LNCS 7604

Jos C. M. Baeten
Tom Ball
Frank S. de Boer (Eds.)

Theoretical Computer Science

7th IFIP TC 1/WG 2.2 International Conference, TCS 2012
Amsterdam, The Netherlands, September 2012
Proceedings

ifip

Springer

Jos C. M. Baeten Tom Ball
Frank S. de Boer (Eds.)

Theoretical Computer Science

7th IFIP TC 1/WG 2.2 International Conference, TCS 2012
Amsterdam, The Netherlands, September 26-28, 2012
Proceedings

Springer

Volume Editors

Jos C. M. Baeten
Centrum Wiskunde & Informatica (CWI)
Science Park 123, 1098 XG Amsterdam, The Netherlands
E-mail: jos.baeten@cwi.nl

Tom Ball
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
E-mail: tball@microsoft.com

Frank S. de Boer
Centrum Wiskunde & Informatica (CWI)
Science Park 123, 1098 XG Amsterdam, The Netherlands
E-mail: f.s.de.boer@cwi.nl

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-33474-0 e-ISBN 978-3-642-33475-7
DOI 10.1007/978-3-642-33475-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012946731

CR Subject Classification (1998): F.1.1-2, F.4.3, F.2.2, F.4.1, G.2.2

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

The conference TCS 2012, the 7th IFIP International Conference on Theoretical Computer Science, was organised by IFIP Technical Committee 1 (Foundations of Computer Science) and its 9 working groups, and IFIP Working Group 2.2 (Formal Descriptions of Programming Concepts), and was associated to the IFIP World Computing Congress, also held in Amsterdam in the same week. The TCS conference provides a meeting place for the theoretical computer science community where the latest results in computation theory can be presented and more broadly experts in theoretical computer science can get together to share insights and ask questions about the future directions of the field. TCS 2012 was associated with The Alan Turing Year 2012. Previous conferences of this series were held in Sendai (2000), Montreal (2002), Toulouse (2004), Santiago (2006), Milan (2008), and Brisbane (2010).

This volume contains the papers presented at the TCS conference held on September 26–28, 2012, hosted by the Centrum Wiskunde & Informatica in Amsterdam. There were 48 submissions. Each submission was reviewed by 3 program committee members (exceptionally, by 4 or 2). The committee decided to accept 25 papers. The conference program also included 3 invited talks by Rajeev Alur, Yuri Gurevich, and Jiri Wiedermann.

TCS 2012 was sponsored by the International Federation for Information Processing (IFIP), the Netherlands Organisation for Scientific Research (NWO), Microsoft Research, the Institute for Programming research and Algorithmics (IPA), and Centrum Wiskunde & Informatica (CWI).

We thank the members of the program committee and the additional reviewers for their work, the invited speakers for their contributions, and all authors who submitted their work to TCS 2012.

July 2012

Jos C.M. Baeten
Tom Ball
Frank S. de Boer

Organization

Program Committee

Jos C.M. Baeten	Centrum Wiskunde & Informatica
Tom Ball	Microsoft Research
Ahmed Bouajjani	LIAFA, University of Paris 7 (Paris Diderot)
Ana Cavalcanti	University of York
Frank S. De Boer	Centrum Wiskunde & Informatica
Susanne Graf	Université Joseph Fourier / CNRS / VERIMAG
Peter Gruenwald	Centrum Wiskunde & Informatica
Juraj Hromkovic	ETH Zurich
Jan Jürjens	TU Dortmund and Fraunhofer ISST
Joseph Kiniry	IT University of Copenhagen
Martin Kutrib	Universität Giessen
Aart Middeldorp	University of Innsbruck
Ugo Montanari	Università di Pisa
Peter Müller	ETH Zürich
David Naumann	Stevens Institute of Technology
Catuscia Palamidessi	INRIA and LIX, Ecole Polytechnique
Jan Rutten	Centrum Wiskunde & Informatica
Davide Sangiorgi	University of Bologna
Jeffrey Shallit	University of Waterloo
Leen Torenvliet	University of Amsterdam
Igor Walukiewicz	CNRS, LaBRI
Jim Woodcock	University of York

Additional Reviewers

Andova, Suzana
Atig, Mohamed Faouzi
Bergfeld, Jort
Bodlaender, Hans
Boeckenhauer, Hans-Joachim
Bonchi, Filippo
Bonsangue, Marcello
Bors, Adrian
Brengos, Tomasz
Bruni, Roberto
Butterfield, Andrew
Caires, Luis
Cairns, Paul
Cimini, Matteo
Courcelle, Bruno
Dal Lago, Ugo
De Liguoro, Ugo
de Wolf, Ronald
Enea, Constantin
Fantechi, Alessandro
Felgenhauer, Bertram
Fenner, Stephen
Fijalkow, Nathanaël
Fiore, Marcelo

Fokkink, Wan	Martin, Barnaby
Foniok, Jan	Meckel, Katja
Foster, Simon	Miller, Dale
Freitas, Leo	Montanaro, Ashley
Gebauer, Heidi	Mytkowicz, Todd
Geuvers, Herman	Neurauter, Friedrich
Gouw, Stijn	Pirandola, Stefano
Hansen, Helle	Rispal, Chloé
Herbreteau, Frédéric	Rossmanith, Peter
Hirschkoff, Daniel	Rot, Jurriaan
Holzer, Markus	Rutten, Jan
Horbach, Matthias	Salvati, Sylvain
Jaghoori, Mohammad Mahdi	Sammartino, Matteo
Jakobi, Sebastian	Sangnier, Arnaud
Jongmans, Sung-Shik T.Q.	Schöpp, Ulrich
Kameyama, Yukiyoshi	Silva, Alexandra
Keller, Lucia	Smula, Jasmin
Kishida, Kohei	Sobocinski, Pawel
Klin, Bartek	Steffen, Björn
Komm, Dennis	Sternagel, Christian
Kosowski, Adrian	Thiemann, René
Kratsch, Dieter	Truthe, Bianca
Krug, Sacha	Ummels, Michael
Loreti, Michele	van Leeuwen, Erik Jan
Lozes, Etienne	Van Raamsdonk, Femke
Luettgen, Gerald	Vicario, Enrico
Mackie, Ian	Winter, Joost
Malcher, Andreas	Zantema, Hans
Markovski, Jasen	Zhang, Lijun

Table of Contents

Computability and Non-computability Issues in Amorphous Computing*

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. Amorphous computing systems consist of a huge set of tiny simple stationary or mobile processors whose computational, communication and sensory part is reduced to an absolute minimum. In an airborne medium the processors communicate via a short-range radio while in a waterborne medium via molecular communication. In some cases the computational part of the processors can be simplified down to finite state automata or even combinatorial circuits and the system as a whole can still possess universal computational power with a high probability. We will argue that the amorphous systems belong among the simplest (non-uniform) universal computational devices. On the other hand, it is questionable as to what extent the standard universal models of computation can faithfully capture the behavior of amorphous computing systems whose functionality also depends on the non-computational and/or unpredictable operations of certain parts of the entire system.

1 Introduction

The notion of amorphous computing systems, i.e., of computational systems lacking any concrete "architecture", has emerged by the end of the 1990's. Initially, the development of such systems started as an engineering enterprise motivated by technological advancement in the field of micro-electro-mechanical systems, wireless communications and digital electronics (cf. [1], [2], [4], [5], [6], [7], [12], [13], [14]). Technological progress enabled integration of sensing, data processing and wireless communication capabilities into a single processor. In these systems the miniaturization has been pushed to its limits resulting, presumably, into processors of almost molecular size with the respective communication and computing facilities adequately (and thus, severely) restricted. These limitations alone, and the fact that systems consisting of huge numbers of processors are considered, have jointly called for the change of the basic computational and communication paradigms of distributed computing systems. These new paradigms also seem to have a potential to challenge certain computability issues related to our understanding of computing.

Nowadays we see amorphous computing systems in many forms (cf. [19]). Amorphous computing systems typically consist of a huge set of tiny simple processors equipped with small memory, random number generator, simple wireless communication means, sensors and an energy source. The processors are randomly distributed

* This work was partially supported by RVO 67985807 and the GA ČR grant No. P202/10/1333.

over a closed area in which they can move, or are static. In the former case, the processors move by their own locomotion means, or by external forces, like Brownian motion, wind or stream. In an airborne medium the processors communicate via radio while in a waterborne medium via molecular communication. Moreover, in order to operate as envisaged some amorphous computing systems may exploit non-computable features, i.e., operations that cannot be realized computationally, such as self-replication or disintegration.

In this note we will focus our attention on specific instances of amorphous computing systems by which we will illustrate some remarkable aspects of amorphous computing systems.

First, we will be interested in their universality, i.e., in their ability to simulate arbitrary computations (of a Turing machine, say). Namely, for some amorphous computing systems this ability is by far not obvious due to the drastic restrictions imposed on computational and communication parts of the system's processors.

Strictly speaking, many (envisaged) applications of amorphous systems do not require universality. Single-purpose systems enabling, e.g., centralized data collection, a geographic area monitoring, intra-body multi-modal health monitoring, or drug delivery to certain body locations, often do. Nevertheless, universality qualifies these systems among the programmable systems which are, in principle, capable to perform an arbitrary algorithmic or even robotic task.

The second intriguing feature which we will be interested in is the problem of the reverse simulation of amorphous computing systems by standard models of universal computations. Namely, by their very nature, some amorphous computing systems are in fact described as physical (rather than purely computational) systems whose operation also depends on abilities of their processors and those of their environment that are of non-computational nature. This appears to be a serious obstacle for their faithful simulation on standard computational devices.

In the sequel, in Section 2 we will briefly describe two relatively advanced and unusual amorphous computing systems: an airborne, so-called flying amorphous computing system, and a waterborne system, so-called nanomachines. Here we will only provide a high-level description of the respective machines and of their computational and non-computational mechanisms in order to give the reader the main ideas of their functioning. In a more detail these systems have been introduced in earlier writings of the present author (cf. [11], [18]). Based on the previous descriptions, we will discuss the related universality issues in Section 3. In Section 4 we will concentrate on the problem of a reverse simulation of the previously described amorphous systems on a universal computational model. Conclusions are in Section 5.

2 Universality in Amorphous Computing Systems

2.1 Flying Amorphous Computer

A flying amorphous computer consists of a set of asynchronous processors. Each processor posses a clock running with the same speed as all the other processors; however, the "ticking" of all clocks is not synchronized. Each processor is modeled by a "miniature" RAM with a finite number of registers capable of storing integers up to size N, with N denoting the number of nodes of the underlying amorphous system. Each processor is equipped by a random number generator and a single-channel radio device of a limited communication range. Initially, the processors have no unique identifier.

A severe restriction is imposed on the communication abilities of processors. A processor P_1 could receive a message sent by processor P_2 if and only if the following conditions hold true: (i) the processors are in the communication range of each other, (ii) P_1 is in a listening mode, and (iii) P_2 is in a broadcast mode, and it is the only processor within the communication radius of P_1 broadcasting at that time.

There is no mechanism making it possible to distinguish the case of no broadcast from that of broadcast collision. These restrictions concerning the radio communication are among the weakest ones that one can expect to be fulfilled by any simple radio communication device. The expected benefit from such restrictions is a simple engineering design of processors.

Note that the communication among the processors is complicated by the fact that the processors work asynchronously, have no identifiers, communication is one-way only, and there is no broadcast collision detection mechanism. As long as the processors remain anonymous (i.e., have no identifiers) a broadcasting processor has no means to learn that its message has been received by some processor. Under such circumstance, the randomized protocol designed in [10] enabling a reliable delivery of a message among processors within the communication range of each other works with a high probability, as follows.

The key idea is that the processors should broadcast a message sporadically in order to prevent message delivery (i.e., broadcast) conflicts, and repeatedly in order to maximize the likelihood of a successful delivery. The analysis of such a protocol reveals that the probability of sending should depend inversely on the expected number of a node's neighbors and should be repeated more times to handle the case of more processors in a node's neighborhood (cf. [17]).

Now, let us assume that cN of such processors, $c > 1$, fly around randomly in a confined convex volume. They form a dynamic network with a variable topology. The nodes of the network are created by processors with wireless communication links emerging asynchronously among the processors that find themselves within the communication radius of each other and fulfill the restriction for a successful one-way communication mentioned before. Our goal is to program the processors in such a way that they all together can simulate a RAM with N registers. Doing so, each RAM register will be represented in one processor of the flying computer.

The main problem is to keep the system operating under steadily changing topology of the communication network where new communication paths emerge, while the previous ones vanish. Some nodes may even become temporarily inaccessible since they may not find themselves within the communication range of other nodes. The latter problem can be solved under the assumption that no node in the network remains for ever isolated.

Thanks to this assumption, once processors do possess unique addresses, a message sent to a node with a given address would in a finite time reach this node and this node could send an acknowledgment that in a finite time will reach the sender.

The schema of the simulation is as follows. There is one specific node, a so-called leader. First, the leader invites all nodes to generate a random number within the range $[1..cN]$. Such an invitation is realized by “flooding” the net by an appropriate signal reaching all nodes with a high probability using the previously described protocol. Doing so, we cannot make use of the acknowledgments (since addresses are not yet available) and therefore a sufficient time must be allowed for the signal to spread over the entire volume with a high probability. Once the addresses are generated, the acknowledgment mechanism is used in all subsequent computations. Next, the duplicates are eliminated by a randomized algorithm described in [11] and the addresses are transformed into the range $[1..N]$. Now the simulation itself can start. It is a relatively straightforward procedure in which the next step is initiated by the leader only after the sender (i.e., the leader) obtains an acknowledgment from the receiver.

Although the whole system can correctly simulate a RAM (with a bounded memory size) with arbitrary high probability, the simulation time cannot be bounded by any function. However, if the address assignment process is successful (and this can be guaranteed with an arbitrary large probability), the simulation terminates within a finite time and always delivers the correct result. This computer has been described in full detail in [9] and later it was presented in [11].

2.2 Nanomachines

Recent unmatched improvements in nanotechnologies have enabled serious consideration of nano-scale machines whose size is of order 10^{-6} mm. Their prospective fabrication will make use of molecular self-assembly or of modifications of real bacteria via genetical engineering. To get an idea about the dimensions of objects we are considering, the size of a real bacteria is of the order of a few micrometers (i.e., of thousandths of millimeter, 10^{-6} m) while the size of a molecule is of the order of nanometers (i.e., 10^{-9} m). Thus, a nanomachine is about 1000 times bigger than a molecule and its surface and volume is still larger by a few orders of magnitude.

Next we will briefly describe so-called *self-reproducing mobile embodied automata* (nanomachines for short) whose information exchange mechanism is based on molecular communication.

Each nanomachine consists of two main components: there is its *embodiment* — the body, and its *computational part*.

The embodiment of any nanomachine consists of a set of receptors and emitters (pores), internal sensors, a set of timers, a self-reproducing mechanism, random bit generator and possibly of other devices depending on the type of embodiment (e.g., locomotive organs in the form of flagella, cilia, etc.).

Each receptor is specialized for detection of a specific type of molecules. These molecules find themselves in the environment surrounding the machine. Both the machines and the molecules move by convection (diffusion and advection). Moreover, the nanomachines can also move by their own means. For each type of molecules each nanomachine has at its disposal several tens of receptors; their exact number is irrelevant. A molecule gets recognized only in the case when it enters into contact with the respective receptor.

Timers are internal mechanisms ("organs") without any external input. Each timer is preset for a fixed time. Each timer returns either 0 or 1. A timer can be reset to 0 by the machine's finite state control. Upon expiration of time for which the timer has been initially set the timer returns 1. Values to which the timers are preset depend on the type of a timer as well as on the properties of the environment (especially on its volume, but also on the properties of some molecules detected by the sensors — e.g., on the degradation time of the molecules). Timers of the same type are the same in all nanomachines.

The self-reproducing mechanism is a non-computational mechanism which is triggered by automaton entering a special reproducing state. In such a case, the nanomachine splits into two identical copies of itself, with their finite controls entering the initial state.

The random bit generator is an "organ" that upon each activation returns either 0 or 1 with the same probability.

The computational part of each nanomachine is created by a finite-state (Mealy) automaton whose actions are controlled by a transition function. In a single move each automaton reads its inputs obtained from its receptors and from other sensors or organs. Depending on these inputs and on its current state, the automaton issues instructions for the machine's organs concerning their next actions: releasing the molecules from the receptors, secreting signal molecules via the pores (output ports), resetting the timers, and instructing its locomotive organs. Last but not least, the control enters a new (possibly a reproduction) state. The use of timers and of a random number generator effectively turns the automata at hand into timed probabilistic automata.

