, where $\uplus$ denotes multiset union, where the resulting environment is the one having empty domain for $I = \emptyset$. The notations $\Gamma + \Delta$ and $\Gamma +_{i\in I} \Delta_i$ are just particular cases of the previous one. $\Gamma \setminus \mathtt{x}$ is the environment assigning $[\,]$ to $\mathtt{x}$, and acting as Γ otherwise; $\mathtt{x}_1 : \mathtt{A}_1, \ldots, \mathtt{x}_n : \mathtt{A}_n$ is the environment assigning $\mathtt{A}_i$ to $\mathtt{x}_i$, for $1 \leq i \leq n$, and $[\,]$ to any other variable.

$$\frac{}{\mathtt{x} : [\rho] \vdash \mathtt{x} : \rho}\ (\mathtt{var}) \qquad \frac{\Gamma \vdash \mathtt{t} : \rho}{\Gamma \setminus \mathtt{x} \vdash \lambda\mathtt{x}.\mathtt{t} : \Gamma(\mathtt{x}) \to \rho}\ (\to \mathtt{I})$$

$$\frac{\Gamma \vdash \mathtt{t} : [\sigma_i]_{i\in I} \to \rho \qquad (\Delta_i \vdash \mathtt{u} : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash \mathtt{tu} : \rho}\ (\to \mathtt{E})$$

Fig. 3. The type assignment system $\mathcal{M}$ for the λ-calculus

Rule $(\to \mathtt{E})$ enables the typability of non strongly normalizing terms, when $I = \emptyset$. For example $\mathtt{x} : [[\,] \to \alpha] \vdash \mathtt{x}((\lambda\mathtt{y}.\mathtt{yy})(\lambda\mathtt{y}.\mathtt{yy})) : \alpha$, where the outermost application is typed neglecting its unsolvable argument. This feature is shared by all the intersection type systems characterizing solvability. The same holds for the fundamental *subject reduction* property: if $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$ and $\mathtt{t} \to_\beta \mathtt{u}$, then $\Pi' \rhd \Gamma \vdash \mathtt{u} : \sigma$. What is peculiar to $\mathcal{M}$ is the fact that the size of Π' is strictly smaller than that of Π, whenever the reduction $\mathtt{t} \to_\beta \mathtt{u}$ takes place in an occurrence of $\mathtt{t}$ which is *typed in* Π. This property, stated in Thm. 1.1 below, allows for an original, combinatorial proof of the fact that typed terms do have $\mathtt{hnf}$.

Definition 2

- *The set* $\mathtt{o}(t)$ *of occurrences of* $\mathtt{t}$ *is the set of contexts* $\mathtt{C}$ *such that there exists a term* $\mathtt{u}$ *verifying* $\mathtt{C}[\mathtt{u}] = \mathtt{t}$, $\mathtt{u}$ *being the subterm of* $\mathtt{t}$ *at the occurrence* $\mathtt{C}$.
- *Given* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$, *the set* $\mathtt{to}(\Pi) \subseteq \mathtt{o}(\mathtt{t})$ *of typed occurrences of* $\mathtt{t}$ *in* Π *is defined by induction on* $\mathtt{meas}(\Pi)$ *as follows:*
  - $\mathtt{to}(\Pi) = \{\Box\}$ *if* Π *is an istance of the axiom.*
  - $\mathtt{to}(\Pi) = \{\Box\} \cup \{\lambda\mathtt{x}.\mathtt{C} \mid \mathtt{C} \in \mathtt{to}(\Pi')\}$ *if the subject of* Π *is* $\lambda\mathtt{x}.\mathtt{t}$ *and* Π' *is the subderivation of* Π *typing* $\mathtt{t}$.
  - $\mathtt{to}(\Pi) = \{\Box\} \cup \{\mathtt{Cu} \mid \mathtt{C} \in \mathtt{to}(\Pi')\} \cup (\bigcup_{i\in I}\{\mathtt{tC} \mid \mathtt{C} \in \mathtt{to}(\Pi'_i)\})$ *if the subject of* Π *is* $\mathtt{tu}$, Π' *is the subderivation of* Π *typing* $\mathtt{t}$, *and* Π'_i, *for* $i \in I$, *are the sub-derivations of* Π *typing* $\mathtt{u}$.
- *Given* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$, *we say that* $\mathtt{t}$ *is in* Π*-normal form if for all* $\mathtt{C} \in \mathtt{to}(\Pi)$, *the subterm of* $\mathtt{t}$ *at the occurrence* $\mathtt{C}$ *is not a redex.*

Theorem 1

1. *(Subject reduction)* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$ *and* $\mathtt{t} \to_\beta \mathtt{u}$ *imply* $\Pi' \rhd \Gamma \vdash \sigma$ *where* $\mathtt{meas}(\Pi') \leq \mathtt{meas}(\Pi)$. *Moreover, if the reduced redex is typed in* Π, *then* $\mathtt{meas}(\Pi') < \mathtt{meas}(\Pi)$.
2. *(Characterization)* $\Gamma \vdash \mathtt{t} : \sigma$ *if and only if* $\mathtt{t}$ *has* $\mathtt{hnf}$.

Proof. For 1 see [20] and for 2 see [5].

3 Inhabitation for System $\mathcal{M}$

The system $\mathcal{M}$ allows to type a term without giving types to all its subterms through the rule $(\to \mathtt{E})$ in case $I = \emptyset$. So in order to reconstruct the subject of a derivation we need a notation for these untyped subterms. We will use the standard notion of *approximate normal form* [1], which can be defined through the following grammar:

$$\mathtt{a}, \mathtt{b}, \mathtt{c} \ ::= \ \Omega \mid \mathcal{N} \qquad \mathcal{N} \ ::= \ \lambda \mathtt{x}.\mathcal{N} \mid \mathcal{L} \qquad \mathcal{L} \ ::= \ \mathtt{x} \mid \mathcal{L}\mathtt{a}$$

Approximate normal forms are ordered by the smallest contextual order $\leq$ such that $\Omega \leq \mathtt{a}$, for all $\mathtt{a}$. We write $\mathtt{a} \leq \mathtt{t}$ when the term $\mathtt{t}$ is obtained from $\mathtt{a}$ by replacing all the occurrences of Ω by terms.