Thus, the instructions for the machines are transmitted via elements from a finite set of molecular signals. Prior to sending a new signal, the environment must be cleared of the previous signal molecules. This is done by endowing the molecules with a certain self-destruction ability — after a certain time they spontaneously disintegrate into components that are not interpreted as any signals. These components are continuously absorbed by nanomachines and re-cycled inside their bodies in order to produce other molecular structures.

During their operation the self-reproducing mobile nanomachines communicate via so-called *quorum sensing,* i.e., by making collective decisions based on the density of nanomachine population. This density is inferred from the concentration of signal molecules emitted by the machines within a confined space.

In a given volume the machines multiply and emit the signal molecules until their maximal concentration has been reached. Then, they make a collective decision in which they simulate one step of a counter automaton. The resulting amorphous system was shown to be able to model a counter automaton [18], [15]. Thus, sequences of nanomachine populations of growing size obey a universal computing power.

Except of the timers and an organ serving as a random bit generator a further modification of the embodiment of the underlying automata may include a memory organ. Then, the task of memorizing the current state can be delegated to that organ. Consequently, the computational mechanism of each nanomachine could be simplified down to combinatorial circuits of bounded depths (circuits from the complexity class AC_0).

The last mentioned model is of interest not only from a practical point of view, since it could lead to a simpler engineering of nanomachines, but also from the viewpoint of the theory of universal computing machines, as we will see in the sequel.

3 What Is the Simplest Universal Computational Model?

For many computer scientists, a Turing machine, or a counter machine (also known as Minsky machine [8]) is considered as the simplest computational model possessing a universal computing power. In fact, there exists a plethora of universal computational devices and it is a matter of taste to select the simplest one from among those models. As a rule, they are represented by the devices with a fixed rigid architecture that enables them to enter configurations from a potentially infinite set of configurations: they are, in fact, infinite automata. None finite device (finite even in the physical meaning of this word) can subsequently enter an unbounded number of different configurations. Inevitably, the size of the device must grow with the number of reachable configurations. This is also the case of the cellular automata possessing a universal computing power. Note that for each input the corresponding cellular automaton is finite; however, the number of its elements grows with the input size.

The next (in)appreciable property of the known universal computational systems is their "non-homogeneity" — they cannot be disassembled into smaller, in some sense elementary identical computational parts that could be reassembled in an arbitrary manner so as to give rise to a universal computing system. A partial departure from this rule is given by cellular automata that almost satisfy our requirement of homogeneity. Clearly, a cellular automaton can be disassembled into individual finite automata. It is obvious that finite automata are simpler computational devices than, e.g., Turing machines — they only accept regular languages and, therefore, are not universal. Nevertheless, upon a re-assemblage into a cellular automaton one must respect the original topology of the underlying network although the individual automata need not be returned to their original positions — they can be interchanged at wish. Then a universal computing power "emerges" again. Note that in the latter case the universal computing power will not emerge from among any (multi)set of arbitrarily connected finite automata — the condition for that to happen is that the the communication links among the automata must follow a certain regular pattern.

The last consideration brings us to the following reformulation of the question from the title of this section: does there exist a simple computational device whose multisets possess universal computational power even when there is no fixed topology of communication links among their individual elements?

The answer to this question had been prepared in the previous two subsections. Both flying amorphous computers and the nanomachines are the candidates for such devices. This answer must further be stated more precisely. First, in order to ensure that computations of arbitrary space complexity could be realized we must always speak of sequences (or populations) of growing size of such systems. (The corresponding devices are called non-uniform computational devices.) Second, we can only speak of simulations achieving their goal with a high probability.

If we had to make a choice between the two amorphous computing systems standing as candidates for the position of the simplest universal computing device the priority should probably be given to the population of nanomachines controlled by circuits. This is because their activity is governed by the simplest computing devices (viz. circuits). However, there is a price we have to pay for this simplicity. This is the fact that in addition to the purely computational part there is a non-computational mechanism comprising the "body" of the corresponding units of the system. In both cases, this body mainly consists of a communication mechanism, and in the case of nanomachines, the body also contains other, more complicated non-computational components, such as a self-reproducing mechanism, locomotive organs, etc. Moreover, in the case of nanomachines, a "collaboration" of the signal molecules — their disintegration in due time, was necessary. The operation of the system as a whole has been achieved by cooperation and orchestration of activities of all participating components controlled by computational parts of the units. From this point of view, cellular automata can bee seen as highly idealized models of amorphous systems considered in this paper in which it has been abstracted from the embodiment and communication mechanisms.

We conclude that amorphous computing systems considered above belong among the simplest (non-uniform) universal computing devices since their functionality is fully defined by the functionality of any of its parts, and there is no need to describe the "architecture" of the system as a whole.

4 The Problematic Simulation of Amorphous Systems by Turing Machines

When speaking about the universal computing power of amorphous computing systems in the previous sections we indicated how such systems can simulate devices which are already known to possess such a power. For the purpose of their reverse simulation, thanks to their embodiment and many non-computational features, amorphous computing systems should better be regarded as physical, rather than abstract mathematical systems. Thus, in this case, simulation should bridge the gap between a physical system and an abstract mathematical system. The crux of the problem is that behavior of the underlying physical system cannot be formally described down to the smallest detail. This is a source of difficulties when considering the reverse simulation of amorphous computing systems on universal models of computation.

What we have to do in this case is to duplicate, or imitate the functioning of amorphous computing systems on some formal model of computation in such a way that the behavior of the latter system closely mimics the observed behavior of the former system. Unfortunately, this appears to be practically impossible due to the unpredictable behavior of important elements of amorphous computing systems. For instance, in the case of flying amorphous computer, the trajectories of processors are continuous and random as well as the asynchronous communication activities of processors. All these activities run in parallel and concurrently. Moreover, e.g., in the case of nanomachines, the degree of parallelism increases with time (in the first phase of rising up the population of nanomachines). There are additional parallel processes running at each nanomachine (such as signal molecules sensing/emitting) and also those corresponding to the interactions of molecules in the environment. The disintegration processes of signal molecules must also be taken into account. It is difficult to imagine how emulation of such processes could run on a sequential computer (or any other computer with a bounded parallelism) keeping track of a potentially unbounded number of spatially and temporally related physical variables. Discretization of continuous processes underlying the operation of amorphous computing systems (e.g., think of the movement of the processors) within a discrete computational model, asynchronicity and a potentially unbounded parallelism may thus introduce unsurmountable timing problems for simulation of such processes.

These facts challenge the popular belief that any physical computational device can be emulated within any other, and especially, within any universal model of computation (cf. [3] and the references therein).

Amorphous computing systems also offer an interesting answer to the question ”what everything can compute?”. At the same time, they seem to present an example of computations that is captured neither by Turing machines nor by any of their parallelized or other known variants. Obviously, any answer to the question ”what is computation?” should also cover computations of amorphous computing systems.

5 Conclusions

Amorphous computing systems confront us with an interesting dichotomy. They harness non-computational mechanisms for the purpose of computing, but they also exploit computing in order to control these non-computational mechanisms. Thus, amorphous computing systems present a class of computing systems in which physical aspects, manifested through their embodiment, play an important role. Amorphous computing systems can be proved to be computationally universal with a high probability. The respective proofs can come through thanks to focusing to certain computational aspects of their functionality and postulating the expected outcomes of non-computational operations that also contribute to the mechanism of computation. Thus, when reasoning about computational universality of amorphous systems we reason, in fact, about a more abstract, simplified, often probabilistic model of an amorphous system.

The situation changes when we want to capture the behavior of a “real” amorphous computing system (as opposed to that of its model) on a universal computer. This might be the case when we want, e.g., to tune some physical parameters of an amorphous system in order to achieve its better practical performance. In such a case, we have to simulate the physical system as it is, i.e., inclusively of its non-computational aspects which are important for keeping the entire system performing its computational task. The non-computational aspects are determined by complex physical and in some cases, also chemical interactions among the basic elements of amorphous systems which cannot be described as computational processes.

An additional problem in simulating (or more precisely: in emulating) an amorphous computing system stems from the fact that such systems, in dependence on the input size, are capable to perform a potentially unbounded number of parallel operations in constant time. This cannot be done by any uniform physical model of a known parallel universal computer with a constant number of processors. Perhaps the self-reproduction mobile embodied automata present the first step towards a truly universal computational model.

References

1. Abelson, H., et al.: Amorphous Computing. MIT Artificial Intelligence Laboratory Memo No. 1665 (August 1999)
2. Abelson, H., Allen, D., Coore, D., Hanson, Ch., Homsy, G., Knight Jr., T.F., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R.: Amorphous Computing. Communications of the ACM 43(5), 74–82 (2000)
3. Nagy, N., Akl, S.G.: Time inderterminacy, non-universality in computation, and the demise of the Church-Turing thesis, School of Computing, Queen's University, Kingston, Ontario. Technical Report No. 2011-580, August 19, 27 p. (2011)
4. Arvind, D.K., Wong, K.J.: Speckled Computing - A Disruptive Technology for Network Information Appliances. In: Proc. IEEE International Symposium on Consumer Electronics (ISCE 2004), pp. 219–223 (2004)
5. Coore, D.: Introduction to Amorphous Computing. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 99–109. Springer, Heidelberg (2005)
6. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next century challenges: mobile networking for "Smart Dust". In: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom 1999, pp. 271–278. ACM (August 1999)
7. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Emerging Challenges: Mobile Networking for Smart Dust. Journal of Communications and Networks 2, 188–196 (2000)
8. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
9. Petrů, L.: Universality in Amorphous Computing. PhD Disseration Thesis. Dept. of Math. and Physics, Charles University, Prague (2009)
10. Petrů, L., Wiedermann, J.: A Model of an Amorphous Computer and Its Communication Protocol. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 446–455. Springer, Heidelberg (2007)
11. Petrů, L., Wiedermann, J.: A Universal Flying Amorphous Computer. In: Calude, C.S., Kari, J., Petre, I., Rozenberg, G. (eds.) UC 2011. LNCS, vol. 6714, pp. 189–200. Springer, Heidelberg (2011)
12. Sailor, M.J., Link, J.R.: Smart dust: nanostructured devices in a grain of sand. Chemical Communications 11, 1375 (2005)
13. Shah, S.C., Chandio, F.H., Park, M.: Speckled Computing: Evolution and Challenges. In: Proc. IEEE International Conference on Future Networks, pp. 181–185 (2009)
14. Warneke, B., Last, M., Liebowitz, B., Pister, K.S.J.: Smart Dust: communicating with a cubic-millimeter computer. Computer 34(1), 44–51 (2001)
15. Wiedermann, J., Petru, L.: Computability in Amorphous Structures. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 781–790. Springer, Heidelberg (2007)
16. Wiedermann, J., Petrů, L.: Communicating Mobile Nano-Machines and Their Computational Power. In: Cheng, M. (ed.) NanoNet 2008. LNICST, vol. 3, pp. 123–130. Springer, Heidelberg (2009)
17. Wiedermann, J., Petrů, L.: On the Universal Computing Power of Amorphous Computing Systems. Theory of Computing Systems 46(4), 995–1010 (2009), http://www.springerlink.com/content/k2x6266k78274m05/fulltext.pdf
18. Wiedermann, J.: Nanomachine Computing by Quorum Sensing. In: Kelemen, J., Kelemenová, A. (eds.) Păun Festschrift. LNCS, vol. 6610, pp. 203–215. Springer, Heidelberg (2011)
19. Wiedermann, J.: The Many Forms of Amorphous Computational Systems. In: Zenil, H. (ed.) A Computable Universe: Understanding and Exploring Nature As Computation. World Scientific Publishing Company (to appear, 2012)

Static Single Information Form for Abstract Compilation*

Davide Ancona and Giovanni Lagorio

DIBRIS, Università di Genova, Italy
{Davide.Ancona,Giovanni.Lagorio}@unige.it

Abstract. In previous work we have shown that more precise type analysis can be achieved by exploiting union types and static single assignment (SSA) intermediate representation (IR) of code.

In this paper we exploit static single information (SSI), an extension of SSA proposed in literature and adopted by some compilers, to allow assignments of more precise types to variables in conditional branches. In particular, SSI can be exploited rather easily and effectively to infer more precise types in dynamic object-oriented languages, where explicit runtime typechecking is frequently used.

We show how the use of SSI form can be smoothly integrated with abstract compilation, our approach to static type analysis. In particular, we define abstract compilation based on union and nominal types for a simple dynamic object-oriented language in SSI form with a runtime typechecking operator, to show how precise type inference can be.

1 Introduction

In previous work [6] we have shown that more precise type analysis can be achieved by exploiting union types and static single assignment (SSA) [8] intermediate representation (IR) of code. Most modern compilers (among others, GNU's GCC [15], the SUIF compiler system [14], Java HotSpot [12], and Java Jikes RVM [10]) and formal software development tools implement efficient algorithms for translating code in advanced forms of IR particularly suitable for static analysis, thus offering the concrete opportunity of exploiting such IRs to obtain more precise type analysis and inference, and to fruitfully reuse those software components devoted to IR generation.

Abstract compilation [5,4,6] is a modular approach to static type analysis aiming to reconcile types and symbolic execution: an expression e is well-typed if the goal generated by compiling e succeeds w.r.t. the coinductive[1] model of the constraint logic program obtained by compiling the source program in which the expression is executed. In such a model terms are types representing possibly infinite sets of values, and goal resolution corresponds to symbolic execution.

* This work has been partially supported by MIUR DISCO - Distribution, Interaction, Specification, Composition for Object Systems.

[1] Coinduction allows proper treatment of recursive types and methods [5].

Abstract compilation is particularly suited for implementing type inference and global type analysis of dynamic object-oriented languages in a modular way since one can provide several compilation schemes for the same language, each corresponding to a different kind of analysis, without changing the inference engine, which typically implements coinductive constraint logic programming [17,16,4]. For instance, in previous work we have defined compilation schemes based on union and structural object types, to support parametric and data polymorphism, (that is, polymorphic methods and fields) to obtain precise type analysis, and a smooth integration with the nominal type annotations contained in the programs and the inferred structural types [5]; other proposed compilation schemes aim to detect uncaught exceptions [4], or to integrate SSA IR in the presence of imperative features [6].

In this paper we exploit static single information (SSI), an extension of SSA proposed in literature [2,19], to allow more precise type inference in conditional branches guarded by runtime typechecks. SSI has been already adopted by compiler frameworks as LLVM [20], PyPy [3], and SUIF [18], and proved to be more effective than SSA for performing data flow analysis, program slicing, and interprocedural analysis. Similar IRs are adopted as well in formal software development tools.

We show how SSI can be exploited rather easily and effectively by abstract compilation to improve type inference of dynamic object-oriented languages, where explicit runtime typechecks are frequently used.

To this aim, we formally define the operational semantics of a simple dynamic object-oriented language in SSI form equipped with a runtime typechecking operator, and then provide an abstract compilation scheme based on union and nominal types supporting more precise type inference of branches guarded by explicit runtime typechecks.

The paper is structured as follows: Section 2 introduces SSA and SSI IRs and motivates their usefulness for type analysis; Section 3 formally defines the SSI IR of a dynamic object-oriented language equipped with an operator **instanceof** for runtime typechecking. Section 4 presents a compilation scheme for the defined IR, based on nominal and union types, and Section 5 concludes with some considerations on future work. Abstract compilation of the code examples in Section 2 together with the results of the resolution of some goals can be found in an extended version of this paper.[2]

2 Type Analysis with SSA and SSI

In this section SSA and SSI IRs are introduced and their usefulness for type analysis is motivated.

Type Analysis with Static Single Assignment Form
Method `read()` declared below, in a dynamic object-oriented language, creates and returns a shape which is read through method `nextLine()` that reads the

[2] Available at `ftp://ftp.disi.unige.it/person/AnconaD/tcs12long.pdf`

next available string from some input source. The partially omitted methods `readCircle()` and `readSquare()` read the needed data from the input, create, and return a new corresponding instance of `Circle` or `Square`.

```
class ShapeReader {
   ...
   nextLine() {...}
   readCircle() { ... return new Circle(...); }
   readSquare() { ... return new Square(...); }
   read() {
      st = this.nextLine();
      if(st.equals("circle")) {
         sh = this.readCircle();
         this.print("A circle with radius ");
         this.print(sh.getRadius());
      }
      else if(st.equals("square")) {
         sh = this.readSquare();
         this.print("A square with side ");
         this.print(sh.getSide());
      }
      else throw new IOException();
      this.print("Area = ");
      this.print(sh.area());
      return st;
   }
}
```

Although method `read()` is type safe, no type can be inferred for `sh` to correctly typecheck the method; indeed, when method `area()` is invoked, variable `sh` may hold an instance of `Circle` or `Square`, therefore the most precise type that can be correctly assigned to `sh` is `Circle`$\vee$`Square`. However, if `sh` has type `Circle`$\vee$`Square`, then both `sh.getRadius()` and `sh.getSide()` do not typecheck.

There are two different kinds of approaches to solve the problem shown above. One can either define a rather sophisticated flow-sensitive type system, where each occurrence of a single variable can be associated with a different type, or typecheck the SSA IR, in which the method can be compiled.

In an SSA IR the value of each variable is determined by exactly one assignment statement [8]. To obtain this property, a flow graph is built, and a suitable renaming of variables is performed to keep track of the possibly different versions of the same variable; following Singer's terminology [19] we call these versions *virtual registers*. Conventionally, this is achieved by using a different subscript for each virtual register corresponding to the same variable. For instance, in the SSA IR of method `read()` there are three virtual registers (`sh`$_0$, `sh`$_1$ and `sh`$_2$) for the variable `sh`.

```
read() {
  b1:{st0 = this.nextLine();
      if(st0.equals("circle"))
         jump b2;
      else
         jump b3;}
  b2:{sh0=this.readCircle();
  this.print("A circle with radius ");
      this.print(sh0.getRadius());
      jump b5;}
  b3:{if(st0.equals("square"))
         jump b4;
      else
         jump b6;}
```

```
    b4:{sh₁=this.readSquare();
    this.print("A square with side ");
        this.print(sh₁.getSide());
        jump b5;}
    b5:{sh₂=φ(sh₀,sh₁);
    this.print("Area = ");
        this.print(sh₂.area());
        jump out;}
    b6:{throw new IOException();}
    out:{return sh₂;}
}
```

To transform a program into SSA form, a pseudo-function, conventionally called φ-function, needs to be introduced to correctly deal with merge points. For instance, in block `b5` the value of `sh` can be that of either `sh`$_0$ or `sh`$_1$, therefore a new virtual register `sh`$_2$ has to be introduced to preserve the SSA property. The expression φ(`sh`$_0$,`sh`$_1$) simply keeps track of the fact that the value of `sh`$_2$ is determined either by `sh`$_0$ or `sh`$_1$.

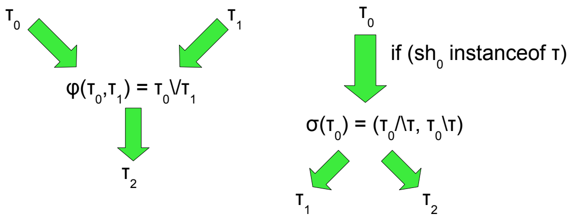


Fig. 1. Type theoretic interpretation of φ-function and σ-function

At the level of types, the φ-function naturally corresponds to the union type constructor (Figure 1): arrows correspond to data flow and, as usual, to ensure soundness the type at the origin of an arrow must be a subtype of the type the arrow points to. That is, for the types shown in the figure, $\tau_0, \tau_1 \leq \tau_0 \vee \tau_1 \leq \tau_2$.

Thanks to pretty standard and efficient algorithms for transforming source programs into SSA IR [8,9], the flow analysis phase, where source code is transformed into IR, can be kept separate from the subsequent type analysis phase, favoring simplicity and reuse. Indeed, flow analysis and consequent transformation into IR is implemented by most compilers and formal software development tools. Abstract compilation makes such an approach even more modular, by dividing the overall process into three separate stages. First, the source code is transformed into a suitable IR. Then, the IR is compiled into a set of Horn clauses and a goal. Finally, the goal is resolved with an appropriate inference engine (typically, implementing coinductive constraint logic programming [4]).

This paper mainly focuses on the second stage of the overall process (that is, compilation from IR into Horn clauses), in the particular case when the adopted IR allows precise flow analysis, as happens with SSI, of dynamic object-oriented languages.

Type Analysis with Static Single Information Form

Let us consider method `largerThan(sh)` of class `Square`, where **instanceof** is exploited to make the method more efficient in case the parameter `sh` contains an instance of (a subclass) of `Square`.

```
class Square {
...
   largerThan(sh) {
     if(sh instanceof Square)
        return this.side > sh.side;
     else
        return this.area() > sh.area();
   }
}
```

The method is transformed into the following SSA IR:

```
  largerThan(sh0) {
    b1:{if(sh0 instanceof Square)
           jump b2;
        else
           jump b3;}
    b2:{r0=this.side > sh0.side;
        jump out;}
    b3:{r1=this.area() > sh0.area();
        jump out;}
    out:{r2=φ(r0,r1);
        return r2;}
}
```

Since variable `sh` is not updated, both blocks `b2` and `b3` refer to the same virtual register $\texttt{sh}_0$. As a consequence, the only possible type that can be correctly associated with $\texttt{sh}_0$ is `Square`, thus making the method of little use. However, this problem can be addressed if one considers the SSI IR of the method [2,19].

```
  largerThan(sh0) {
    b1:{if(sh0 instanceof Square) with (sh1,sh2) = σ(sh0)
                                       (this1,this2) = σ(this0)
           jump b2;
        else
           jump b3;}
    b2:{r0=this1.side > sh1.side;
        jump out;}
    b3:{r1=this2.area() > sh2.area();
        jump out;}
    out:{r2=φ(r0,r1);
        return r2;}
}
```

SSI is an extension of SSA enforcing the additional constraint that all variables must have different virtual registers in the branches of conditional expressions. Such a property is obtained by a suitable renaming and by the insertion of a pseudo function, called σ-function. As a consequence, suitable virtual registers and a σ-function have to be introduced also for the read-only pseudo-variable **this**.

The notion of σ-function is the dual of φ-function (Figure 1); the type theoretic interpretation of σ depends on the specific kind of guard. If it is a runtime typecheck (of the form ($\texttt{sh}_0$ **instanceof** `Square`) as in the example), then σ splits the type τ_0 of $\texttt{sh}_0$ in the type $\tau_0 \wedge \texttt{Square}$, assigned to $\texttt{sh}_1$, and in the type $\tau_0 \backslash$`Square`, assigned to $\texttt{sh}_2$, where the intersection and the complement operators have to be properly defined (see Section 4). For instance, if $\texttt{sh}_0$ has type `Square`$\vee$`Circle`, then $\texttt{sh}_1$ has type `(Square`$\vee$`Circle)`$\wedge$`Square`$=$`Square`, and $\texttt{sh}_2$ has type `(Square`$\vee$`Circle)`$\backslash$`Square`$=$`Circle`, therefore `Square`$\vee$`Circle` turns out to be a valid type for the parameter $\texttt{sh}_0$ of the method `largerThan`.

For what concerns **this**, in this particular example no real split would be necessary: **this**$_0$ has type `Square`, so `Square` is split into `(Square,Square)`, that is, both **this**$_1$ and **this**$_2$ have the same type `Square`.

3 Language Definition

In this section we formally define an SSI IR for a simple dynamic object-oriented language equipped with an **instanceof** operator for performing runtime type-checking. Even though we have chosen a familiar Java-like syntax both for the IR and the source code used in the examples, the language is fully dynamic: code does not contain any type annotation, hence, under this point of view the language is quite different from Java.

$$
\begin{array}{rcl}
\mathit{prog} & ::= & \overline{cd}^n\ \{\overline{b}^n\} \\
\mathit{cd} & ::= & \texttt{class}\ c_1\ \texttt{extends}\ c_2\ \{\ \overline{f}^n\ \overline{md}^k\ \} \quad (c_1 \neq \mathit{Object}) \\
\mathit{md} & ::= & m(\overline{r}^n)\ \{\overline{b}^n\} \\
b & ::= & l{:}e \\
r & ::= & x_i \\
e & ::= & r \mid \texttt{new}\ c(\overline{e}^n) \mid e.f \mid e_0.m(\overline{e}^n) \mid e_1; e_2 \mid r = e \\
 & & \mid e_1.f = e_2 \mid \texttt{jump}\ l \mid r = \varphi(\overline{r}^n) \mid \texttt{return}\ r \\
 & & \mid \texttt{if}\ (r\ \texttt{instanceof}\ c)\ \texttt{with}\ \overline{(r', r'') = \sigma(r''')}^n\ \texttt{jump}\ l_1\ \texttt{else jump}\ l_2
\end{array}
$$

Syntactic assumptions: inheritance is not cyclic, method bodies are in correct SSI form and are terminated with a unique `return` statement, method and class names are disjoint, no name conflicts in class, field, method and parameter declarations, main expression and declared parameters cannot be `this`.

Fig. 2. SSI intermediate language

A program is a collection of class declarations followed by an anonymous main method with no parameters and contained in an anonymous class (conventionally its fully qualified name is $\epsilon.\epsilon$), whose body is a sequence of blocks (see the comments on method bodies below).

The notation $\overline{cd}^n$ is a shortcut for $cd_1, \ldots, cd_n$. A class declares its direct superclass (only single inheritance is supported), its fields, and its methods. *Object* is the usual predefined root class of the inheritance tree; every class comes equipped with the implicit constructor with parameters corresponding to all fields, in the same order as they are inherited and declared. For simplicity, no user constructors can be declared.

Method bodies are sequences of uniquely labeled blocks that contain sequences of expressions. We assume that all blocks contain exactly one jump, necessarily placed at the end of the block. Three different kinds of jumps are considered: local unconditional and conditional jumps, and returns from methods. Method bodies are implicitly assumed to be in correct SSI IR: each virtual register is determined by exactly one assignment statement, and all variables must have different virtual registers in the branches of conditional expressions. Finally, all method bodies contain exactly one return expression, which is always placed at the end of the body.[3]

Virtual registers have the form x_i, where x is the corresponding variable. If r is a virtual register, then $var(r)$ returns the variable the register refers to, therefore if $r = x_i$, then $var(r) = x$. The receiver object can be referred inside method bodies with the special implicit parameter `this`, hence the IR contains virtual registers of the form $\texttt{this}_i$.

Besides usual statements and expressions, we consider φ and σ pseudo-function assignments. Conditional jumps contain σ-functions which split each virtual register r occurring in either branches into two new distinct versions used in the blocks labeled by $\texttt{l}_1$, and $\texttt{l}_2$, respectively. The guard can only be of the form (r **`instanceof`** c); however, more elaborated guards can be easily expressed in terms of this primitive one by suitable transformations during the compilation from the source code to the IR. Depending on the types and abstract compilation scheme, there could be other kinds of guards for which SSI would improve type analysis; for instance, if one includes the type corresponding to the null references, then a guard of the form `(`r `==` **null**`)` would take advantage of SSI to enhance null reference analysis. For those guards for which no type refinement is possible the σ-function performs no split, that is, $\sigma(\tau)=(\tau,\tau)$.

Semantics. To define the small step semantics of the language we first need to specify values v (see Figure 3), which are just identities o of dynamically created objects. Furthermore, we add *frame expressions* $ec\{e\}$, where ec is an execution context; frame expressions are *runtime expressions* needed for defining the small step semantics of method calls. An execution context ec is a pair consisting of a stack frame fr and a fully qualified name μ. A frame expression $\langle fr, \mu\rangle\{e\}$ corresponds to the execution of a call to a method m declared in class c, where e is the residual expression (yet to be evaluated) of the currently executed block, fr is the stack frame of the method call, and $\mu = c.m$ is the fully qualified name of the method.

Stack frames fr map virtual registers to their corresponding values. Each association is labeled with a distinct time-stamp t, which specifies how recently the register has been updated (higher time-stamp values correspond to more recent updates). Such labels are used to define the semantics of φ-function assignments.

Heaps $\mathcal{H}$ map object identifiers o to objects, that is, pairs consisting of a class name c and the set of field names f with their corresponding value v.

[3] Such a constraint does not imply any loss of generality, since it is always possible to add new virtual registers and to insert a φ-function when the source code contains multiple returns.

$$
\begin{aligned}
v &::= o \quad \text{(values)}\\
e &::= v \mid ec\{e\} \mid \ldots \quad \text{(runtime expressions)}\\
ec &::= \langle fr, \mu\rangle \quad \text{(execution context)}\\
fr &::= \overline{r \mapsto_t v}^k \quad \text{(stack frames)}\\
\mu &::= c.m \quad \text{(full method names)}\\
\mathcal{H} &::= \overline{o \mapsto \langle c, \overline{f \mapsto v}^j \rangle}^k \quad \text{(heaps)}\\
\mathcal{C}[\cdot] &::= [\cdot] \mid ec\{\mathcal{C}[\cdot]\} \mid \texttt{new}\ c(\overline{v}^n, \mathcal{C}[\cdot], \overline{e}^j) \mid \mathcal{C}[\cdot].f \mid \mathcal{C}[\cdot].m(\overline{e}^k) \mid v_0.m(\overline{v}^j, \mathcal{C}[\cdot], \overline{e}^k)\\
&\quad \mid \mathcal{C}[\cdot]; e \mid x = \mathcal{C}[\cdot] \mid \mathcal{C}[\cdot].f = e \mid v.f = \mathcal{C}[\cdot]\\
&\quad \mid \texttt{if}\ (\mathcal{C}[\cdot])\ \texttt{with}\ \overline{(x', x'') = \sigma(x''')}^n\ \texttt{jump}\ l_1\ \texttt{else jump}\ l_2
\end{aligned}
$$

Fig. 3. Syntactic definitions instrumental to the operational semantics

Figure 4 shows the execution rules. Three different judgments are defined: the main judgment $\overline{cd}^n\ \{\overline{b}^n\} \Rightarrow v$ states that the main method $\{\overline{b}^n\}$ of program $\overline{cd}^n$ evaluates to value v. Such a judgment directly depends on the auxiliary judgment $\mathcal{H} \vdash e \rightarrow \mathcal{H}', e'$, stating that e rewrites to e' in $\mathcal{H}$, yielding the new heap $\mathcal{H}'$, and whose definition uses the auxiliary judgment $\mathcal{H}, ec \vdash e \rightarrow \mathcal{H}', ec', e'$, having the meaning that redex e rewrites to e' in $\mathcal{H}$ and ec, yielding the new execution context ec' and heap $\mathcal{H}'$. All the auxiliary judgments and functions[4] should be parametrized by the whole executing program, $\overline{cd}^n$, which, however, is kept implicit to favor readability.

Rule (main) defines the main judgment; a value v is returned if the runtime expression $\langle \epsilon_{fr}, \epsilon.\epsilon\rangle\{e\}$, where e is the first block (retrieved by the auxiliary function *firstBlock*) of the main method, transitively rewrites to v (and a heap $\mathcal{H}$ which is discarded). The evaluation context $\langle \epsilon_{fr}, \epsilon.\epsilon\rangle$ of the frame expression specifies that initially the frame (ϵ_{fr}) is empty (neither `this`, nor parameters are accessible), and that execution starts in the main method, whose fully qualified name is $\epsilon.\epsilon$ (recall that, conventionally, the main method is anonymous and is contained in an anonymous class).

The auxiliary judgment $\mathcal{H} \vdash e \rightarrow \mathcal{H}', e'$ is defined by the three rules (meth-call) (a new execution context is created), (ctx) (evaluation continues in the currently active execution context), and (return) (the current execution context is closed).