Let $\mathcal{A}(\mathtt{t}) = \{\mathtt{a} \mid \exists \mathtt{u}\ \mathtt{t} \to_\beta^* \mathtt{u} \text{ and } \mathtt{a} \leq \mathtt{u}\}$ be the set of *approximants* of the λ-term $\mathtt{t}$, and let $\bigvee$ denote the least upper bound w.r.t. $\leq$. We write $\uparrow_{i \in I} \mathtt{a}_i$ to denote that $\bigvee\{\mathtt{a}_i\}_{i \in I}$ does exist. It is easy to check that, for every $\mathtt{t}$ and $\mathtt{a}_1, \ldots \mathtt{a}_n \in \mathcal{A}(\mathtt{t})$, $\uparrow_{i \in \{1,\ldots,n\}} \mathtt{a}_i$. An approximate normal form $\mathtt{a}$ is a *head subterm* of $\mathtt{b}$ if either $\mathtt{b} = \mathtt{a}$ or $\mathtt{b} = \mathtt{cc}'$ and $\mathtt{a}$ is a head subterm of $\mathtt{c}$. System $\mathcal{M}$ gives types to approximate normal forms, by simply assuming that no type can be assigned to the constant Ω. It is easy to check that, if $\Gamma \vdash \mathtt{a} : \sigma$ and $\mathtt{a} \leq \mathtt{b}$ (resp. $\mathtt{a} \leq \mathtt{t}$) then $\Gamma \vdash \mathtt{b} : \sigma$ (resp. $\Gamma \vdash \mathtt{t} : \sigma$). Given $\Pi \rhd \Gamma \vdash \mathtt{t} : \tau$, where $\mathtt{t}$ is in Π-normal form, we denote by $\mathcal{A}(\Pi)$ the minimal approximant $\mathtt{b}$ of $\mathtt{t}$ such that $\Pi \rhd \Gamma \vdash \mathtt{b} : \tau$. Formally,

Definition 3. *Given* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$, *where* $\mathtt{t}$ *is in* Π*-normal form,* $\mathcal{A}(\Pi) \in \mathcal{A}(\mathtt{t})$ *is defined by induction on* $\mathtt{meas}(\Pi)$ *as follows:*

- *If* $\Pi \rhd \Gamma \vdash \mathtt{x} : \rho$, *then* $\mathcal{A}(\Pi) = \mathtt{x}$.
- *If* $\Pi \rhd \Gamma \vdash \lambda \mathtt{x}.\mathtt{t} : \mathtt{A} \to \rho$ *follows from* $\Pi' \rhd \Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{t} : \rho$, *then* $\mathcal{A}(\Pi) = \lambda \mathtt{x}.\mathcal{A}(\Pi')$, $\mathtt{t}$ *being in* Π'*-normal form.*
- *If* $\Pi \rhd \Gamma = \Gamma' +_{i \in I} \Delta_i \vdash \mathtt{tu} : \rho$ *follows from* $\Pi' \rhd \Gamma \vdash \mathtt{t} : [\sigma_i]_{i \in I} \to \rho$ *and* $(\Pi'_i \rhd \Delta_i \vdash \mathtt{u} : \sigma_i)_{i \in I}$, *then* $\mathcal{A}(\Pi) = \mathcal{A}(\Pi')(\bigvee_{i \in I} \mathcal{A}(\Pi'_i))$ *(remark that* $\mathtt{t}$ *is in* Π'*-normal form,* $\mathtt{u}$ *if in* Π'_i*-normal form, for all* $i \in I$, *and that* $\uparrow_{i \in I} \mathcal{A}(\Pi'_i)$, *since* $\mathcal{A}(\Pi'_i) \in \mathcal{A}(\mathtt{u})$, *for all* $i \in I$*).*

Remark that, in the final item of the definition above, the approximate normal form corresponding to the case $I = \emptyset$ is $\mathcal{A}(\Pi')\Omega$.

A simple induction on $\mathtt{meas}(\Pi)$ allows to show the following:

Proposition 1. *If* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$ *and* $\mathtt{t}$ *is in* Π*-normal form, then* $\Pi \rhd \Gamma \vdash \mathcal{A}(\Pi) : \sigma$.

3.1 The Inhabitation Algorithm

The inhabitation rules are given in Fig. 4. The algorithm, given an environment Γ and a type σ, builds the set $\mathtt{T}(\Gamma,\sigma)$ containing *all* the approximate normal forms $\mathtt{a}$ such that there exists a derivation $\Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$, with $\mathtt{a} = \mathcal{A}(\Pi)$, then stops. The algorithm uses two auxiliary predicates, namely $\mathtt{TI}(\Gamma,[\sigma_i]_{i\in I})$ and $\mathtt{H}^{\Delta}_{\mathtt{a}}(\Gamma,\sigma)\triangleright\tau$. The set $\mathtt{TI}(\Gamma,[\sigma_i]_{i\in I})$ contains all the approximate normal forms $\mathtt{a} = \bigvee_{i\in I}\mathtt{a}_i$ such that $\Gamma = +_{i\in I}\Gamma_i$, $\mathtt{a}_i \in \mathtt{T}(\Gamma_i,\sigma_i)$ for all $i\in I$, and $\uparrow_{i\in I}\mathtt{a}_i$. Finally, $\mathtt{H}^{\Delta}_{\mathtt{a}}(\Gamma,\sigma)\triangleright\tau$ contains all the approximate normal forms $\mathtt{b}$ such that $\mathtt{a}$ is a head subterm of $\mathtt{b}$, and such that if $\mathtt{a}\in\mathtt{T}(\Delta,\sigma)$ then $\mathtt{b}\in\mathtt{T}(\Gamma+\Delta,\tau)$.

$$\frac{\mathtt{a}\in\mathtt{T}(\Gamma+(\mathtt{x}:\mathtt{A}),\tau) \qquad \mathtt{x}\notin\mathtt{dom}(\Gamma)}{\lambda\mathtt{x}.\mathtt{a}\in\mathtt{T}(\Gamma,\mathtt{A}\to\tau)}\ (\mathtt{Abs})$$

$$\frac{(\mathtt{a}_i\in\mathtt{T}(\Gamma_i,\sigma_i))_{i\in I} \qquad \uparrow_{i\in I}\mathtt{a}_i}{\bigvee_{i\in I}\mathtt{a}_i\in\mathtt{TI}(+_{i\in I}\Gamma_i,[\sigma_i]_{i\in I})}\ (\mathtt{Union})$$

$$\frac{\mathtt{a}\in\mathtt{H}^{\mathtt{x}:[\sigma]}_{\mathtt{x}}(\Gamma,\sigma)\triangleright\tau}{\mathtt{a}\in\mathtt{T}(\Gamma+(\mathtt{x}:[\sigma]),\tau)}\ (\mathtt{Head}) \qquad \frac{\sigma=\tau}{\mathtt{a}\in\mathtt{H}^{\Delta}_{\mathtt{a}}(\emptyset,\sigma)\triangleright\tau}\ (\mathtt{Final})$$

$$\frac{\Gamma=\Gamma_0+\Gamma_1 \qquad \mathtt{b}\in\mathtt{TI}(\Gamma_0,\mathtt{A}) \qquad \mathtt{a}\in\mathtt{H}^{\Delta+\Gamma_0}_{\mathtt{cb}}(\Gamma_1,\sigma)\triangleright\tau}{\mathtt{a}\in\mathtt{H}^{\Delta}_{\mathtt{c}}(\Gamma,\mathtt{A}\to\sigma)\triangleright\tau}\ (\mathtt{Prefix})$$

Fig. 4. The inhabitation algorithm for the λ-calculus

Notice the particular case $I=\emptyset$ in (Union), which gives $\Omega\in\mathtt{TI}(\emptyset,[\,])$, where $\emptyset$ denotes the environement having empty domain. The algorithm is not an obvious extension of the classical inhabitation algorithm for simple types [2,10]. In particular, when restricted to simple types, it reconstructs all the normal forms inhabiting a given type, while the original algorithm reconstructs just the *long η-normal forms*. This is achieved thanks to a non deterministic behaviour, illustrated in the Example 1.1 below.

Example 1

1. Let $\Gamma=\emptyset$ and $\sigma=[[\alpha]\to\alpha]\to[\alpha]\to\alpha$. Given input (Γ,σ), the algorithm can have the following two behaviours:
   (1) Choosing the sequence of rules: (Abs), (Abs), (Head), (Prefix), (Final) the final approximant is $\lambda\mathtt{xy.xy}$;
   (2) Choosing the sequence of rules: (Abs), (Head), (Final) the final approximant is $\lambda\mathtt{x.x}$.
2. Let $\Gamma=\emptyset$ and $\sigma=[[]\to\alpha]\to\alpha$. Given input (Γ,σ), by the sequence of rules: (Abs), (Head), (Prefix), ((Union), (Final)) we obtain $\lambda\mathtt{x.x}\Omega$.

Definition 4. *In order to show that the inhabitation algorithm terminates, we define a measure on types and environments, as follows:*

$\#(\alpha) = 1$ $\quad$ $\#([\sigma_i]_{i\in I}) = \sum_{i\in I}\#(\sigma_i) + 1$

$\#(\mathtt{A} \to \rho) = \#(\mathtt{A}) + \#(\rho) + 1$ $\quad$ $\#(\Gamma) = \sum_{\mathtt{x}\in \mathtt{dom}(\Gamma)} \#(\Gamma(\mathtt{x}))$

The measure is then be extended to the judgements of the algorithm:

$\#(\mathtt{T}(\Gamma,\rho)) = \#(\mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma,\rho) \triangleright \tau) = \#(\Gamma) + \#(\rho)$

$\#(\mathtt{TI}(\Gamma,\mathtt{A})) = \#(\Gamma) + \#(A)$

Lemma 1 (Termination). *The inhabitation algorithm terminates.*

Proof Hint. To each call $C \in \{\mathtt{T}(\_,\_), \mathtt{TI}(\_,\_), \mathtt{H}^{\_}_{\_}(\_,\_) \triangleright \_\}$ of the algorithm, we associate a tree T_C as follows: nodes are labeled with elements of C. A node n' is a son of n iff there exists some instance of a rule having n as conclusion and n' as premise. Thus, all possible runs of C are encoded in the tree T_C, which is finitely branching. Moreover, it is easy to see that the measure $\#()$ strictly decreases along the branches of T_C, so that every branch has finite depth. Hence, T_C is finite by König's Lemma, *i.e.* the algorithm terminates.

Soundness and completeness of the inhabitation algorithm follow from the following Lemma, relating typings of approximate normal forms in system $\mathcal{M}$ and runs of the algorithm:

Lemma 2. $\mathtt{a} \in \mathtt{T}(\Gamma,\sigma) \Leftrightarrow \exists \Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ *such that* $\mathtt{a} = \mathcal{A}(\Pi)$.

Proof. ($\Rightarrow$): We prove the following statements, by induction on the rules in Fig. 4:

a) $\mathtt{a} \in \mathtt{T}(\Gamma,\sigma) \Rightarrow \exists \Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ such that $\mathtt{a} = \mathcal{A}(\Pi)$.

b) $\mathtt{a} \in \mathtt{TI}(+_{i\in I}\Gamma_i, [\sigma_i]_{i\in I}) \Rightarrow \exists (\Pi_i \triangleright \Gamma_i \vdash \mathtt{a}_i : \sigma_i)_{i\in I}$ such that $\mathtt{a}_i = \mathcal{A}(\Pi_i)$, for $i \in I$, $\uparrow_{i\in I} \mathtt{a}_i$ and $\mathtt{a} = \bigvee_{i\in I} \mathtt{a}_i$.

c) $\mathtt{a} \in \mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma,\sigma) \triangleright \tau \Rightarrow$ there exists a function F associating to each derivation $\Sigma \triangleright \Delta \vdash \mathtt{b} : \sigma$ such that $\mathtt{b} = \mathcal{A}(\Sigma)$, a derivation $\Pi \triangleright \Gamma + \Delta \vdash \mathtt{a} : \tau$ such that $\mathtt{a} = \mathcal{A}(\Pi)$.

If $\lambda\mathtt{x}.\mathtt{a} \in \mathtt{T}(\Gamma, \mathtt{A} \to \tau)$ follows from $\mathtt{a} \in \mathtt{T}(\Gamma + (\mathtt{x} : \mathtt{A}), \tau)$ by (`Abs`), then we conclude by the *i.h.*(a) and by an application of $(\to \mathtt{I})$.

If $\bigvee_{i\in I} \mathtt{a}_i \in \mathtt{TI}(+_{i\in I}\Gamma_i, [\sigma_i]_{i\in I})$ follows from $(\mathtt{a}_i \in \mathtt{T}(\Gamma_i,\sigma_i))_{i\in I}$ and $\uparrow_{i\in I} \mathtt{a}_i$ by (`Union`), then by *i.h.*(a), there exist $(\Pi_i \triangleright \Gamma_i \vdash \mathtt{a}_i : \sigma_i)_{i\in I}$ such that for all $i \in I$, $\mathtt{a}_i = \mathcal{A}(\Pi_i)$, and we are done.

If $\mathtt{a} \in \mathtt{T}(\Gamma + (\mathtt{x} : [\sigma]), \tau)$ follows from $\mathtt{a} \in \mathtt{H}^{\mathtt{x}:[\sigma]}_{\mathtt{x}}(\Gamma,\sigma) \triangleright \tau$ by (`Head`), then the i. h. (c) provides a function associating to each derivation $\Sigma \triangleright \mathtt{x} : [\sigma] \vdash \mathtt{x} : \sigma$ a derivation $\Pi \triangleright \Gamma + \mathtt{x} : [\sigma] \vdash \mathtt{a} : \tau$ such that $\mathtt{a} = \mathcal{A}(\Pi)$, since $\mathtt{x} = \mathcal{A}(\Sigma)$. Applying this function to the unique derivation of $\mathtt{x} : [\sigma] \vdash \mathtt{x} : \sigma$, we get the suitable typing of $\mathtt{a}$.

If $\mathtt{a} \in \mathtt{H}^{\Delta}_{\mathtt{a}}(\emptyset,\sigma) \triangleright \tau$ follows from $\sigma = \tau$ by (`Final`), then the identity function satisfies the requirements of (c).