In rule (meth-call), the object referenced by o is retrieved from the heap to find its class, c. Then, the auxiliary functions *firstBlock* and *params* return the first block of the method and its parameters, respectively. The result of the evaluation is a frame expression, where the new stack frame maps parameters to their corresponding arguments, and $\texttt{this}_0$ to o, the fully qualified name $c.m$ corresponds to the invoked method, and the expression is the first block of the method. Rule (ctx) deals with context closure. Contexts (the standard definition is in Figure 3) correspond to a deterministic call-by-value and left-to-right evaluation strategy. A single computation step in the current execution context (corresponding to the most nested frame expression) is performed. The active execution context is extracted by *currentEC*; then, if the redex e rewrites to e'

[4] The straightforward definitions of the auxiliary functions have been omitted.

$$(\text{main})\,\frac{\mathit{firstBlock}(\epsilon.\epsilon) = e \qquad \epsilon_{\mathcal{H}} \vdash \langle \epsilon_{\mathit{fr}}, \epsilon.\epsilon\rangle\{e\} \to^{*} \mathcal{H}, v}{\overline{cd}^{n}\ \{\overline{b}^{n}\} \Rightarrow v}$$

$$(\text{meth-call})\,\frac{\begin{array}{l}\mathcal{H}(o) = \langle c, \_\rangle \\ \mathit{firstBlock}(c.m) = e \\ \mathit{params}(c.m) = \overline{r}^{n} \\ \mathit{fr} = \overline{r \mapsto v}^{n}, \mathtt{this}_0 \mapsto o\end{array}}{\begin{array}{l}\mathcal{H} \vdash \mathcal{C}[o.m(\overline{v}^{n})] \\ \quad \to \mathcal{H}, \mathcal{C}[\langle \mathit{fr}, c.m\rangle\{e\}]\end{array}} \qquad (\text{ctx})\,\frac{\begin{array}{l}\mathit{currentEC}(\mathcal{C}[\cdot]) = ec \\ \mathcal{H}, ec \vdash e \to \mathcal{H}', ec', e' \\ \mathcal{C}'[\cdot] = \mathit{updateEC}(\mathcal{C}[\cdot], ec')\end{array}}{\mathcal{H} \vdash \mathcal{C}[e] \to \mathcal{H}', \mathcal{C}'[e']}$$

$$(\text{return})\,\frac{}{\begin{array}{l}\mathcal{H} \vdash \mathcal{C}[\langle \mathit{fr}, \mu\rangle\{\mathtt{return}\ r\}] \\ \quad \to \mathcal{H}, \mathcal{C}[\mathit{fr}(r)]\end{array}} \qquad (\text{fld-acc})\,\frac{\mathcal{H}(o) = \langle c, \overline{f \mapsto v}^{n}\rangle \quad f = f_j}{\mathcal{H}, ec \vdash o.f \to \mathcal{H}, ec, v_j}$$

$$(\text{reg})\,\frac{}{\mathcal{H}, \langle \mathit{fr}, \mu\rangle \vdash r \to \mathcal{H}, \langle \mathit{fr}, \mu\rangle, \mathit{fr}(r)} \qquad (\text{new})\,\frac{\begin{array}{c}o \text{ fresh in } \mathcal{H} \\ \mathit{fieldNames}(c) = \overline{f}^{n}\end{array}}{\begin{array}{l}\mathcal{H}, ec \vdash \mathtt{new}\ c(\overline{v}^{n}) \\ \quad \to \mathcal{H}[\langle c, \overline{f \mapsto v}^{n}\rangle / o], ec, o\end{array}}$$

$$(\text{seq})\,\frac{}{\mathcal{H}, ec \vdash v; e \to \mathcal{H}, ec, e} \qquad (\text{reg-asn})\,\frac{}{\begin{array}{l}\mathcal{H}, \langle \mathit{fr}, \mu\rangle \vdash r = v \\ \quad \to \mathcal{H}, \langle \mathit{fr}[v/r], \mu\rangle, v\end{array}}$$

$$(\text{fld-asn})\,\frac{\begin{array}{l}\mathcal{H}(o) = \langle c, \overline{f \mapsto v}^{n}\rangle \\ f = f_j \quad \text{if } i = j \text{ then } v'_i = v \\ \qquad\qquad\qquad \text{else } v'_i = v_i\end{array}}{\begin{array}{l}\mathcal{H}, ec \vdash o.f = v \\ \quad \to \mathcal{H}[\langle c, \overline{f \mapsto v'}^{n}\rangle / o], ec, v\end{array}} \qquad (\text{jump})\,\frac{\mathit{block}(\mu, l) = e}{\begin{array}{l}\mathcal{H}, \langle \mathit{fr}, \mu\rangle \vdash \mathtt{jump}\ l \\ \quad \to \mathcal{H}, \langle \mathit{fr}, \mu\rangle, e\end{array}}$$

$$(\text{phi})\,\frac{}{\mathcal{H}, \langle \mathit{fr}, \mu\rangle \vdash r_0 = \varphi(\overline{r}^{n}) \to \mathcal{H}, \langle \mathit{fr}[\mathit{mru}(\mathit{fr}, \overline{r}^{n})/r_0], \mu\rangle, v}$$

$$(\text{if})\,\frac{\begin{array}{l}\mathcal{H}(\mathit{fr}(r)) = \langle c', \_\rangle \\ \text{if } c' \leq c \text{ then } \ l' = l_1, \mathit{fr}' = \mathit{fr}[\overline{\mathit{fr}(r''')/r'}^{n}] \\ \qquad\qquad \text{else } \ l' = l_2, \mathit{fr}' = \mathit{fr}[\overline{\mathit{fr}(r''')/r''}^{n}] \\ \mathit{block}(\mu, l') = e\end{array}}{\begin{array}{l}\mathcal{H}, \langle \mathit{fr}, \mu\rangle \vdash \mathtt{if}\ (r\ \mathtt{instanceof}\ c)\ \mathtt{with}\ \overline{(r', r'') = \sigma(r''')}^{n} \\ \qquad \mathtt{jump}\ l_1\ \mathtt{else}\ \mathtt{jump}\ l_2 \\ \qquad \to \mathcal{H}, \langle \mathit{fr}, \mu\rangle, e\end{array}}$$

Fig. 4. Small-step semantics

yielding $\mathcal{H}'$ and ec' (see the other rules defining the auxiliary evaluation judgment), then the $\mathcal{C}[e]$ rewrites to $\mathcal{C}'[e']$, yielding the new heap $\mathcal{H}'$; context $\mathcal{C}'[\ ]$ is obtained from $\mathcal{C}[\ ]$ by updating the frame expression corresponding to the active execution context with the new execution context ec'. In rule (return) the current execution context is closed, the heap is unaffected, and the result is the value associated with the returned virtual register r in the frame of the closing context.

The remaining rules define the auxiliary judgment $\mathcal{H}, ec \vdash e \rightarrow \mathcal{H}', ec', e'$, one for any distinct kind of redex. In rule (reg) a virtual register is accessed by extracting the corresponding value from the stack frame fr.Variable and field assignments evaluate to their right values; rule (reg-asn) has the side effect of updating, in the current stack frame fr, the value of the virtual register r and its associated time-stamp (this is implicit in the definition of $fr[v/r]$) since, after the assignment, r becomes the most recently updated register. Rule (fld-asn) deals with field assignments: the object referenced by o is retrieved from the heap, and its value updated. In rule (seq) the left-hand-side value in a sequence expression is discarded to allow evaluation to proceed with the next expression. In rule (phi), register r_0 is updated with the value (denoted by $mru(fr, \overline{r}^n)$) of the most recently updated register in the stack frame, between $\overline{r}^n$. In rule (new) a new object, identified by a fresh reference o, is added to the heap $\mathcal{H}$. The fields $\overline{f}^n$ of the newly created object are initialized by the values passed to the constructor. In rule (fld-acc) field accesses are evaluated: the object is retrieved from the heap, and the resulting expression is the value of the selected field. Rules (jump) and (if) deal with unconditional and conditional jumps, respectively. The evaluation of a jump returns the expression e contained in the block labeled by l' in the method μ of the current execution context. The conditional jump (rule (if)) selects which branch to execute and which virtual registers have to be updated, depending on whether the value $fr(r)$ of the register r is a reference to an object of a subclass of c. If it is the case, then the returned expression is that labeled by l_1 and the virtual registers $\overline{r'}^n$ are updated; otherwise, the returned expression is that labeled by l_2 and the virtual registers $\overline{r''}^n$ are updated.

As an example, let us consider the expression $\texttt{x}_0$=**new** `C()`;**return** $\texttt{x}_0$ in a program where `C` is defined and has no fields.

Then $\epsilon_{\mathcal{H}} \vdash \langle \epsilon_{fr}, \epsilon.\epsilon \rangle \{\texttt{x}_0\texttt{=}\textbf{new}\ \texttt{C();}\textbf{return}\ \texttt{x}_0\} \rightarrow \mathcal{H}, \langle \epsilon_{fr}, \epsilon.\epsilon \rangle \{\texttt{x}_0\texttt{=}o\texttt{;}\textbf{return}\ \texttt{x}_0\}$ by rules (ctx) (with context $\texttt{x}_0\texttt{=}[\cdot]\texttt{;}\textbf{return}\ \texttt{x}_0$) and (new), where $\mathcal{H} = o \mapsto \langle \texttt{C}, \epsilon \rangle$; $\mathcal{H} \vdash \langle \epsilon_{fr}, \epsilon.\epsilon \rangle \{\texttt{x}_0\texttt{=}o\texttt{;}\textbf{return}\ \texttt{x}_0\} \rightarrow \mathcal{H}, \langle fr, \epsilon.\epsilon \rangle \{o\texttt{;}\textbf{return}\ \texttt{x}_0\}$ by rules (ctx) (with context $[\cdot]\texttt{;}\textbf{return}\ \texttt{x}_0$) and (var-asn), where $fr = \texttt{x} \mapsto o, \texttt{x}_0 \mapsto o$; $\mathcal{H} \vdash \langle fr, \epsilon.\epsilon \rangle \{o\texttt{;}\textbf{return}\ \texttt{x}_0\} \rightarrow \mathcal{H}, \langle fr, \epsilon.\epsilon \rangle \{\textbf{return}\ \texttt{x}_0\}$ by rules (ctx) (with context $\langle fr, \epsilon.\epsilon \rangle \{[\cdot]\}$) and (seq); finally, $\mathcal{H} \vdash \langle fr, \epsilon.\epsilon \rangle \{\textbf{return}\ \texttt{x}_0\} \rightarrow \mathcal{H}, o$ by rule (return) (with context $[\cdot]$).

4 Abstract Compilation

In this section we define an abstract compilation scheme for programs in the SSI IR presented in Section 3. Programs are translated into a Horn formula *Hf* (that is, a logic program) and a goal B; type analysis and inference amounts to coinductive resolution of B, that is the greatest Herbrand model of Hf [5] is considered. The proof of soundness of such a translation is sketched in the Appendix.

In previous work [5,6] we have used expressive structural types; however, since SSI favors more precise type analysis, we have preferred to follow a simpler approach based on nominal types. Structural types could be used as well to allow data polymorphism, with the downside that subtyping relation becomes much more involved and termination issues must be addressed.

Subtyping is treated as an ordinary predicate, thus allowing only global analysis; compositional analysis can be obtained by considering subtyping as a constraint, and by using coinductive constraint logic programming [4].

The compilation of programs, class, and method declarations is defined in Figure 5. We follow the usual syntactic conventions for logic programs: logical variable names begin with upper case, whereas predicate and functor names begin with lower case letters. Underscore denotes anonymous logical variables that occur only once in a clause; [ ] and $[e|l]$ respectively represent the empty list, and the list where e is the first element, and l is the rest of the list.

$$(\text{prog})\frac{\forall\, i = 1..n\ cd_i \rightsquigarrow \mathit{Hf}_i \qquad e \rightsquigarrow (t \,|\, B)}{\overline{cd}^n\ e \rightsquigarrow (\mathit{Hf}^d \cup \overline{\mathit{Hf}}^n | B)}$$

$$(\text{class})\frac{\forall\, i = 1..k\ md_i\ \textbf{in}\ c_1 \rightsquigarrow \mathit{Hf}_i \quad \mathit{inhFields}(c_1) = \overline{f'}^h}{\begin{array}{l}\texttt{class}\ c_1\ \texttt{extends}\ c_2\ \{\ \overline{f}^n\ \ \overline{md}^k\ \} \rightsquigarrow \overline{\mathit{Hf}}^k \cup \\ \left\{\begin{array}{l} \mathit{class}(c_1) \leftarrow \mathit{true}. \\ \mathit{extends}(c_1, c_2) \leftarrow \mathit{true}. \\ \overline{\mathit{dec\_field}(c_1, f)}^n \leftarrow \mathit{true}. \\ \mathit{new}(\mathit{CE}, c_1, [\overline{T'}^h, \overline{T}^n]) \leftarrow \mathit{new}(\mathit{CE}, c_2, [\overline{T'}^h]), \overline{\mathit{field\_upd}(\mathit{CE}, c_1, f, T)}^n. \end{array}\right\}\end{array}}$$

$$(\text{meth})\frac{\overline{b}^n \rightsquigarrow (t \,|\, B)}{\begin{array}{l} m(\overline{r}^n)\{\overline{b}^n\}\ \textbf{in}\ c \rightsquigarrow \\ \left\{\begin{array}{l} \mathit{dec\_meth}(c, m) \leftarrow \mathit{true}. \\ \mathit{has\_meth}(\mathit{CE}, c, m, [\mathit{This}_0, \overline{r}^n], t) \leftarrow \mathit{subclass}(\mathit{This}_0, c), B. \end{array}\right\}\end{array}}$$

$$(\text{body})\frac{\forall\, i = 1..n\ b_i \rightsquigarrow B_i}{\overline{b}^n\ \ l{:}\texttt{return}\ r \rightsquigarrow (r \,|\, \overline{B}^n)}$$

Fig. 5. Compilation of programs, class, and method declarations and bodies

Each rule defines a different compilation judgment. The judgment $\overline{cd}^n\ e \rightsquigarrow (\mathit{Hf}^d \cup \overline{\mathit{Hf}}^n | B)$ states that the program $\overline{cd}^n\ e$ is compiled into the pair $(\mathit{Hf}^d \cup \overline{\mathit{Hf}}^n | B)$, where $\mathit{Hf}^d \cup \overline{\mathit{Hf}}^n$ is a Horn formula (that is, a set of Horn clauses), and B is a goal.[5] Type inference of the main expression is obtained by coinductive

[5] For simplicity we use the same meta-variable B to denote conjunctions of atoms (that is, clause bodies), and goals, even though more formally goals are special clauses of the form $\mathit{false} \leftarrow B$.

resolution of the goal B in $\mathit{Hf}^d \cup \overline{\mathit{Hf}}^n$. The Horn formula Hf^d contains all clauses that are independent of the program (Figure 7), whereas each Hf_i is obtained by compiling the class declaration cd_i (see below); the goal B is generated from the compilation of the main expression e (see below); the term t corresponds to the returned type of e; it is ignored here, but it is necessary for compiling expressions.

The compilation of a class declaration `class` c_1 `extends` c_2 $\{\ \overline{f}^n\ \ \overline{md}^k\ \}$ is a set of clauses, including each clause Hf_i obtained by compiling the method md_i (see below), clauses asserting that class c_1 declares field f_i, for all $i = 1..n$, and three specific clauses for predicates *class*, *extends*, and *new*. The clause for *new* deserves some explanations: the atom $\mathit{new}(ce, c, [\overline{t}^n])$ succeeds iff the invocation of the implicit constructor of c with n arguments of type $\overline{t}^n$ is type safe in the global class type environment ce. The class environment ce is required for compiling field access and update expressions (Figure 6): it is a finite map (simply represented by a list) associating class names with field records (finite maps again simply represented by lists) assigning types to all fields of a class. Class environments are required because of nominal types: abstract compilation with structural types allows data polymorphism on a per-object basis, whereas here we obtain only a very limited form of data polymorphism on a per-class basis. Type safety of object creation is checked by ensuring that object creation for the direct superclass c_2 is correct, where only the first part h of the arguments corresponding to the inherited fields (returned by the auxiliary function *inhFields* whose straightforward definition has been omitted) are passed; then, predicate *field_upd* defined in Figure 7 checks that all remaining n arguments, corresponding to the new fields declared in c_1, have types that are compatible with those specified in the class environment. The clause dealing with the base case for the root class *Object* is also defined in Figure 7.

The judgment $m(\overline{r}^n)\{\overline{b}^n\}$ **in** $c \rightsquigarrow \mathit{Hf}$ states that the method declaration $m(\overline{r}^n)\{\overline{b}^n\}$ contained in class c compiles to Horn clauses Hf. Two clauses are generated per method declaration: the first simply states that method m is declared in class c (and is needed to deal with inherited methods Figure 7), whereas the second is obtained by compiling the body of the method. The atom $\mathit{has\_meth}(ce, c, m, [t_0, \overline{t}^n], t)$ succeeds iff, in class environment ce method m of class c can be safely invoked on target object of type t_0, with n arguments of type $\overline{t}^n$ and returned value of type t. The predicate *subclass* (defined in Figure 7) ensures that the method can be invoked only on objects that are instances of c or one of its subclasses. For simplicity we assume that all names (including `this`) are translated to themselves, even though, in practice, appropriate injective renaming should be applied [5]. The compilation of a method body $\overline{b}^n$ l:`return` r consists of the type of the returned virtual register r, and the conjunction of all the atoms generated by the compilation of blocks $\overline{b}^n$.