Finally, if $\mathtt{a} \in \mathtt{H}_{\mathtt{c}}^{\Delta}(\Gamma_0 + \Gamma_1, \mathtt{A} \to \sigma) \triangleright \tau$ follows from $\mathtt{b} \in \mathtt{TI}(\Gamma_0, \mathtt{A})$ and $\mathtt{a} \in \mathtt{H}_{\mathtt{cb}}^{\Delta+\Gamma_0}(\Gamma_1, \sigma) \triangleright \tau$ by (Prefix), then we have to provide a function F associating to each derivation $\Sigma \triangleright \Delta \vdash \mathtt{c} : \mathtt{A} \to \sigma$ such that $\mathtt{c} = \mathcal{A}(\Sigma)$, a derivation $\Pi \triangleright \Delta + \Gamma_0 + \Gamma_1 \vdash \mathtt{a} : \tau$ such that $\mathtt{a} = \mathcal{A}(\Pi)$. To begin with, the *i.h.*(b) applied to $\mathtt{b} \in \mathtt{TI}(\Gamma_0, \mathtt{A})$ provides a family of derivations $(\Pi_i \triangleright \Gamma_0^i \vdash \mathtt{b}_i : \sigma_i)_{i\in I}$ such that $\mathtt{b}_i = \mathcal{A}(\Pi_i)$, $\Gamma_0 = +_{i\in I}\Gamma_0^i$, $A = [\sigma_i]_{i\in I}$, $\uparrow_{i\in I} \mathtt{b}_i$ and $\mathtt{b} = \bigvee_{i\in I} \mathtt{b}_i$. Rule $(\to \mathtt{E})$ with premises Σ and $\{\Pi_i\}_{i\in I}$, gives a type derivation $\Pi' \triangleright \Delta + \Gamma_0 \vdash \mathtt{cb} : \sigma$, such that $\mathtt{cb} = \mathcal{A}(\Pi')$. Then, the *i.h.*(c) applied to $\mathtt{a} \in \mathtt{H}_{\mathtt{cb}}^{\Delta+\Gamma_0}(\Gamma_1, \sigma) \triangleright \tau$ provides a function F' such that $F'(\Pi') \triangleright \Delta + \Gamma_0 + \Gamma_1 \vdash \mathtt{a} : \tau$ and $\mathtt{a} = \mathcal{A}(F'(\Pi'))$.To conclude, we set $F(\Sigma) = F'(\Pi')$.

($\Leftarrow$): We prove the following statements, by induction on the definition of $\mathcal{A}(\Pi)$ (Def. 3):

1. Given $\Sigma \triangleright \Delta \vdash \mathtt{b} : \tau$ and $\Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$, if $\mathtt{b} = \mathcal{A}(\Sigma)$ and $\mathtt{a} = \mathcal{A}(\Pi)$ are $\mathcal{L}$-approximate normal forms, and Σ is a left-subtree of Π, then there exists Γ' s.t. $\Gamma = \Gamma' + \Delta$ and $\mathtt{H}_{\mathtt{a}}^{\Delta+\Gamma'}(\Theta, \sigma) \triangleright \rho \subseteq \mathtt{H}_{\mathtt{b}}^{\Delta}(\Theta + \Gamma', \tau) \triangleright \rho$.
2. $\Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ and $\mathtt{a} = \mathcal{A}(\Pi)$ imply $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma)$.

1. If $\mathtt{a} = \mathtt{x}$, then Π is an instance of the axiom (var); Σ being a left subtree of Π, we get $\Sigma = \Pi$, $\mathtt{b} = \mathtt{x}$, $\Gamma' = \emptyset$, $\sigma = \tau$ and the inclusion $\mathtt{H}_{\mathtt{b}}^{\Delta+\Gamma'}(\Theta, \sigma) \triangleright \rho \subseteq \mathtt{H}_{\mathtt{a}}^{\Delta}(\Theta + \Gamma', \tau) \triangleright \rho$ holds trivially.

If $\mathtt{a} = \mathtt{ca}'$, $\mathtt{c}$ being a $\mathcal{L}$-approximate normal form, then the last rule of Π is an instance of $(\to \mathtt{E})$, with premises $\Pi' \triangleright \Gamma'' \vdash \mathtt{c} : [\sigma_i]_{i\in I} \to \sigma$ and $(\Pi_i \triangleright \Gamma_i \vdash \mathtt{a}' : \sigma_i)_{i\in I}$, so that $\Gamma = \Gamma'' +_{i\in I} \Gamma_i$; moreover $\Sigma \triangleright \Delta \vdash \mathtt{b} : \tau$ is also a left-subtree of Π'. We have in this case $\mathtt{a}' = \bigvee_{i\in I} \mathcal{A}(\Pi_i)$, where by the *i.h.*(2), $\mathcal{A}(\Pi_i) \in \mathtt{T}(\Gamma_i, \sigma_i)$. Then $\mathtt{H}_{\mathtt{ca}'}^{\Delta+\Gamma''+_{i\in I}\Gamma_i}(\Theta, \sigma) \triangleright \rho \subseteq_{(\mathtt{Prefix})} \mathtt{H}_{\mathtt{c}}^{\Delta+\Gamma''}(\Theta +_{i\in I} \Gamma_i, [\sigma_i]_{i\in I} \to \sigma) \triangleright \rho \subseteq_{i.h.(1)} \mathtt{H}_{\mathtt{b}}^{\Delta}(\Theta + \Gamma'' +_{i\in I} \Gamma_i, \sigma) \triangleright \rho$.

2. If $\mathtt{a}$ is a $\mathcal{L}$-approximate normal form, then $\exists \tau$ s.t. $\Gamma = \Gamma_0 + (\mathtt{x} : [\tau])$ and the type derivation $\mathtt{x} : [\tau] \vdash \mathtt{x} : \tau$ is a left subtree of $\Gamma_0 + (\mathtt{x} : [\tau]) \vdash \mathtt{a} : \sigma$. Then we have $\mathtt{a} \in_{(\mathtt{Final})} \mathtt{H}_{\mathtt{a}}^{\Gamma}(\emptyset, \sigma) \triangleright \sigma \subseteq_{\mathrm{Point}(1)} \mathtt{H}_{\mathtt{x}}^{\mathtt{x}:[\tau]}(\Gamma_0, \tau) \triangleright \sigma \subseteq_{(\mathtt{Head})} \mathtt{T}(\Gamma_0 + \{\mathtt{x} : [\tau]\}, \sigma)$. Otherwise, $\mathtt{a} = \lambda\mathtt{x}.\mathtt{a}'$, and we conclude by the *i.h.*(2) on $\mathtt{a}'$.

Theorem 2 (Soundness and Completeness)

1. *If* $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma)$ *then, for all* $\mathtt{t}$ *such that* $\mathtt{a} \leq \mathtt{t}$, $\Gamma \vdash \mathtt{t} : \sigma$.
2. *If* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$ *then there exists* $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ *such that* $\mathtt{t}'$ *is in* Π'*-normal form, and* $\mathcal{A}(\Pi') \in \mathtt{T}(\Gamma, \sigma)$.

Proof. Soundness follows from Lem. 2 ($\Rightarrow$) and the remark that, if $\Gamma \vdash \mathtt{a} : \sigma$ and $\mathtt{a} \leq \mathtt{t}$, then $\Gamma \vdash \mathtt{t} : \sigma$. Completeness follows from Thm. 1.1, ensuring that given $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$, there exists $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ such that $\mathtt{t}'$ is in Π'-normal form, then from Prop. 1 and Lem. 2 ($\Leftarrow$).

4 Adding Pairs and Projections

The language Λ_π is an extension of λ-calculus with pairs and projections. Its terms and contexts are defined by the following grammars:

$$\begin{array}{ll} \mathtt{t},\mathtt{u},\mathtt{v} ::= & \mathtt{x} \mid \lambda\mathtt{x}.\mathtt{t} \mid \mathtt{t}\mathtt{u} \mid \pi_1\mathtt{t} \mid \pi_2\mathtt{t} \mid \langle \mathtt{t},\mathtt{u}\rangle \\ \mathtt{C} \quad ::= & \Box \mid \lambda\mathtt{x}.\mathtt{C} \mid \mathtt{C}\mathtt{t} \mid \mathtt{t}\mathtt{C} \mid \pi_1\mathtt{C} \mid \pi_2\mathtt{C} \mid \langle \mathtt{t},\mathtt{C}\rangle \mid \langle \mathtt{C},\mathtt{t}\rangle \end{array}$$

The reduction relation, also denoted by $\to$, is the contextual closure of the following rules:

$(\lambda\mathtt{x}.\mathtt{t})\mathtt{u} \to_\beta \mathtt{t}\{\mathtt{u}/\mathtt{x}\} \qquad \pi_1\langle \mathtt{t},\mathtt{u}\rangle \to_\pi \mathtt{t} \qquad \pi_2\langle \mathtt{t},\mathtt{u}\rangle \to_\pi \mathtt{u}$

As usual, $\to^*$ denotes the reflexive-transitive closure of $\to$. We write $\overline{\mathtt{t}}$ and $\overline{\pi}$ (resp. $\overline{\mathtt{t}}_n$ and $\overline{\pi}_n$) to denote a possibly empty sequence (resp. a sequence of length n) of terms and projections, respectively.

Λ_π inherits from the λ-calculus important properties, such as confluence. Solvability is defined as usual, but pairs are solvable independently from their content, since we want to consider them as lazy data structures:

Definition 5

1. *A head context is a context of the shape:* $(\lambda\overline{\mathtt{x}}.\Box)\overline{\mathtt{t}}$*;*
2. *A term* $\mathtt{t}$ *is solvable if and only if there is a head context* $\mathtt{C}$ *such that* $\mathtt{C}[\mathtt{t}] \to^* \mathtt{u}$*, where* $\mathtt{u}$ *is either a pair* $\langle \mathtt{u}_1,\mathtt{u}_2\rangle$ *or the identity* $\mathtt{I}$*.*

We will prove that solvability can be syntactically characterized by the notion of $\mathtt{hnf}$, defined by the following grammar:

$$\mathcal{J} ::= \lambda\mathtt{x}.\mathcal{J} \mid \langle t,t\rangle \mid \mathcal{P} \qquad\qquad \mathcal{P} ::= \mathtt{x} \mid \mathcal{P}\mathtt{t} \mid \pi_1\mathcal{P} \mid \pi_2\mathcal{P}$$

The *head variable* of a term in $\mathcal{P}$ is defined by: $\mathtt{x}$ is the head variable of $\mathtt{x}$ and $\mathtt{x}$ is the head variable of $\mathtt{u}\mathtt{t}$ (resp. $\pi_i\mathtt{u}$, $i = 1,2$) if $\mathtt{x}$ is the head variable of $\mathtt{u}$. A term *has* $\mathtt{hnf}$ if it reduces to a $\mathtt{hnf}$.

We will prove now that if a term has $\mathtt{hnf}$ then it is solvable. The converse will be proved through a suitable type assignment system, which will be introduced in the next subsection.

Lemma 3. *It* $\mathtt{t}$ *has* $\mathtt{hnf}$ *then it is solvable.*

Proof. Let us define, for every sequence of projections $\overline{\pi}$ and every term $\mathtt{t}$, the term $\mathtt{P}_{\overline{\pi}}(\mathtt{t})$ such that $\overline{\pi}\mathtt{P}_{\overline{\pi}}(\mathtt{t}) \to^* \mathtt{t}$. If $\overline{\pi}$ is the empty sequence, $\mathtt{P}_{\overline{\pi}}(\mathtt{t}) = \mathtt{t}$, if $\overline{\pi} = \overline{\pi'}\pi_1$ (resp. $\overline{\pi'}\pi_2$), then $\mathtt{P}_{\overline{\pi}}(\mathtt{t}) = \langle \mathtt{P}_{\overline{\pi'}}(\mathtt{t}), I\rangle$ (resp. $\langle I, \mathtt{P}_{\overline{\pi'}}(\mathtt{t})\rangle$). Moreover, let Π and J be two lists of same lenght, the first containing sequences of projections and the second one natural numbers. The term $\nabla(\Pi, J)$ is defined inductively as follows:
if Π and J are empty, then $\nabla(\Pi, J) = \mathtt{I}$ else if $\Pi = \overline{\pi} :: \Pi'$ and $J = j :: J'$ then $\nabla(\Pi, J) = \mathtt{P}_{\overline{\pi}}(\lambda\overline{\mathtt{y}}_j.\nabla(\Pi', J'))$.
Now we prove that, if $\mathtt{t}$ has a $\mathcal{P}$-$\mathtt{hnf}$ and $\mathtt{x}$ is its head variable, then for all Π, J there exists a term $O^{\mathtt{t}}_{\nabla(\Pi,J)}$ such that $\mathtt{t}\{O^{\mathtt{t}}_{\nabla(\Pi,J)}/\mathtt{x}\} \to^* \nabla(\Pi, J)$. If $\mathtt{t} = \mathtt{x}\overline{\mathtt{t}}_n$, then $O^{\mathtt{t}}_{\nabla(\Pi,J)} = \lambda\overline{\mathtt{y}}_n.\nabla(\Pi, J)$.
If $\mathtt{t} = \overline{\pi}\mathtt{u}\mathtt{t}_n$, then $O^{\mathtt{t}}_{\nabla(\Pi,J)} = O^{\mathtt{u}}_{\nabla(\overline{\pi}::\Pi,n::J)}$. In fact, $\overline{\pi}\mathtt{u}\mathtt{t}_n\{O^{\mathtt{u}}_{\nabla(\overline{\pi}::\Pi,n::J)}/\mathtt{x}\}$
$\to^* \overline{\pi}(\mathtt{u}\{O^{\mathtt{u}}_{\nabla(\overline{\pi}::\Pi,n::J)}/\mathtt{x}\})\mathtt{t}'_n \to^*_{(i.h.)} \overline{\pi}\nabla(\overline{\pi} :: \Pi, n :: J)\overline{\mathtt{t}'}_n$ which reduces to $\overline{\pi}\mathtt{P}_{\overline{\pi}}(\lambda\overline{\mathtt{y}}_n.\nabla(\Pi, J))\overline{\mathtt{t}'}_n \to^* \nabla(\Pi, J)$, where $\overline{\mathtt{t}'}_n = \overline{\mathtt{t}}_n\{O^{\mathtt{u}}_{\nabla(\mathtt{P}_{\overline{\pi}}::\Pi,n::J)}/\mathtt{x}\}$. Now, to show the statement of the lemma we proceed by cases.