Figure 6 defines abstract compilation for blocks, and expressions.

$$(\text{block})\frac{e \leadsto (t \mid B)}{l{:}e \leadsto B} \qquad (\text{seq})\frac{e_1 \leadsto (t_1 \mid B_1) \qquad e_2 \leadsto (t_2 \mid B_2)}{e_1; e_2 \leadsto (t_2 \mid B_1, B_2)}$$

$$(\text{c-jmp})\frac{\begin{array}{l}\text{if } \mathit{var}(r'''_i) = \mathit{var}(r)\\ \text{ then } t'_i = T, t''_i = F\\ \text{ else } t'_i = r'''_i, t''_i = r'''_i \qquad T, F \text{ fresh}\end{array}}{\begin{array}{c}\texttt{if } (r \texttt{ instanceof } c) \texttt{ with } \overline{(r', r'') = \sigma(r''')}^n \texttt{ jump } l_1 \texttt{ else jump } l_2 \texttt{ in } \leadsto\\ (\mathit{void} \mid \mathit{inter}(r, c, T), \mathit{diff}(r, c, F), \overline{\mathit{var\_upd}(r', t'), \mathit{var\_upd}(r'', t'')}^n)\end{array}}$$

$$(\text{var-upd})\frac{e \leadsto (t \mid B)}{r = e \leadsto (t \mid B, \mathit{var\_upd}(r, t))} \qquad (\text{jmp})\frac{}{\texttt{jump } l \leadsto (\mathit{void} \mid \mathit{true})}$$

$$(\text{field-upd})\frac{e_1 \leadsto (t_1 \mid B_1) \qquad e_2 \leadsto (t_2 \mid B_2)}{e_1.f = e_2 \leadsto (t_2 \mid B_1, B_2, \mathit{field\_upd}(CE, t_1, f, t_2))}$$

$$(\text{phi})\frac{}{r = \varphi(\overline{r}^n) \leadsto (\bigvee \overline{r}^n \mid \mathit{var\_upd}(r, \bigvee \overline{r}^n))}$$

$$(\text{new})\frac{\forall\, i = 1..n \;\; e_i \leadsto (t_i \mid B_i)}{\texttt{new } c(\overline{e}^n) \leadsto (c \mid \overline{B}^n, \mathit{new}(CE, c, [\overline{t}^n]))}$$

$$(\text{field-acc})\frac{e \leadsto (t \mid B) \quad R \text{ fresh}}{e.f \leadsto (R \mid B, \mathit{field}(CE, t, f, R))}$$

$$(\text{invk})\frac{\forall\, i = 0..n \;\; e_i \leadsto (t_i \mid B_i) \quad R \text{ fresh}}{e_0.m(\overline{e}^n) \leadsto (R \mid B_0, \overline{B}^n, \mathit{invoke}(CE, t_0, m, [\overline{t}^n], R))}$$

$$(\text{var})\frac{}{r \leadsto (r \mid \mathit{true})}$$

Fig. 6. Compilation of blocks and expressions

Compiling a block $l{:}e$ returns the conjunction of atoms obtained by compiling e; the type t of e is discarded. The compilation of $e_1; e_2$ returns the type of e_2 and the conjunction of atoms generated from the compilation of e_1 and e_2. The compilation of an unconditional jump generates the type *void* and the empty conjunction of atoms *true*. A conditional jump has type *void* as well, but a non-empty sequence of predicates is generated to deal with the splitting performed by the σ-functions; predicates *inter* and *diff* (defined in Figure 7) compute the intersection T and the difference F between the type of r and c, respectively, and predicate *var_upd* (defined in Figure 7) ensures that the type of virtual registers r'_i and r''_i are compatible with the pairs of types returned by the σ-functions. In case r'''_i refers to the same variable of r the types of such a pair are the computed intersection T and difference F, respectively, otherwise the pair (r'''_i, r'''_i) is returned (hence, no split is actually performed). Compilation of assignments to virtual registers and fields yields the conjunction of the atoms generated from the corresponding sub-expressions, together with the atoms that ensure that the assignment is type compatible (with predicates *var_upd* and *field_upd* defined in Figure 7). The returned type is the type of the right-hand side expression. Compilation of φ-function assignments to virtual registers is just an instantiation of rule (var-upd) where the type of the expression is the union of the types of the virtual registers passed as arguments to φ. Compilation rules for object creation, field selection, and method invocation follow the same pattern: the type of the expression is a fresh logical variable (except for object creation) corresponding to the type returned by the specific predicate (*new*, *field*, and *invoke* defined in Figure 7). The generated atoms are those obtained from the compilation of the sub-expressions, together with the atom specific of the expression. Rules (var) is straightforward.

The clauses (see Hf^d in (prog)) that do not depend on the specific program are defined in Figure 7 in the Appendix.

Predicate *subtype* defining the subtyping relation deserves some comments: as expected, classes c_1 and c_2 are both subtypes of $c_1 \vee c_2$, but c_1 is not a subtype of c_2 when C_1 is a proper subclass of c_2: since no rule is imposed on method overriding, subclassing is not subtyping. Consider for instance the following source code snippet:

```
class Square { ... equals(s){return this.side==s.side;} ... }
class ColoredSquare extends Square {
    ... equals(cs){return this.side==cs.side&&this.color==cs.color;} ... }
```

According to our compilation scheme, the expression `s1.equals(s2)` has type `Bool` if `s1` and `s2` have type `Square` and `Square`$\vee$`ColoredSquare`, respectively, but the same expression is not well-typed if `s1` has type `ColoredSquare` (hence, `ColoredSquare`$\not\leq$`Square`), since `s2` cannot contain an instance of `Square` for which field `color` is not defined. Subtyping is required for defining the predicates *var_upd* and *field_upd* for virtual register and field updates: the type of the source must be a subtype of the type of the destination.

Type *empty* is the bottom of the subtyping relation. The predicates *inter* and *diff* (see below) can generate *empty* when a branch is unreachable. Of course, field accesses and method invocations on type *empty* are correct (in practice they can only occur in dead code).

Predicate *field* looks up the type of a field in the global class environment, and is defined in terms of the auxiliary predicates *has_field*, *class_fields*, *field_type*, and *no_def*. In particular, predicate *has_field* checks that a class has actually a certain field, either declared or inherited. The definitions of *class_fields*, *field_type*, and *no_def* are straightforward (*no_def* ensures that a map does not contain multiple entries for a key), whereas the clause for *has_field* dealing with inherited fields is similar to the corresponding one for *invoke* (see below).

If the target object has a class type c, then the correctness of method invocation is checked with predicate *has_meth* applied to class c and to the same list of arguments where, however, the type c of **this** is added at the beginning. If the target object has a union type, predicate *invoke* checks that method invocation is correct for both types of the union, and then merges the types of the results into a single union type.

Finally, the clause for *has_meth* deals with the inherited methods: if class c does not declare method m, then *has_meth* must hold on the direct superclass of c.

Predicates *inter* and *diff* define type splitting for σ-functions; both predicates never fail, but the type *empty* is returned when a branch is not reachable. Their definition is straightforward: the former keeps all classes that satisfies the runtime typechecking, the latter all classes that do not satisfy it.

5 Conclusion

We have shown how SSI IR can be exploited by abstract compilation for static global type analysis of programs written in a dynamic object-oriented language equipped with an **instanceof** operator for runtime typechecks. The approach allows a rather precise analysis with just nominal and union types.

We have stated soundness of type inference and sketched the proof; to do that, a small step operational semantics of the SSI IR language has been formally defined; this is also an original contribution, since, to our knowledge, no prior formal definition of the semantics of an SSI IR language can be found in literature.

There already exists interesting work on type inference of dynamic object-oriented languages, and several papers have defined and studied flow sensitive type systems [7,1,11,13] (to mention just a few). The main distinguishing feature of abstract compilation, when compared with all other approaches, is its modularity. Abstract compilation allows one to implement different kinds of analysis, for different languages with a suitable compilation scheme, by using the same inference engine. Furthermore, abstract compilation is particularly suited for directly compiling IR languages, as SSI, to greatly improve the precision of the analysis.

Guarded type promotion [21] for Java allows more precise type analysis of branches guarded by dynamic type checks in a very similar way as in our approach. However, here we consider the more challenging problem of inferring types for a dynamic language where no type annotations are provided by the programmers.

References

1. An, J., Chaudhuri, A., Foster, J.S., Hicks, M.: Dynamic inference of static types for Ruby. In: POPL, pp. 459–472 (2011)
2. Ananian, C.S.: The static single information form. Technical Report MITLCS-TR-801. MIT (1999)
3. Ancona, D., Ancona, M., Cuni, A., Matsakis, N.: RPython: a Step Towards Reconciling Dynamically and Statically Typed OO Languages. In: DLS 2007, pp. 53–64. ACM (2007)
4. Ancona, D., Corradi, A., Lagorio, G., Damiani, F.: Abstract Compilation of Object-Oriented Languages into Coinductive CLP(X): Can Type Inference Meet Verification? In: Beckert, B., Marché, C. (eds.) FoVeOOS 2010. LNCS, vol. 6528, pp. 31–45. Springer, Heidelberg (2011)
5. Ancona, D., Lagorio, G.: Coinductive Type Systems for Object-Oriented Languages. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 2–26. Springer, Heidelberg (2009)

6. Ancona, D., Lagorio, G.: Idealized coinductive type systems for imperative object-oriented programs. RAIRO - Theoretical Informatics and Applications 45(1), 3–33 (2011)
7. Anderson, C., Giannini, P., Drossopoulou, S.: Towards Type Inference for JavaScript. In: Gao, X.-X. (ed.) ECOOP 2005. LNCS, vol. 3586, pp. 428–452. Springer, Heidelberg (2005)
8. Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Efficiently computing static single assignment form and the control dependence graph. TOPLAS 13, 451–490 (1991)
9. Das, D., Ramakrishna, U.: A practical and fast iterative algorithm for phi-function computation using DJ graphs. TOPLAS 27(3), 426–440 (2005)
10. Alpern, B., et al.: The jalapeño virtual machine. IBM Systems Journal 39 (2000)
11. Foster, J.S., Terauchi, T., Aiken, A.: Flow-sensitive type qualifiers. In: PLDI, pp. 1–12 (2002)
12. Griesemer, R., Mitrovic, S.: A compiler for the java hotspottm virtual machine. In: The School of Niklaus Wirth, "The Art of Simplicity", pp. 133–152 (2000)
13. Heidegger, P., Thiemann, P.: Recency Types for Analyzing Scripting Languages. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 200–224. Springer, Heidelberg (2010)
14. Holloway, G.: The machine-SUIF static single assignment library. Technical report, Harvard School of Engineering and Applied Sciences (2001)
15. Novillo, D.: Tree SSA - a new optimization infrastructure for GCC. In: GCC Developers' Summit, pp. 181–193 (2003)
16. Simon, L., Bansal, A., Mallya, A., Gupta, G.: Co-Logic Programming: Extending Logic Programming with Coinduction. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 472–483. Springer, Heidelberg (2007)
17. Simon, L., Mallya, A., Bansal, A., Gupta, G.: Coinductive Logic Programming. In: Etalle, S., Truszczyński, M. (eds.) ICLP 2006. LNCS, vol. 4079, pp. 330–345. Springer, Heidelberg (2006)
18. Singer, J.: Static single information form in machine SUIF. Technical report, University of Cambridge Computer Laboratory, UK (2004)
19. Singer, J.: Static Program Analysis based on Virtual Register Renaming. PhD thesis, Christs College (2005)
20. Tavares, A., Pereira, F.M., Bigonha, M., Bigonha, R.: Efficient SSI conversion. In: SBLP 2010 (2010)
21. Winther, J.: Guarded type promotion (eliminating redundant casts in Java). In: FTfJP 2011. ACM (2011)

$$\begin{array}{l}
\mathit{class}(\mathit{object}) \leftarrow \mathit{true}.\\
\mathit{subclass}(X, X) \leftarrow \mathit{class}(X).\\
\mathit{subclass}(X, Y) \leftarrow \mathit{extends}(X, Z), \mathit{subclass}(Z, Y).\\
\mathit{subtype}(\mathit{empty}, \_) \leftarrow \mathit{true}.\\
\mathit{subtype}(T, T) \leftarrow \mathit{true}.\\
\mathit{subtype}(T1 \vee T2, T) \leftarrow \mathit{subtype}(T1, T), \mathit{subtype}(T2, T).\\
\mathit{subtype}(T, T1 \vee \_) \leftarrow \mathit{subtype}(T, T1).\\
\mathit{subtype}(T, \_ \vee T2) \leftarrow \mathit{subtype}(T, T2).\\
\mathit{field}(CE, \mathit{empty}, \_, \_) \leftarrow \mathit{true}.\\
\mathit{field}(CE, C, F, T) \leftarrow \mathit{has\_field}(C, F), \mathit{class\_fields}(CE, C, R), \mathit{field\_type}(R, F, T).\\
\mathit{field}(CE, T1 \vee T2, F, FT1 \vee FT2) \leftarrow \mathit{field}(CE, T1, F, FT1),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{field}(CE, T2, F, FT2).\\
\mathit{class\_fields}([C : R|CE], C, R) \leftarrow \mathit{no\_def}(C, CE).\\
\mathit{class\_fields}([C1 : \_|CE], C2, R) \leftarrow \mathit{class\_fields}(CE, C2, R), C1 \neq C2.\\
\mathit{field\_type}([F{:}T|R], F, T) \leftarrow \mathit{no\_def}(F, R).\\
\mathit{field\_type}([F1 : \_|R], F2, T) \leftarrow \mathit{field\_type}(R, F2, T), F1 \neq F2.\\
\mathit{no\_def}(\_, [\,]) \leftarrow \mathit{true}.\\
\mathit{no\_def}(K1, [K2 : \_|Tl]) \leftarrow \mathit{no\_def}(K1, Tl), K1 \neq K2.\\
\mathit{invoke}(\_, \mathit{empty}, \_, \_, \_) \leftarrow \mathit{true}.\\
\mathit{invoke}(CE, C, M, A, RT) \leftarrow \mathit{has\_meth}(CE, C, M, [C|A], RT).\\
\mathit{invoke}(CE, T1 \vee T2, M, A, RT1 \vee RT2) \leftarrow \mathit{invoke}(CE, T1, M, A, RT1),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{invoke}(CE, T2, M, A, RT2).\\
\mathit{new}(\_, \mathit{object}, [\,]) \leftarrow \mathit{true}.\\
\mathit{has\_field}(C, F) \leftarrow \mathit{dec\_field}(C, F).\\
\mathit{has\_field}(C, F) \leftarrow \mathit{extends}(C, P), \mathit{has\_field}(P, F), \neg \mathit{dec\_field}(C, F).\\
\mathit{has\_meth}(CE, C, M, A, R) \leftarrow \mathit{extends}(C, P), \mathit{has\_meth}(CE, P, M, A, R),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \neg \mathit{dec\_meth}(C, M).\\
\mathit{var\_upd}(T1, T2) \leftarrow \mathit{subtype}(T2, T1).\\
\mathit{field\_upd}(CE, C, F, T2) \leftarrow \mathit{field}(CE, C, F, T1), \mathit{subtype}(T2, T1).\\
\mathit{inter}(C1, C2, C1) \leftarrow \mathit{subclass}(C1, C2).\\
\mathit{inter}(C1, C2, \mathit{empty}) \leftarrow \neg \mathit{subclass}(C1, C2).\\
\mathit{inter}(T1 \vee T2, C, IT1 \vee IT2) \leftarrow \mathit{inter}(T1, C, IT1), \mathit{inter}(T2, C, IT2).\\
\mathit{diff}(C1, C2, C1) \leftarrow \mathit{class}(C1), \neg \mathit{subclass}(C1, C2).\\
\mathit{diff}(C1, C2, \mathit{empty}) \leftarrow \mathit{class}(C1), \mathit{subclass}(C1, C2).\\
\mathit{diff}(T1 \vee T2, C, IT1 \vee IT2) \leftarrow \mathit{diff}(T1, C, IT1), \mathit{diff}(T2, C, IT2).
\end{array}$$

Fig. 7. Clauses defining the predicates used by the abstract compilation

A Proof of Soundness

To sketch the proof of soundness of abstract compilation, abstract compilation of expressions has to be extended to cover also runtime expressions, hence we define the new judgment $\mathcal{H}, \mathit{fr}, e \rightsquigarrow (t \mid B)$ (defined in Figure 8) stating that the runtime expression e compiles to the type t and the conjunction of atoms B in the heap $\mathcal{H}$ and stack frame fr. Heaps and stack frames are needed for compiling object values and virtual registers. All rules obtained as straightforward extension of the corresponding rules in Figure 6 have been omitted. In the sequel, all statements depend on a particular program $\overline{cd}^n$; furthermore, we state that the

$$(\text{obj})\frac{\mathcal{H}(o) = \langle c, \_\rangle}{\mathcal{H}, fr, o \rightsquigarrow (c \mid true)} \qquad (\text{frame})\frac{\mathcal{H}, fr', e \rightsquigarrow (t \mid B)}{\mathcal{H}, fr, \langle fr', \mu\rangle\{e\} \rightsquigarrow (t \mid B)}$$

$$(\text{c-jmp-then})\frac{fr(r) = c' \quad c' \leq c \quad \forall\, i = 1..n\ fr(r'_i) = t'_i \quad \begin{array}{l}\text{if } var(r'''_i) = var(r) \\ \text{then } t''_i = c' \text{ else } t''_i = fr(r'''_i)\end{array}}{\begin{array}{l}\mathcal{H}, fr, \texttt{if } (r \texttt{ instanceof } c) \texttt{ with } \overline{(r', r'') = \sigma(r''')}^{n} \texttt{ jump } l_1 \texttt{ else jump } l_2 \textbf{ in} \rightsquigarrow \\ (void \mid \overline{var\_upd(t', t'')}^{n})\end{array}}$$

$$(\text{c-jmp-else})\frac{fr(r) = c' \quad c' \not\leq c \quad \forall\, i = 1..n\ fr(r''_i) = t'_i \quad \begin{array}{l}\text{if } var(r'''_i) = var(r) \\ \text{then } t''_i = c' \text{ else } t''_i = fr(r'''_i)\end{array}}{\begin{array}{l}\mathcal{H}, fr, \texttt{if } (r \texttt{ instanceof } c) \texttt{ with } \overline{(r', r'') = \sigma(r''')}^{n} \texttt{ jump } l_1 \texttt{ else jump } l_2 \textbf{ in} \rightsquigarrow \\ (void \mid \overline{var\_upd(t', t'')}^{n})\end{array}}$$

$$(\text{var-upd})\frac{\mathcal{H}, fr, e \rightsquigarrow (t \mid B) \quad fr(r) = t'}{\mathcal{H}, fr, r = e \rightsquigarrow (t \mid B, var\_upd(t', t))}$$

$$(\text{phi})\frac{\forall\, i = 0..n\ fr(r_i) = t_i}{\mathcal{H}, fr, r_0 = \varphi(\overline{r}^{n}) \rightsquigarrow (\overline{\vee t}^{n} \mid var\_upd(t_0, \overline{\vee t}^{n}))} \qquad (\text{var})\frac{fr(r) = t}{\mathcal{H}, fr, r \rightsquigarrow (t \mid true)}$$

Fig. 8. Compilation of runtime expressions

coinductive resolution of a goal succeeds to mean that it succeeds w.r.t. the abstract compilation of $\overline{cd}^{n}$.