If $\mathtt{t} \in \mathcal{P}$, then let $\mathtt{x}$ be the head of $\mathtt{t}$ and let $\mathtt{C} = (\lambda \mathtt{x}.\Box)O^{\mathtt{t}}_{\nabla(\epsilon,\epsilon)}$. By the previous point we have $\mathtt{C}[\mathtt{t}] = (\lambda \mathtt{x}.\mathtt{t})O^{\mathtt{t}}_{\nabla(\epsilon,\epsilon)} \to \mathtt{t}\{O^{\mathtt{t}}_{\nabla(\epsilon,\epsilon)}/\mathtt{x}\} \to^* \nabla(\epsilon,\epsilon) = \mathtt{I}$. If $\mathtt{t} = \lambda\overline{\mathtt{y}}_n.\langle \mathtt{u}, \mathtt{v}\rangle$, then let $\mathtt{C} = \Box\overline{\mathtt{I}}_n$. Then $\mathtt{C}[\mathtt{t}] \to^* \langle \mathtt{u}, \mathtt{v}\rangle$. If $\mathtt{t} = \lambda\overline{\mathtt{y}}_n.\mathtt{u}$, where $\mathtt{u} \in \mathcal{P}$, then let $\mathtt{x}$ be the head of $\mathtt{u}$. If $\mathtt{x} \notin \overline{\mathtt{y}}_n$, we let $\mathtt{C} = (\lambda \mathtt{x}.\Box)O^{\mathtt{u}}_{\nabla(\epsilon,\epsilon)}\overline{\mathtt{I}}_n$. Then $\mathtt{C}[\mathtt{t}] \to^* (\lambda\overline{\mathtt{y}}_n.\mathtt{u}\{O^{\mathtt{u}}_{\nabla(\epsilon,\epsilon)}/\mathtt{x}\})\overline{\mathtt{I}}_n \to^* (\lambda\overline{\mathtt{y}}_n.\mathtt{I})\overline{\mathtt{I}}_n \to^* \mathtt{I}$. If $\mathtt{x} = \mathtt{y}_i$, we let $\mathtt{C} = (\Box)\mathtt{I}_{i-1}O^{\mathtt{u}}_{\nabla(\epsilon,\epsilon)}\overline{\mathtt{I}}_{n-i}$. Then $\mathtt{C}[\mathtt{t}] \to^* \mathtt{u}\{O^{\mathtt{u}}_{\nabla(\epsilon,\epsilon)}/\mathtt{x}\}\{\overline{\mathtt{I}}_{n-1}/\overline{\mathtt{y}}_{n-1}\} \to^* \mathtt{I}\{\overline{\mathtt{I}}_{n-1}/\overline{\mathtt{y}}_{n-1}\} = \mathtt{I}$.

4.1 The Type Assignment System $\mathcal{P}$

We now present the system $\mathcal{P}$, an extension of system $\mathcal{M}$ which assigns types to Λ_π-terms in such a way that typability coincides with solvability.

Definition 6

1. *The set $\mathcal{TP}$ of types is extended using the following grammar:*

$$\begin{array}{lll} \sigma, \tau, \rho ::= \omega_\times \mid \alpha \mid \mathtt{A} \to \tau \mid \times_1(\tau) \mid \times_2(\tau) & & \textit{(types)} \\ \mathtt{A} \quad ::= [\sigma_i]_{i\in I} & & \textit{(multiset types)} \end{array}$$

   where $\omega_\times$ is a type constant.
2. *A typing judgement is a triple of the form $\Gamma \vdash \mathtt{t} : \mathtt{A}$, where Γ is a typing environment defined as in Def. 1. The type system $\mathcal{P}$ is obtained by adding to the system $\mathcal{M}$ the rules given in Fig. 5.*
3. *The definition of left-subtree of a derivation Σ is as Def. 1.4, with the the following additional cases:*
   - *a derivation $\Pi \triangleright \Delta \vdash \mathtt{t} : \sigma$ is a left subtree of a derivation Σ if Π is the premise of $\Sigma' \triangleright \Delta \vdash \langle \mathtt{t}, \mathtt{u}\rangle : \times_1(\sigma)$ and Σ' is a left subtree of Σ (and similarly for the case $\times_2(\sigma)$).*
   - *a derivation $\Pi \triangleright \Delta \vdash \mathtt{t} : \times_i(\sigma)$ is a left subtree of a derivation Σ if Π is the premise of $\Sigma' \triangleright \Delta \vdash \pi_i(\mathtt{t}) : \sigma$ and Σ' is a left subtree of Σ, for $i = 1, 2$.*

The constant $\omega_\times$ is a universal type for pairs, making all pairs typable. The system is relevant, and it enjoys both subject reduction (in a weighted version) and subject expansion. The notions of occurrences and typed occurrences are extended as expected from those of Def. 2.

$$\frac{}{\vdash \langle \mathtt{t}, \mathtt{u}\rangle : \omega_\times}\ (\mathtt{emptypair})$$

$$\frac{\Gamma \vdash \mathtt{t} : \sigma}{\Gamma \vdash \langle \mathtt{t}, \mathtt{u}\rangle : \times_1(\sigma)}\ (\mathtt{pair1}) \qquad \frac{\Gamma \vdash \mathtt{u} : \tau}{\Gamma \vdash \langle \mathtt{t}, \mathtt{u}\rangle : \times_2(\tau)}\ (\mathtt{pair2})$$

$$\frac{\Gamma \vdash \mathtt{t} : \times_1(\sigma)}{\Gamma \vdash \pi_1\mathtt{t} : \sigma}\ (\mathtt{proj1}) \qquad \frac{\Gamma \vdash \mathtt{t} : \times_2(\sigma)}{\Gamma \vdash \pi_2\mathtt{t} : \sigma}\ (\mathtt{proj2})$$

Fig. 5. Typing rules for pairs in $\mathcal{P}$

Lemma 4

1. *(Subject reduction)* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$ *and* $\mathtt{t} \to \mathtt{u}$ *imply* $\Pi' \rhd \Gamma \vdash \mathtt{u} : \sigma$ *where* $\mathtt{meas}(\Pi') \leq \mathtt{meas}(\Pi)$. *In particular, if the reduced redex is typed, then* $\mathtt{meas}(\Pi') < \mathtt{meas}(\Pi)$.
2. *(Subject expansion)* $\Gamma \vdash \mathtt{u} : \sigma$ *and* $\mathtt{t} \to \mathtt{u}$ *imply* $\Gamma \to \mathtt{t} : \sigma$.

Lemma 5. *Let* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$. *Then* $\mathtt{t}$ *has* $\mathtt{hnf}$.

Proof. By induction on $\mathtt{meas}(\Pi)$ using Lem. 4.1.

The solvability characterization is proved in the next theorem.

Theorem 3. *The following statements are equivalent: (1)* $\mathtt{t}$ *is solvable; (2)* $\mathtt{t}$ *has* $\mathtt{hnf}$*; (3)* $\mathtt{t}$ *is typable in system* $\mathcal{P}$.

Proof. $2 \Rightarrow 1$ holds by Lem. 3 and $3 \Rightarrow 2$ holds by Lem. 5. We now show $1 \Rightarrow 3$: $\mathtt{t}$ solvable implies, by definition, the existence of a context $\mathtt{C}$ such that $\mathtt{C} = (\lambda\overline{\mathtt{x}}.\Box)\overline{\mathtt{v}}$ and either $\mathtt{C}[\mathtt{t}] \to^* \mathtt{I}$ or $\mathtt{C}[\mathtt{t}] \to^* \langle \mathtt{u}_1, \mathtt{u}_2 \rangle$, for some $\mathtt{u}_1, \mathtt{u}_2$. Both $\mathtt{I}$ and $\langle \mathtt{u}_1, \mathtt{u}_2 \rangle$ can be typed, so $\mathtt{C}[\mathtt{t}]$ is typed, by Lem. 4.2. Note that $\mathtt{C}[\mathtt{t}] = (\lambda\overline{\mathtt{x}}.\mathtt{t})\overline{\mathtt{v}}$, so for typing $\mathtt{C}[\mathtt{t}]$ we need to type $\mathtt{t}$.

5 Inhabitation for System $\mathcal{P}$

We extend approximate normal forms (*cf.* Sec. 3) as follows:

$$\mathtt{a}, \mathtt{b}, \mathtt{c} \ ::= \ \Omega \mid \mathcal{N} \qquad \mathcal{N} \ ::= \ \lambda\mathtt{x}.\mathcal{N} \mid \langle \mathtt{a}, \mathtt{a} \rangle \mid \mathcal{L} \qquad \mathcal{L} \ ::= \ \mathtt{x} \mid \mathcal{L}\mathtt{a} \mid \pi_i \mathcal{L}$$

The order relation $\leq$ and the sets $\mathcal{A}(\mathtt{t})$ are defined as in the case of the pure λ-calculus. The type assignment system $\mathcal{P}$ for the Λ_π-calculus is extended to approximate normal forms, assuming as before that no type can be assigned to the constant Ω. Given $\Pi \rhd \Gamma \vdash \mathtt{t} : \tau$, where $\mathtt{t}$ is in $\mathtt{hnf}$, we extend the definition of minimal approximant of $\mathtt{t}$.

Definition 7. *Given* $\Pi \rhd \Gamma \vdash \mathtt{t} : \sigma$*, where* $\mathtt{t}$ *is in* Π*-normal form,* $\mathcal{A}(\Pi) \in \mathcal{A}(\mathtt{t})$ *is defined by extending Def. 3 with the the following additional cases:*

- *If* $\Pi \rhd \ \vdash \langle \mathtt{t}, \mathtt{u} \rangle : \omega_\times$*, then* $\mathcal{A}(\Pi) = \langle \Omega, \Omega \rangle$.
- *If* $\Pi \rhd \Gamma \vdash \langle \mathtt{t}_1, \mathtt{t}_2 \rangle : \times_i(\tau)$ *follows from* $\Pi_i \rhd \Gamma \vdash \mathtt{t}_i : \tau$*, then* $i = 1$ *implies* $\mathcal{A}(\Pi) = \langle \mathcal{A}(\Pi_1), \Omega \rangle$ *and* $i = 2$ *implies* $\mathcal{A}(\Pi) = \langle \Omega, \mathcal{A}(\Pi_2) \rangle$.
- *If* $\Pi \rhd \Gamma \vdash \pi_i \mathtt{t} : \tau$ *follows from* $\Pi' \rhd \Gamma \vdash \mathtt{t} : \times_i(\tau)$*, then* $\mathcal{A}(\Pi) = \pi_i \mathcal{A}(\Pi')$.

5.1 The Inhabitation Algorithm

The algorithm in Fig. 4 is extended with the additional rules in Fig 6.

Example 2. Let $\Gamma = \emptyset$ and $\tau = \times_1([\alpha] \to \alpha)$ and $\sigma = [\tau] \to \tau$. Then given the input (Γ, σ), the algorithm can have the following two behaviours:

Choosing the sequence of rules: $(\mathtt{Abs}), (\mathtt{Prod1}), (\mathtt{Abs}), (\mathtt{Head}), (\mathtt{Proj}), (\mathtt{Prefix}), (\mathtt{Final})$ the final approximant is $\lambda\mathtt{y}.\langle \lambda\mathtt{x}.\pi_1 \mathtt{y}\mathtt{x}, \Omega \rangle$;

Choosing the sequence of rules: $(\mathtt{Abs}), (\mathtt{Head}), (\mathtt{Final})$ the final approximant is $\lambda\mathtt{y}.\mathtt{y}$;

$$\frac{\mathtt{a} \in \mathtt{H}^{\Delta}_{\pi_i(\mathtt{b})}(\Gamma, \sigma) \triangleright \tau}{\mathtt{a} \in \mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma, \times_i(\sigma)) \triangleright \tau}\ (\mathtt{Proj}) \qquad \frac{}{\langle \Omega, \Omega \rangle \in \mathtt{T}(\emptyset, \omega_\times)}\ (\mathtt{Pair})$$

$$\frac{\mathtt{a} \in \mathtt{T}(\Gamma, \tau)}{\langle \mathtt{a}, \Omega \rangle \in \mathtt{T}(\Gamma, \times_1(\tau))}\ (\mathtt{Prod1}) \qquad \frac{\mathtt{a} \in \mathtt{T}(\Gamma, \tau)}{\langle \Omega, \mathtt{a} \rangle \in \mathtt{T}(\Gamma, \times_2(\tau))}\ (\mathtt{Prod2})$$

Fig. 6. The inhabitation algorithm for the λ-calculus with products

To show that the inhabitation algorithm terminates, we extend the measure given in Sec. 2 by adding $\#(\omega_\times) = 1$ and $\#(\times_i(\tau)) = \#(\tau) + 1$. Termination and soundness hold, and the extension is straightforward.

Lemma 6 (Termination). *The inhabitation algorithm terminates.*

Proof. As for Lem. 1, using the extended measure above.

Lemma 7. $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma) \Leftrightarrow \exists \Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ *such that* $\mathtt{a} = \mathcal{A}(\Pi)$.

Proof. We follow the proof of Lem. 2, with the suitable additional cases:

($\Rightarrow$): Let $\mathtt{a} \in \mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma, \times_i(\sigma)) \triangleright \tau$ follow from $\mathtt{a} \in \mathtt{H}^{\Delta}_{\pi_i(\mathtt{b})}(\Gamma, \sigma) \triangleright \tau$ by ($\mathtt{Proj}$). Suppose $\Delta \vdash \mathtt{b} : \times_i(\sigma)$ $(i = 1, 2)$. Then $\Delta \vdash \pi_i(\mathtt{b}) : \sigma$. By the *i.h.* (c) we get $\Pi \triangleright \Gamma + \Delta \vdash \mathtt{a} : \tau$, where $\mathtt{a} = \mathcal{A}(\Pi)$ and we are done.

If $\langle \Omega, \Omega \rangle \in \mathtt{T}(\emptyset, \omega_\times)$ follows by ($\mathtt{Pair}$); then $\Pi \triangleright \vdash \langle \Omega, \Omega \rangle : \omega_\times$, and $\langle \Omega, \Omega \rangle = \mathcal{A}(\Pi)$.

If $\langle \mathtt{a}, \Omega \rangle \in \mathtt{T}(\Gamma, \times_1(\tau))$ follows from $\mathtt{a} \in \mathtt{T}(\Gamma, \tau)$ by ($\mathtt{Prod1}$), then by the *i.h.* (a) $\Gamma \vdash \mathtt{a} : \tau$. Then by ($\mathtt{pair1}$), $\exists \Pi \triangleright \Gamma \vdash \langle \mathtt{a}, \Omega \rangle : \times_1(\tau)$, and we are done, since $\langle \mathtt{a}, \Omega \rangle = \mathcal{A}(\Pi)$. Analougsly for ($\mathtt{Prod2}$).