Lemma 1 (Progress). *If $\mathcal{H}, \epsilon_{fr}, e \rightsquigarrow (t \mid B)$, and the coinductive resolution of B succeeds, then either e is a value, or there exist $\mathcal{H}'$ and e' s.t. $\mathcal{H} \vdash e \rightarrow \mathcal{H}', e'$.*

Lemma 2 (Subject Reduction). *If $\mathcal{H} \vdash e \rightarrow \mathcal{H}', e'$ and $\mathcal{H}, \epsilon_{fr}, e \rightsquigarrow (t \mid B)$ and the coinductive resolution of B succeeds with grounding substitution θ, then there exist t' and B' s.t. $\mathcal{H}, \epsilon_{fr}, e' \rightsquigarrow (t' \mid B')$, the coinductive resolution of B' succeeds with grounding substitution θ', and $subtype(\theta' t', \theta t)$ succeeds.*

Theorem 1 (Soundness). *If $\mathcal{H}, \epsilon_{fr}, e \rightsquigarrow (t \mid B)$ and the coinductive resolution of B succeeds with grounding substitution θ, and $\mathcal{H} \vdash e \rightarrow^{*} \mathcal{H}', e'$, and there exist no $\mathcal{H}''$ and e'' s.t. $\mathcal{H}' \vdash e' \rightarrow \mathcal{H}'', e''$, then e' is an object value o s.t. $\mathcal{H}'(o) = \langle c, \_\rangle$, and $subtype(c, \theta t)$ succeeds.*

Input-Driven Stack Automata

Suna Bensch[1], Markus Holzer[2], Martin Kutrib[2], and Andreas Malcher[2]

[1] Department of Computing Science,
Umeå University, 90187 Umeå, Sweden
suna@cs.umu.se

[2] Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,kutrib,malcher}@informatik.uni-giessen.de

Abstract. We introduce and investigate input-driven stack automata, which are a generalization of input-driven pushdown automata that recently became popular under the name visibly pushdown automata. Basically, the idea is that the input letters uniquely determine the operations on the pushdown store. This can nicely be generalized to stack automata by further types of input letters which are responsible for moving the stack pointer up or down. While visibly pushdown languages share many desirable properties with regular languages, input-driven stack automata languages do not necessarily so. We prove that deterministic and nondeterministic input-driven stack automata have different computational power, which shows in passing that one cannot construct a deterministic input-driven stack automaton from a nondeterministic one. We study the computational capacity of these devices. Moreover, it is shown that the membership problem for nondeterministic input-driven stack automata languages is NP-complete.

1 Introduction

Finite automata have intensively been studied and, moreover, have been extended in several different ways. Typical extensions in the view of [8] are pushdown tapes [4], stack tapes [9], or Turing tapes. The investigations in [8] led to a rich theory of abstract families of automata, which is the equivalent to the theory of abstract families of languages (see, for example, [18]). On the other hand, slight extensions to finite automata such as a one-turn pushdown tape lead to machine models that can no longer be determinized, that is, the nondeterministic machine model is more powerful than the deterministic one. Moreover, fundamental problems such as membership become more complicated than for languages accepted by finite automata. For example, the equivalence problem turns out to be undecidable, while for regular languages this problem is decidable, and its complexity depends on the machine type used (deterministic or nondeterministic finite automata).

Recently a pushdown automaton model, called visibly pushdown automaton, was popularized by [1], which shares many desirable properties with regular languages, but still is powerful enough to describe important context-free-like behavior. The idea on visibly pushdown automata is that the input letters uniquely determine whether the automaton pushes a symbol, pops a symbol, or leaves the pushdown unchanged. Such devices date back to the seminal paper [15] and its follow-ups [2] and [6], where this machine model is called input-driven pushdown automaton. One of the most important properties on visibly pushdown automata languages or, equivalently, input-driven pushdown automata languages is that deterministic and nondeterministic automata are equally powerful. Moreover, the language class accepted is closed under almost all basic operations in formal language theory. Since the recent paper [1], visibly pushdown automata are a vivid area of research, which can be seen by the amount of literature, for example, [1,3,5,10,16,17]. In some of these papers yet another name is used for visibly pushdown automata, namely nested word automata, which may lead to some confusion.

Here we generalize the idea of input-driven pushdown automata to input-driven stack automata. Since the main difference between a pushdown and a stack is that the latter storage type is also allowed to read information from the inside of the stack and not only from the top, the idea that input letters control the stack behavior easily applies. Hence, in addition to the letters that make the automaton push, pop, or leave the stack unchanged, two new types of letters that allow the movement of the stack pointer up or down are introduced. This leads us to the strong version of a input-driven stack automaton. Relaxing the condition on being input-driven when reading the stack contents, gives the basic idea of a weak input-driven stack automaton. We compare both automata models and show that the strong version is strictly less powerful than the corresponding weak version for deterministic devices. Moreover, when staying with the same model, nondeterminism turns out to be more powerful than determinism, which shows in passing that in both cases determinization is not possible. This sharply contrasts the situation for input-driven pushdown automata. Concerning decidability questions, we would like to note that the results in [9] imply that emptiness is decidable for nondeterministic input-driven stack automata and equivalence with regular languages is decidable for deterministic input-driven stack automata. Finally, we also show that the fixed membership problem for input-driven stack automata languages, even for the strong automaton model, has the same complexity as for languages accepted by ordinary stack automata, namely it is NP-complete, and therefore intractable. This again is in sharp contrast to the situation on input-driven pushdown automata languages, whose membership problem is NC^1-complete [6] while ordinary pushdown automata languages are LOGCFL-complete [20].

2 Preliminaries and Definitions

We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A nondeterministic one-way stack automaton is a classical nondeterministic pushdown automaton which is enhanced with an additional pointer that is allowed to move inside the stack without altering the stack contents. In this way it is possible to read but not to change information which is stored on the stack. Such a stack automaton is called input-driven, if the next input symbol defines the next action on the stack, that is, pushing a symbol onto the stack, popping a symbol from the stack, changing the internal state without changing the stack, or moving inside the stack by going up or down. To this end, we assume that the input alphabet Σ is partitioned into the sets Σ_c, Σ_r, Σ_i, Σ_u, and Σ_d, that control the actions push (call), pop (return), state change without changing the stack (internal), and up and down movement of the stack pointer. A formal definition reads as follows.

Definition 1. *A* strong nondeterministic one-way input-driven stack automaton (1sNVSA) *is a system* $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$*, where*

1. Q *is the finite set of* internal states*,*
2. Σ *is the finite set of* input symbols *consisting of the disjoint union of sets* Σ_c*,* Σ_r*,* Σ_i*,* Σ_u*, and* Σ_d*,*
3. Γ *is the finite set of* stack symbols*,*
4. $\perp \in \Gamma$ *is the* initial stack *or* bottom-of-stack symbol*,*
5. Γ' *is a marked copy of* Γ*, that is* $\Gamma' = \{\, a' \mid a \in \Gamma \,\}$*, where the symbol* $\perp'$ *is denoted by* $\perp$*,*
6. $q_0 \in Q$ *is the* initial state*,*
7. $F \subseteq Q$ *is the set of* accepting states*, and*
8. δ_c *is the partial transition function mapping* $Q \times \Sigma_c$ *into the subsets of* $Q \times (\Gamma' \setminus \{\perp\})$*,*
9. δ_r *is the partial transition function mapping* $Q \times \Sigma_r \times \Gamma'$ *into the subsets of* Q*,*
10. δ_i *is the partial transition function mapping* $Q \times \Sigma_i$ *into the subsets of* Q*,*
11. δ_d *is the partial transition function mapping* $Q \times \Sigma_d \times (\Gamma \cup \Gamma')$ *into the subsets of* Q*,*
12. δ_u *is the partial transition function mapping* $Q \times \Sigma_u \times (\Gamma \cup \Gamma')$ *into the subsets of* Q*.*

A *configuration* of a 1sNVSA $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$ at some time $t \geq 0$ is a quadruple $c_t = (q, w, s, p)$, where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, $s \in \Gamma'\Gamma^*\perp \cup \{\perp\}$ gives the current stack contents, and $1 \leq p \leq |s|$ gives the current position of the stack pointer. Let $s = s_n s_{n-1} \cdots s_1$ denote the stack contents. Consider the projection $[\cdot] : (\Gamma \cup \Gamma')^+ \to (\Gamma \cup \Gamma')$ such that $s[p] = s_p$, for $1 \leq p \leq n$. Furthermore, let φ be a mapping which marks the first letter of a string in Γ^+, that is, $\varphi : \Gamma^+ \to \Gamma'\Gamma^*$ such that $\varphi(a_1 a_2 \cdots a_n) = a_1' a_2 \cdots a_n$. By definition, $\varphi(\perp) = \perp$.

The *initial configuration* for input w is set to $(q_0, w, \perp, 1)$. During its course of computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $a \in \Sigma$, $w \in \Sigma^*$, $s \in \Gamma'\Gamma^*\perp \cup \{\perp\}$, $t \in \Gamma^*\perp$, $1 \leq p \leq |s|$, and $Z \in \Gamma$. We set

1. $(q, aw, s, p) \vdash (q', w, Z'\varphi^{-1}(s), |s| + 1)$, if $a \in \Sigma_c$ and $(q', Z) \in \delta_c(q, a)$,
2. $(q, aw, Z't, p) \vdash (q', w, \varphi(t), |t|)$, if $a \in \Sigma_r$ and $q' \in \delta_r(q, a, Z')$,
3. $(q, aw, \perp, 1) \vdash (q', w, \perp, 1)$, if $a \in \Sigma_r$ and $q' \in \delta_r(q, a, \perp)$,
4. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_i$ and $q' \in \delta_i(q, a)$,
5. $(q, aw, s, p) \vdash (q', w, s, p + 1)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \notin \Gamma'$,
6. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \in \Gamma'$,
7. $(q, aw, s, p) \vdash (q', w, s, p - 1)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] \neq \perp$,
8. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \perp$.

As usual, we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$.

So, the pushing of a symbol onto the stack is described by Σ_c and δ_c, and the popping of a symbol is described by Σ_r and δ_r. With the help of the mappings φ and φ^{-1} it is possible to mark the new topmost symbol suitably. The internal change of the state without altering the stack contents is described by Σ_i and δ_i. We remark that δ_c and δ_i do not depend on the topmost stack symbol, but only on the current state and input symbol. This is not a serious restriction since every automaton can be modified in such a way that the topmost stack symbol is additionally stored in the state set. In this context, the question may arise of how a state can store the new topmost stack symbol in case of popping. This can be solved by a similar construction as given in [12], where every pushdown automaton is converted to an equivalent pushdown automaton such that every stack symbol is a pair of stack symbols consisting of the symbol on the stack and its immediate predecessor.

The moves inside the stack are described by Σ_d, δ_d and Σ_u, δ_u, respectively. Up-moves at the top of the stack and down-moves at the bottom of the stack can only change the state, but do not affect the position of the stack pointer. So, the pointer can never go below the bottom and beyond the top of the stack. To ensure the latter the topmost stack symbol is suitably marked. By definition, transition functions δ_c and δ_r can only be applied if the stack pointer is at the topmost stack symbol. Thus, we stipulate the following behavior: if δ_c or δ_r have to be applied and the stack pointer is inside the stack, then the stack pointer is set to the topmost symbol, and the new symbol is pushed onto the stack or the topmost symbol is popped off the stack. The bottom-of-stack symbol $\perp$ can neither be pushed onto nor be popped from the stack.

The language accepted by a 1sNVSA is precisely the set of words w such that there is some computation beginning with the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered:

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \perp, 1) \vdash^* (q, \lambda, s, p) \text{ with } q \in F, \\ s \in \Gamma'\Gamma^*\perp \cup \{\perp\}, \text{ and } 1 \leq p \leq |s| \,\}.$$

If in any case each δ_r, δ_c, δ_i, δ_u, and δ_d is either undefined or a singleton, then the stack automaton is said to be *deterministic*. Strong deterministic stack automata are denoted by 1sDVSA. In case that no symbol is ever popped from the stack, that is, $\delta_r = \emptyset$, the stack automaton is said to be *non-erasing*. Strong nondeterministic and deterministic non-erasing stack automata are denoted by 1sNENVSA and 1sNEDVSA. The family of all languages accepted by an input-driven stack automaton of some type X is denoted by $\mathscr{L}(X)$.

In order to clarify our notion we continue with an example.

Example 2. The non-context-free language $\{ a^n b^n c^{n+1} \mid n \geq 1 \}$ is accepted by the 1sNEDVSA $M = \langle \{q_0, q_1, q_2, q_3\}, \Sigma, \{A, \bot\}, \bot, q_0, \{q_3\}, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$, where $\Sigma_c = \{a\}$, $\Sigma_u = \{c\}$, $\Sigma_d = \{b\}$, and $\Sigma_r = \Sigma_i = \emptyset$. The transition functions δ_r and δ_i are undefined, and δ_c, δ_u, and δ_d are as follows.

$$\begin{array}{llll}
(1) & \delta_c(q_0, a) = (q_0, A') & (4) & \delta_u(q_1, c, \bot) = q_2 \\
(2) & \delta_d(q_0, b, A') = q_1 & (5) & \delta_u(q_2, c, A) = q_2 \\
(3) & \delta_d(q_1, b, A) = q_1 & (6) & \delta_u(q_2, c, A') = q_3
\end{array}$$

Since δ_r is undefined, M is non-erasing. An input is accepted only if M eventually enters state q_3. To this end, it must be in state q_2. Similarly, to get into state q_2 it must be in state q_1, and the only possibility to change into state q_1 is from q_0.

If the input does not begin with an a, the computation blocks and rejects immediately. So, any accepting computation starts with a sequence of transitions (1) reading a prefix of the form a^n, for $n \geq 1$. This yields a configuration $(q_0, w_1, A'A^{n-1}\bot, n+1)$. Since for input symbol c no transition is defined from q_0, the remaining input w_1 must have a prefix of the form b^m, for $m \geq 1$. Therefore, M applies one transition (2) and, subsequently, tries to apply $m-1$ transitions (3). If $m < n$, this yields a configuration $(q_1, w_2, A'A^{n-1}\bot, p)$, where $2 \leq p \leq n$, and M blocks and rejects if an a or a c follows. Similarly, if $m > n$, after reading b^n a configuration $(q_1, w_2, A'A^{n-1}\bot, 1)$ is reached, on which M blocks and rejects when trying to read the next b. Therefore, in any accepting computation w_1 must begin with exactly n copies of b. Since for input symbol a no transition is defined from q_1, the remaining input w_2 must have a prefix of the form c^ℓ, for $\ell \geq 1$. Therefore, M applies one transition (4) and, subsequently, applies transitions (5). If $\ell < n$, this yields a configuration $(q_2, w_3, A'A^{n-1}\bot, p)$, where $2 \leq p \leq n$, and M blocks and rejects if an a or a b follows, and does not accept if the input has been consumed. If $\ell \geq n$, a configuration $(q_2, w_3, A'A^{n-1}\bot, n+1)$ is reached. Next, if $\ell = n+1$ then M applies transition (6) and accepts the input $a^n b^b c^{n+1}$. For $\ell > n+1$ the computation blocks. □

Next, we introduce weak variants of input-driven stack automata, for which moves inside the stack are not necessarily input-driven. To this end, we have to extend the domain of δ_d and δ_u appropriately and to adapt the derivation relation $\vdash$ accordingly. First we explain how to modify the transition functions in the definition of input-driven stack automata, where items 11 and 12 are changed to

11′. δ_d is the partial transition function mapping $Q \times \Sigma \times (\Gamma \cup \Gamma')$ into the subsets of Q,
12′. δ_u is the partial transition function mapping $Q \times \Sigma \times (\Gamma \cup \Gamma')$ into the subsets of Q.

In the weak mode, the stack can only by entered by reading an input symbol from Σ_d. Being inside the stack, the pointer may move up and down for any input symbol. When the top of the stack is reached, the stack is left and any new entering needs another input symbol from Σ_d. So, Σ_u is not necessary, but we keep it for the sake of compatibility. For the weak mode, the relation $\vdash$ is adapted by replacing items 5 to 8 by the following ones:

5′. $(q, aw, s, p) \vdash (q', w, s, p+1)$, if $a \in \Sigma$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \notin \Gamma'$,
6′. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \in \Gamma'$,
7′a. $(q, aw, s, p) \vdash (q', w, s, p-1)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, $s[p] \in \Gamma'$, and $s[p] \neq \bot$,
7′b. $(q, aw, s, p) \vdash (q', w, s, p-1)$, if $a \in \Sigma$, $q' \in \delta_d(q, a, s[p])$, $s[p] \in \Gamma$, and $s[p] \neq \bot$,
8′a. $(q, aw, s, p) \vdash (q', w, s, p)$, if $|s| = 1$, $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \bot$.
8′b. $(q, aw, s, p) \vdash (q', w, s, p)$, if $|s| \geq 2$, $a \in \Sigma$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \bot$.

If for every $q \in Q$, $a \in \Sigma$, and $Z \in \Gamma' \cup \Gamma$ each of the three sums $|\delta_r(q, a, Z)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$, $|\delta_c(q, a)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$, and $|\delta_i(q, a)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$ is at most one, then the weak stack automaton is said to be *deterministic*. Weak deterministic stack automata are denoted by 1wDVSA. Moreover, weak nondeterministic and deterministic non-erasing stack automata are denoted by 1wNENVSA and 1wNEDVSA, respectively.

3 Computational Capacity

We investigate the computational capacity of input-driven stack automata working in strong and weak mode, and prove that these machines induce a strict hierarchy of language families with respect to the following three features: (i) strong and weak mode, (ii) determinism and nondeterminism, and (iii) non-erasing and erasing stack. First we consider deterministic machines and compare weak non-erasing mode with strong erasing mode.

Lemma 3. $L_1 = \{\, a^n b a^{n-1} c \mid n \geq 1 \,\} \in \mathscr{L}(\mathit{1wNEDVSA}) \setminus \mathscr{L}(\mathit{1sDVSA})$.

Proof. The main idea for a 1wNEDVSA accepting L_1 is to push some symbol on the stack for every input symbol a. When the b appears in the input the stack pointer is moved one position down, that is, to position n. At the same time a certain state is entered which ensures that the stack pointer moves down for subsequent symbols a. In this way, the pointer reaches the bottom of the stack after reading a^{n-1}. If in this situation a c follows, the input is accepted with an up move, otherwise the input is rejected. Clearly, the automaton constructed is deterministic and non-erasing.

To show that $L_1 \notin \mathscr{L}$(1sDVSA), we assume in contrast to the assertion that L_1 is accepted by a 1sDVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Suppose that $a \notin \Sigma_c$. Since the stack is initially empty up to $\bot$, any application of δ_i, δ_u, δ_d, or δ_r can only alter the current state. Therefore, there are two sufficiently large natural numbers $n_1 \neq n_2$ and a state $q \in Q$, so that

$$(q_0, a^{n_1}ba^{n_1-1}c, \perp, 1) \vdash^* (q, ba^{n_1-1}c, \perp, 1) \vdash^* (f_1, \lambda, s_1, p_1)$$

and

$$(q_0, a^{n_2}ba^{n_2-1}c, \perp, 1) \vdash^* (q, ba^{n_2-1}c, \perp, 1) \vdash^* (f_2, \lambda, s_2, p_2),$$

for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\perp \cup \{\perp\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. This implies

$$(q_0, a^{n_1}ba^{n_2-1}c, \perp, 1) \vdash^* (q, ba^{n_2-1}c, \perp, 1) \vdash^* (f_2, \lambda, s_2, p_2)$$

and, thus, $a^{n_1}ba^{n_2-1}c \in L_1$, which is a contradiction.

Now suppose that $a \in \Sigma_c$. Then for every symbol a, some stack symbol is pushed onto the stack. Since Q is finite, the sequence of states passed through while reading a large number of a's is eventually periodic. Therefore, the sequence of symbols pushed onto the the stack is also eventually periodic. Say, it is of the form uv^*w, where u is a suffix of v, $v \in \Gamma^+$, $w \in \Gamma^*\perp$, and $|w|, |v| \leq |Q|$. Therefore, there are two sufficiently large natural numbers $n_1 \neq n_2$, a state $q \in Q$, strings u, v, w, where u is a suffix of v, $v \in \Gamma^+$, $w \in \Gamma^*\perp$, and natural numbers $k_2 > k_1 > 0$ so that

$$(q_0, a^{n_1}ba^{n_1-1}c, \perp, 1) \vdash^* (q, ba^{n_1-1}c, \varphi(u)v^{k_1}w, n_1+1) \vdash^* (f_1, \lambda, s_1, p_1)$$

and

$$(q_0, a^{n_2}ba^{n_2-1}c, \perp, 1) \vdash^* (q, ba^{n_2-1}c, \varphi(u)v^{k_2}w, n_2+1) \vdash^* (f_2, \lambda, s_2, p_2)$$

for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\perp$, and $1 \leq p_i \leq |s_i|$ for $i = 1, 2$. Since the input symbol b forces M to push or to pop a symbol, or to leave the stack as it is, the stack is decreased by at most one symbol when the b is read. The subsequent input symbols a increase the stack again. So, the bottommost n_1 (n_2) stack symbols are not touched again. Since $\varphi(u)v^{k_1}w$ and $\varphi(u)v^{k_2}w$ have a common prefix of length at most 2, we derive

$$(q_0, a^{n_1}ba^{n_2-1}c, \perp, 1) \vdash^* (q, ba^{n_2-1}c, \varphi(u)v^{k_1}w, n_1+1) \vdash^* (f_2, \lambda, s_3, p_3)$$

for some $s_3 \in \Gamma'\Gamma^*\perp$ and $p_3 = |s_3|$. This implies that $a^{n_1}ba^{n_2-1}c \in L_1$ which is a contradiction and, hence, $L_1 \notin \mathscr{L}(\text{1sDVSA})$. □

Now we turn to show how input-driven stack automata languages are related to some important context-free language families. Let CFL refer to the family of context-free languages. Then Example 2 shows the following result.

Lemma 4. *$L_2 = \{\, a^nb^nc^{n+1} \mid n \geq 1 \,\} \in \mathscr{L}(\mathit{1sNEDVSA}) \setminus \mathit{CFL}$.* □

The next lemma proves a converse relation, namely that a deterministic and linear context-free language is not accepted by any deterministic weak input-driven stack automaton. Let DCFL refer to the family of deterministic context-free and LIN to the family of linear context-free languages, which are both strict sub-families of CFL.

Lemma 5. *Let* $L_3 = \{\, a^n b^m a b^m a^n \mid n, m \geq 1 \,\} \cup \{\, b^n a^n \mid n \geq 1 \,\}$. *Then,* $L_3 \in (\mathit{DCFL} \cap \mathit{LIN}) \setminus \mathscr{L}(\mathit{1wDVSA})$.

Proof. Clearly, L_3 belongs to DCFL $\cap$ LIN. Now assume that L_3 is accepted by some 1wDVSA $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similarly as in the proof of Lemma 3, we conclude $a \in \Sigma_c$ and $b \in \Sigma_c$, since otherwise the words from L_3 of the form b^+a^+ could not be accepted. Thus, every input from $\{a, b\}^+$ forces M to only push symbols onto the stack. Continuing similar as in the second part of the proof of Lemma 3 shows that then words not belonging to L_3 are accepted. This contradiction shows the lemma. □

For the proof of the following lemma that compares deterministic weak and strong input-driven stack automata (conversely to Lemma 3), we use an incompressibility argument. General information on Kolmogorov complexity and the incompressibility method can be found in [14]. Let $w \in \{0,1\}^+$ be an arbitrary binary string of length n. Then the plain Kolmogorov complexity $C(w|n)$ of w denotes the minimal size of a program that knows n and describes w. It is well known that there exist binary strings w of arbitrary length n such that $C(w|n) \geq n$ (see [14], Theorem 2.2.1). Similarly, for any natural number n, $C(n)$ denotes the minimal size of a program that describes n. It is known that there exist infinitely many natural numbers n such that $C(n) \geq \log(n)$.

Lemma 6. *Let* $\hat{}\, : \{0,1\}^* \to \{\hat{0}, \hat{1}\}^*$ *be the homomorphism that maps* 0 *to* $\hat{0}$ *and* 1 *to* $\hat{1}$. $L_4 = \{\, a^{n+m} b^m w \hat{w}^R b^n \mid m, n \geq 1, w \in \{0,1\}^+ \,\} \cup \{\, a^n b^n \mid n \geq 1 \,\} \in \mathscr{L}(\mathit{1sDVSA}) \setminus \mathscr{L}(\mathit{1wNEDVSA})$.

Proof. The rough idea of a 1sDVSA M for L_4 is to push a symbol A onto the stack for every input symbol a, and to pop symbol A from the stack for every input symbol b read. In order to check the infix $w\hat{w}^R$, M pushes a Z for every 0 and an O for every 1 while reading w. Subsequently, it pops a Z for every $\hat{0}$ and an O for every $\hat{1}$ while reading $\hat{w}$. If eventually the stack is empty up to $\perp$, the input is to be accepted and otherwise rejected. The concrete construction on M is straightforward except for one detail. The machine must be able to recognize when the stack is empty. To be more precise, it must know when the stack pointer is moved at the bottom-of-stack symbol even if there are no more moves, as for accepting computations. In order to implement this detail, M marks the first symbol on the stack. When this symbol is popped again, M knows that the stack is empty even without reading $\perp$.

Next, we show that $L_4 \notin \mathscr{L}(\text{1wNEDVSA})$. Contrarily, assume that L_4 is accepted by a 1wNEDVSA $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similar as in the proof of Lemma 3, we conclude that $a \in \Sigma_c$. Moreover, since $\{\, a^n b^n \mid n \geq 1 \,\}$ is a subset of L_4, we conclude that $b \in \Sigma_d$. Otherwise, $b \in \Sigma_c$ or $b \in \Sigma_i$ which leads to a contradiction as shown before. Now we consider an accepting computation K on an input of the form $z = a^{n+m} b^m w \hat{w}^R b^n$, for $w \in \{0,1\}^+$, $n \geq 2$, $k = \lfloor \log(n) \rfloor$, and $m = 2^{2^k} - n$. We distinguish two cases.

1. First, we assume that in K nothing is pushed onto the stack while reading the infix w, and consider the configuration $c = (q, \hat{w}^R b^n, s, p)$ with $s \in \Gamma' \Gamma^* \perp$

and $1 \leq p \leq |s|$ after reading $a^{n+m}b^m w$. We claim that the knowledge of M, n, m, $|w|$, q, and p suffices to write a program that outputs w. This is seen as follows.

Since n and m are known, the stack contents s can be computed by simulating M on input a^{n+m}. Furthermore, due to our assumption that nothing is pushed while reading w, and since q and the pointer position p are known, it is possible to simulate M starting with a situation as configuration c but with arbitrary input. This is done successively on all inputs $\hat{v}b^n$ with $|\hat{v}| = |w|$. If $\hat{v} = \hat{w}^R$, then the simulation ends accepting. On the other hand, if the simulation is accepting, then $\hat{v} = \hat{w}^R$, since otherwise $a^{n+m}b^m w\hat{v}b^n$ with $\hat{v} \neq \hat{w}^R$ would belong to L_4 which is a contradiction. This suffices to identify and output w. The size of the program is a constant for the code itself plus the sizes of M, n, m, $|w|$, q, and p. The size of M and, thus, of q is also a constant, while $p \leq n + m$. So, the size of the program is of order $O(\log(n) + \log(m) + \log(n+m) + \log(|w|))$. Fixing n and m and choosing $|w|$ large enough, we obtain $C(w||w|) \in o(|w|)$ which is a contradiction to the above-cited result that there exist binary strings w of arbitrary length such that $C(w||w|) \geq |w|$.

2. Second, we assume that in K something is pushed onto the stack while reading the infix w, and consider the configuration $c = (q, x, s, |s|)$ with $s \in \Gamma'\Gamma^*\bot$ and $x \in \{0,1\}^*\{\hat{0},\hat{1}\}^+b^+$ is the remaining input when the first symbol has been pushed onto the stack while reading w. We claim that the knowledge of M, k, $|w|$, the length ℓ of the prefix w' of w which has been already read, q, and the last symbol B pushed suffices to write a program that outputs n.

   The stack height $|s|$ is $m + n + 1 = 2^{2^k} + 1$. Therefore, the stack contents s can be computed by simulating M on input $a^{|s|-1}$ using solely the knowledge of k, and adding B on the top. Next the simulation of M beginning in state q while having the stack pointer on the top of the stack contents s is started successively on all inputs $u\hat{v}b^i$ with $u \in \{0,1\}^*$, $|u| = |w| - \ell$, $|\hat{v}| = |w|$, and $1 \leq i \leq 2^{2^k}$. If $w'u\hat{v} = w\hat{w}^R$ and $i = n$, then the simulation ends accepting. On the other hand, if the simulation is accepting, then $w'u\hat{v} = w\hat{w}^R$ and $i = n$. This suffices to identify and output n. Again, the size of the program is a constant for the code itself plus the sizes of M, k, $|w|$, ℓ, q, and $|\Gamma|$. The size of M and, thus, of q and $|\Gamma|$ is also a constant, while $\ell \leq |w|$. So, the size of the program is of order $O(\log(k) + \log(|w|)) = O(\log(\log(n)) + \log(|w|))$. Fixing w and choosing n large enough, we obtain $C(n) \in o(\log(n))$ which is a contradiction since there are infinitely many natural numbers n such that $C(n) \geq \log(n)$.

This proves the lemma. □

With the help of the previous four lemmata we can show the following strict inclusion relations and incomparability results.

Theorem 7. *All inclusions shown in Figure 1 are strict. Moreover, language families that are not linked by a path are pairwise incomparable.*

$$\begin{tikzcd}
 & \mathscr{L}(\text{1wDVSA}) & \mathscr{L}(\text{1wNEDVSA}) \arrow[l] \\
\text{CFL} & \mathscr{L}(\text{1sDVSA}) \arrow[u] & \mathscr{L}(\text{1sNEDVSA}) \arrow[l] \arrow[u] \\
 & \mathscr{L}(\text{VPDA}) \arrow[u] \arrow[ul] &
\end{tikzcd}$$

Fig. 1. Inclusion structure of deterministic language families. The arrows indicate strict inclusions. All families not linked by a path are pairwise incomparable.