($\Leftarrow$): 1. Let $\mathtt{a} = \pi_i \mathtt{a}'$ $(i = 1, 2)$. By construction Π ends by an application of the rule ($\mathtt{proji}$) with premise $\Pi' \triangleright \Gamma \vdash \mathtt{a}' : \times_i(\sigma)$, where by definition $\mathtt{a}' = \mathcal{A}(\Pi')$. Moreover, $\Sigma \triangleright \Delta \vdash \mathtt{b} : \tau$ is also a left-subtree of Π' and $\mathcal{A}(\Pi) = \pi_i \mathtt{a}'$ by definition, thus $\mathtt{H}^{\Delta + \Gamma'}_{\pi_i \mathtt{a}'}(\Theta, \sigma) \triangleright \pi \subseteq_{(\mathtt{Proj})} \mathtt{H}^{\Delta + \Gamma'}_{\mathtt{a}'}(\Theta, \times_i(\sigma)) \triangleright \pi \subseteq_{i.h.(1)} \mathtt{H}^{\Delta}_{\mathtt{b}}(\Theta + \Gamma', \tau) \triangleright \pi$.

2. If $\mathtt{a}$ is a pair then there are three cases to consider. If $\Pi \triangleright \vdash \mathtt{a} : \omega_\times$, then $\mathtt{a} = \langle \Omega, \Omega \rangle \in_{(\mathtt{Pair})} \mathtt{T}(\emptyset, \omega_\times)$. If $\Pi \triangleright \Gamma \vdash \mathtt{a} : \times_1(\tau)$ follows from $\Pi' \triangleright \Gamma \vdash \mathtt{a}' : \tau$ by ($\mathtt{proj1}$), then $\mathtt{a} = \langle \mathtt{a}', \Omega \rangle$. We have $\mathtt{a}' \in_{i.h.(2)} \mathtt{T}(\Gamma, \tau)$, so that $\mathtt{a} \in_{(\mathtt{Prod1})} \mathtt{T}(\Gamma, \times_1(\tau))$. The case $\Pi \triangleright \Gamma \vdash \mathtt{a} : \times_2(\tau)$ is similar.

Theorem 4 (Soundness and Completeness)

1. *If* $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma)$ *then, for all* $\mathtt{t}$ *such that* $\mathtt{a} \leq \mathtt{t}$, $\Gamma \vdash \mathtt{t} : \sigma$.
2. *If* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$ *then there exists* $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ *such that* $\mathtt{t}'$ *is in* Π'*-normal form, and* $\mathcal{A}(\Pi') \in \mathtt{T}(\Gamma, \sigma)$.

Proof. As in the proof of Thm. 2, but using Lem. 7 ($\Rightarrow$) for soundness, Thm. 4.1 and Lem. 7 ($\Leftarrow$) for completeness.

6 Conclusion

We proved that the inhabitation problem is decidable for the types systems $\mathcal{M}$ and $\mathcal{P}$, based on non-idempotent intersection types, and characterizing solvability for the λ-calculus and for its extension with pairs and projections, that we call Λ_π, respectively. To the best of our knowledge, solvability in Λ_π had not been studied before. Our result is a first step towards the study of further extensions of Λ_π, including *patterns* [11].

In fact, the logical characterization of solvability is related to the inhabitation problem of the underlying type system. While this relation is implicit for the λ-calculus, it can become crucial for extensions lacking a syntactical characterization of solvability.

Concerning denotational models, it is well known that system $\mathcal{M}$ induces a relational model of Λ. We aim to complete the picture studying the semantics of Λ_π through the system $\mathcal{P}$.

References

1. Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics, revised edn. North-Holland, Amsterdam (1984)
2. Ben-Yelles, C.: Type-assignment in the lambda-calculus; syntax and semantics. PhD thesis, University of Wales Swansea (1979)
3. Bernadet, A., Lengrand, S.: Non-idempotent intersection types and strong normalisation. Logical Methods in Computer Science 9(4) (2013)
4. Bunder, M.W.: The inhabitation problem for intersection types. In: CATS. CRPIT, vol. 77, pp. 7–14. Australian Computer Society (2008)
5. Carvalho, D.D.: Sèmantique de la logique linéaire et temps de calcul. PhD thesis, Université de la Mediterranée Aix-Marseille 2 (2007)
6. Coppo, M., Dezani-Ciancaglini, M.: An extension of the basic functionality theory for the λ-calculus. Notre Dame J. Form. Log. 21(4), 685–693 (1980)
7. De Benedetti, E., Ronchi Della Rocca, S.: Bounding normalization time through intersection types. In: Proc. of ITRS 2012. EPTCS, pp. 48–57 (2013)
8. Di Gianantonio, P., Honsell, F., Lenisa, M.: A type assignment system for game semantics. Theor. Comput. Sci. 398, 150–169 (2008)
9. Ehrhard, T.: The Scott model of linear logic is the extensional collapse of its relational model. Theor. Comput. Sci. 424, 20–45 (2012)
10. Hindley, J.R.: Basic Simple Type Theory. Cambridge Tracts in Theoretical Comp. Scie. Cambridge University Press, Amsterdam (2008)
11. Jay, C.B., Kesner, D.: First-class patterns. Journal of Functional Programming 19(2), 191–225 (2009)
12. Kesner, D., Ventura, D.: Quantitative types for the linear substitution calculus. In: Diaz, J., Lanese, I., Sangiorgi, D. (eds.) TCS 2014. LNCS, vol. 8705, pp. 296–310. Springer, Heidelberg (2014)
13. Kfoury, A.J.: A Linearization of the Lambda-Calculus and Consequences. J. Logic Comp. 10(3), 411–436 (2000)
14. Kfoury, A.J., Wells, J.B.: Principality and type inference for intersection types using expansion variables. Theor. Comput. Sci. 311(1-3), 1–70 (2004)

15. Krivine, J.L.: Lambda-Calculus, Types and Models. Ellis Horwood, Hemel Hempstead, Masson, Paris (1993)
16. Kurata, T., Takahashi, M.: Decidable properties of intersection type systems. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902, pp. 297–311. Springer, Heidelberg (1995)
17. Mairson, H., Neergaard, P.M.: Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In: Proc. of ICFP 2004, pp. 138–149 (2004)
18. Pagani, M., Ronchi Della Rocca, S.: Linearity, non-determinism and solvability. Fundamenta Informaticae 103, 358–373 (2010)
19. Pagani, M., della Rocca, S.R.: Solvability in resource lambda-calculus. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 358–373. Springer, Heidelberg (2010)
20. Paolini, L., Piccolo, M., Ronchi Della Rocca, S.: Logical relational lambda-models. Accepted for publication in MSCS (2013)
21. Urzyczyn, P.: The emptiness problem for intersection types. Journal of Symbolic Logic 64(3), 1195–1215 (1999)
22. Urzyczyn, P.: personal communication (2014)
23. van Bakel, S.: Complete restrictions of the intersection type discipline. Theor. Comput. Sci. 102, 135–163 (1992)

Author Index