Proof. The inclusions of the language families induced by the input-driven stack automata are clear by definition. Moreover, $\mathscr{L}(\text{VPDA}) \subseteq \text{CFL}$ is immediate and $\mathscr{L}(\text{VPDA}) \subseteq \mathscr{L}(\text{1sDVSA})$ follows since every visibly (deterministic) pushdown automaton is also a deterministic strong input-driven stack automaton (that is not allowed to *read* the internal contents of the stack). First, we show that these inclusions are strict. The strictness of the inclusion $\mathscr{L}(\text{1sDVSA}) \subseteq \mathscr{L}(\text{1wDVSA})$ as well as the inclusion $\mathscr{L}(\text{1sNEDVSA}) \subseteq \mathscr{L}(\text{1wNEDVSA})$ is ensured by language L_1 of Lemma 3. Furthermore, a language similar to L_1 was used in [1] to show that the inclusion $\mathscr{L}(\text{VPDA}) \subseteq \text{CFL}$ is proper. Finally, the strictness of both inclusions $\mathscr{L}(\text{1sNEDVSA}) \subseteq \mathscr{L}(\text{1sDVSA})$ and $\mathscr{L}(\text{1wNEDVSA}) \subseteq \mathscr{L}(\text{1wDVSA})$ follows by language L_4 of Lemma 6. Next, we show the incomparability results. The languages L_2 and L_3 from Lemmata 4 and 5 imply the incomparability of CFL with the all language families of deterministic variants of input-driven stack automata, namely $\mathscr{L}(\text{1sNEDVSA})$, $\mathscr{L}(\text{1wNEDVSA})$, $\mathscr{L}(\text{1sDVSA})$, and $\mathscr{L}(\text{1wDVSA})$. The incomparability of $\mathscr{L}(\text{VPDA})$ with both language families $\mathscr{L}(\text{1sNEDVSA})$ and $\mathscr{L}(\text{1wNEDVSA})$ follows by the languages L_2 from Lemma 4 and L_4 from Lemma 6, and the obvious fact that L_4 is a visibly pushdown language. Finally, $\mathscr{L}(\text{1sDVSA})$ and $\mathscr{L}(\text{1wNEDVSA})$ are incomparable due to the languages L_1 and L_4 from Lemmata 3 and 6. This proves the stated claim. $\square$

In the remainder of this section we investigate the relation between deterministic and nondeterministic input-driven stack automata. To this end consider the language $L_5 = T_1 \cup T_2 \cup T_1' \cup T_2' \cup T_3 \cup T_4$ with

$$T_1 = \{\, a^n d_1^n u_1^{n+1} \mid n \geq 1 \,\}$$
$$T_2 = \{\, a^n d_2^n u_2^{n+1} \mid n \geq 1 \,\}$$
$$T_1' = \{\, a^n u_1^m d_1^n u_1^{n+1} \mid m, n \geq 1 \,\}$$
$$T_2' = \{\, a^n u_2^m d_2^n u_2^{n+1} \mid m, n \geq 1 \,\}$$
$$T_3 = \{\, a^n d_1^{n+1} \mid n \geq 1 \,\}$$

and

$$T_4 = \{\, a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w \hat{w}^R d_1^m d_2^{n+1} \mid m, n \geq 1, w \in \{0,1\}^+ \,\},$$

where $\hat{\ }$ is the mapping introduced in Lemma 6. Then we can prove the following result, which allows us to categorize the different symbols used in the definition of language L_5.

Lemma 8. *Let $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d\rangle$ be a 1wDVSA.*

1. *If T_1 is accepted by M, then $d_1 \notin \Sigma_c \cup \Sigma_r$.*
2. *If T_2 is accepted by M, then $d_2 \notin \Sigma_c \cup \Sigma_r$.*
3. *If T'_1 is accepted by M, then $u_1 \notin \Sigma_c \cup \Sigma_r$.*
4. *If T'_2 is accepted by M, then $u_2 \notin \Sigma_c \cup \Sigma_r$.*

Proof. We only prove the first claim. The other claims can be shown by similar arguments. So assume that T_1 is accepted by M. In order to show $d_1 \notin \Sigma_c \cup \Sigma_r$, we assume in contrast to the assertion that d_1 is in Σ_c or Σ_r. Thus, we distinguish two cases—note that we make no assumption on the containment of the letter a within the sub-alphabets that come from the partition of Σ:

1. Assume that $d_1 \in \Sigma_r$. Note that after reading the word $a^n d_1^n$ the stack of M is empty up to the bottom of stack symbol $\perp$. This is due to the fact, that M is deterministic and $d_1 \in \Sigma_r$. Since Q is finite, there are two distinct sufficiently large numbers $n_1 \neq n_2$ and a state $q \in Q$, such that
$$(q_0, a^{n_1} d_1^{n_1} u_1^{n_1+1}, \perp, 1) \vdash^* (q, u_1^{n_1+1}, \perp, 1) \vdash^* (f_1, \lambda, s_1, p_1)$$
and
$$(q_0, a^{n_2} d_1^{n_2} u_1^{n_2+1}, \perp, 1) \vdash^* (q, u_1^{n_2+1}, \perp, 1) \vdash^* (f_2, \lambda, s_2, p_2),$$
for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma' \Gamma^* \perp \cup \{\perp\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. This implies
$$(q_0, a^{n_1} d_1^{n_1} u_1^{n_2+1}, \perp, 1) \vdash^* (q, u_1^{n_2+1}, \perp, 1) \vdash^* (f_2, \lambda, s_2, p_2)$$
and, thus, $a^{n_1} d_1^{n_1} u_1^{n_2+1} \in T_1$, which is a contradiction.
2. Now suppose that $d_1 \in \Sigma_c$. Then we argue as follows. Since M is deterministic and $d_1 \in \Sigma_c$, we know that the stack height is at least $n+1$ after reading word $a^n d_1^n$ and the stack pointer is on the topmost symbol. Then further reading of u_1^{n+1}—here we make no assumption on letter u_1 and its containment in Σ_c Σ_r, Σ_i, Σ_u, or Σ_d—may only touch the topmost $n+1$ symbols of the stack. Since Q and Γ are finite we find two sufficiently large numbers $n_1 \neq n_2$, a state $q \in Q$ and a stack symbol $Z \in \Gamma$, such that
$$(q_0, a^{n_1} d_1^{n_1} u_1^{n_1+1}, \perp, 1) \vdash^* (q, d_1^{n_1} u_1^{n_1+1}, Z'\gamma_1, |\gamma_1| + 1) \vdash^* (f_1, \lambda, s_1, p_1)$$
and
$$(q_0, a^{n_2} d_1^{n_2} u_1^{n_2+1}, \perp, 1) \vdash^* (q, d_1^{n_2} u_1^{n_2+1}, Z'\gamma_2, |\gamma_2| + 1) \vdash^* (f_2, \lambda, s_2, p_2),$$
for $\gamma_i = \lambda$, if $Z = \perp$, and $\gamma_i \in \Gamma^* \perp$, if $Z \neq \perp$, for $i = 1, 2$, and $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma' \Gamma^* \perp \cup \{\perp\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. Since both topmost stack symbols after processing a^{n_1} and a^{n_2}, respectively, are identical, and

the stack contents below that particular symbol is never touched while processing the remaining part $d_1^{n_1}u_1^{n_1+1}$ and $d_1^{n_2}u_1^{n_2+1}$, respectively, we obtain the accepting computation

$$(q_0, a^{n_1}d_1^{n_2}u_1^{n_2+1}, \bot, 1) \vdash^* (q, d_1^{n_2}u_1^{n_2+1}, Z'\gamma_1, |\gamma_1|+1) \vdash^* (f_2, \lambda, s_3, p_3),$$

for some $s_3 \in \Gamma'\Gamma^*\bot \cup \{\bot\}$ with $|s_3| \geq |\gamma_1|$, and $p_3 \geq |\gamma_1|$. This implies that the word $a^{n_1}d_1^{n_2}u_1^{n_2+1} \in T_1$, which is a contradiction. □

Next we need some notation. Let Σ_c, Σ_r, Σ_i, Σ_u, and Σ_d be a partitioning of Σ. Then we say that a partitioning Σ'_c, Σ'_r, Σ'_i, Σ'_u, and Σ'_d of $\Sigma' \subseteq \Sigma$ is *compatible with* the partitioning of Σ, if $\Sigma'_c \subseteq \Sigma_c$, $\Sigma'_r \subseteq \Sigma_r$, $\Sigma'_i \subseteq \Sigma_i$, $\Sigma'_u \subseteq \Sigma_u$, and $\Sigma'_d \subseteq \Sigma_d$. Then the next lemma, which can be shown by an easy adaptation of the well-known Cartesian product construction for pushdown automata, reads as follows:

Lemma 9. *Let $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d\rangle$ be a 1wDVSA (1sDVSA) and $R \subseteq \Sigma^*$ be a regular language. Then the language $L(M) \cap R$ is accepted by a 1wDVSA (1sDVSA) with a compatible partitioning of the alphabet Σ.* □

Now we are ready to show that there exists a language that belongs to the class induced by the most restricted form of nondeterministic input-driven automata, namely 1sNENVSA, but is not a member of the larger deterministic class 1wDVSA.

Lemma 10. *$L_5 \in \mathscr{L}$(1sNENVSA) $\setminus \mathscr{L}$(1wDVSA).*

Proof. The idea for a 1sNENVSA accepting L_5 is first to guess whether the input belongs to T_1, T_2, T'_1, T'_2, T_3, or T_4. The construction to accept inputs from T_1, T_2, T'_1, T'_2, and T_3 is similar to the construction in Lemma 4. In the constructions for T'_1 and T'_2 we observe that u_1 and u_2 belong to Σ_u. Since the stack pointer is on the top after reading a^n, the processing of u_1 and u_2 only affects the current state, but not the stack or the position of the stack pointer. The construction to accept T_4 is as follows. While reading a's some symbol A is pushed onto the stack up to some moment in which it is nondeterministically decided to push some different symbol B onto the stack for every remaining input symbol a. Then, while reading d_1's and seeing B's the stack pointer moves down and continues moving down while reading d_2's and seeing A's. If the bottom of the stack is reached, the stack pointer moves up while reading u_2's and seeing A's and continues moving up, while reading u_1's and seeing B's on the stack. If the top of the stack is reached, the processing of the infix $w\hat{w}^R$ is done in a similar way as in the proof of Lemma 6. The only difference is that symbols from $\{\hat{0}, \hat{1}\}$ now ensure that the stack pointer moves down instead of popping off the topmost symbol. If the first B of the stack is again reached, the stack pointer continues moving down while reading d_1's and seeing B's and continues to move down while reading d_2's and seeing A's. The input is accepted if the bottom of the stack is eventually reached and rejected otherwise. We observe from the constructions that $a, 0, 1 \in \Sigma_c$, $d_1, d_2, \hat{0}, \hat{1} \in \Sigma_d$, and $u_1, u_2 \in \Sigma_u$. Thus, the automaton constructed is non-erasing and works in the strong mode.

Next, we show by way of contradiction that $L_5 \notin \mathscr{L}(\text{1wDVSA})$. Assume that L_5 is accepted by a 1wDVSA $M = \langle Q, \Sigma, \Gamma, \perp, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similar as in the proof of Lemma 3, we conclude that $a \in \Sigma_c$. Now, we assume that $d_1 \in \Sigma_c \cup \Sigma_r$. Due to Lemma 9 with $R = a^+ d_1^+ u_1^+$ we obtain that T_1 is accepted by some 1wDVSA having $d_1 \in \Sigma_c \cup \Sigma_r$. This is a contradiction to Lemma 8. Thus, $d_1 \notin \Sigma_c \cup \Sigma_r$. Similarly, it can be shown that also $d_2, u_1, u_2 \notin \Sigma_c \cup \Sigma_r$. Finally, we claim that $d_1 \in \Sigma_d$. Otherwise, due to Lemma 9 with $R = a^+ d_1^+$, language T_3 would be accepted by some 1wDVSA having $d_1 \in \Sigma_i \cup \Sigma_u$ which is a contradiction.

The rest of the proof is similar to the proof of Lemma 6 and we leave out some details here. We consider an accepting computation K on an input of the form $z = a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w \hat{w}^R d_1^m d_2^n$, for $w \in \{0,1\}^+$, $n \geq 2$, $k = \lfloor \log(n)) \rfloor$, and $m = 2^{2^k} - n$. Due to the above considerations, we know that nothing is pushed onto or popped off the stack while reading the infix $d_1^m d_2^n u_2^{n+1} u_1^m$. We distinguish now two cases.

1. First, we assume that nothing is pushed onto or popped off the stack while reading the infix w, and consider the configuration $c = (q, \hat{w}^R d_1^m d_2^n, s, p)$ with $s \in \Gamma' \Gamma^* \perp$ and $1 \leq p \leq |s|$ after reading $a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w$. Then, the knowledge of $M, n, m, |w|, q$, and p is sufficient to write a program that outputs w. The size of this program is bounded by $O(\log(n) + \log(m) + \log(n+m) + \log(|w|))$. Hence, $C(w||w|) \in o(|w|)$ which is a contradiction for $|w|$ large enough.
2. Second, we assume that something is pushed onto or popped off the stack while reading the infix w. We consider the configuration $c = (q, x, s, |s|)$ with $s \in \Gamma' \Gamma^* \perp$ and $x \in \{0,1\}^* \{\hat{0}, \hat{1}\}^+ d_1^+ d_2^+$ being the remaining input when the first symbol has been pushed onto or popped off the stack while reading w. Again, the knowledge of M, k, $|w|$, the length ℓ of the prefix w' of w which has been already read, q, and the last pushed or popped symbol $B \in \Gamma'$ is sufficient to write a program that outputs n. The size of this program is bounded by $O(\log(k) + \log(|w|)) = O(\log(\log(n)) + \log(|w|))$. Thus, $C(n) \in o(\log(n))$ which is a contradiction since there are infinitely natural numbers n such that $C(n) \geq \log(n)$.

Hence, there cannot be any 1wDVSA that accepts the language L_5. □

As an immediate corollary of the previous lemma we obtain the following strict inclusions.

Corollary 11. *1. $\mathscr{L}(1sNEDVSA) \subset \mathscr{L}(1sNENVSA)$.*
2. $\mathscr{L}(1sDVSA) \subset \mathscr{L}(1sNVSA)$.
3. $\mathscr{L}(1wNEDVSA) \subset \mathscr{L}(1wNENVSA)$.
4. $\mathscr{L}(1wDVSA) \subset \mathscr{L}(1wNVSA)$. □

These strict inclusions show a large difference between input-driven pushdown automata languages or equivalently visibly pushdown languages, where determinism coincides with nondeterminism on the underlying automaton model. For input-driven stack automata, even for the most restrictive version, the strong machines, nondeterminism is more powerful than determinism. These strictness results are also reflected in the forthcoming result on the complexity of the fixed membership problem. Since the family of languages accepted by ordinary deterministic stack automata belongs to deterministic polynomial time P [13] it follows that this is also true for every language family induced by a deterministic input-driven stack automaton, regardless whether we have a strong or weak machine, or whether the device is non-erasing or not. On the other hand, when changing from ordinary deterministic stack automata to nondeterministic ones, we obtain an NP-complete language family [11,19]. This is also the case for the nondeterministic versions of input-driven stack automata, even for strong non-erasing machines, which is shown next.

Theorem 12. *Each of the language families $\mathscr{L}$(1sNENVSA), $\mathscr{L}$(1sNVSA), $\mathscr{L}$(1wNENVSA), and $\mathscr{L}$(1wNVSA) has an NP-complete fixed membership problem.*

Proof. The containment in NP follows immediately from the fact that ordinary nondeterministic stack automata have an NP-complete fixed membership problem [11,19]. It remains to show NP-hardness. To this end, it suffices to show that the (with respect to set inclusion) smallest language family $\mathscr{L}$(1sNENVSA) has an NP-hard membership problem. We reduce the well-known NP-complete 3SAT problem [7] to the problem under consideration. We encode a Boolean formula $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ with variables $X = \{x_1, x_2, \ldots, x_n\}$, where each clause c_i with $1 \leq i \leq m$ is a disjunction of three literals, by a word $\langle F \rangle \in 1^n(\{0,+,-\}^*\#^*\$)^*$. The prefix encodes the number of input variables. Then each clause c_i, for $1 \leq i \leq m$, of the formula F is encoded by a word w_{c_i} in $\{0,+,-\}^n\#^n$, where at position j, for $1 \leq j \leq n$, there is a 0 (+, −), if the variable x_j does not appear (positively appears, negatively appears) in c_i. These words are separated by \$-symbols and are placed in sequence. Then the language

$$L = \{\, \langle F \rangle \mid F \text{ is a Boolean formula that evaluates to } 1 \,\}$$

is NP-hard. We informally describe how a 1wNENVSA automaton M accepts L. Set $\Sigma_c = \{1\}$, $\Sigma_r = \emptyset$, $\Sigma_i = \{\$\}$, $\Sigma_u = \{\#\}$, and $\Sigma_d = \{0,+,-\}$. On prefix 1^n the automaton pushes either the symbol 0 or 1. In this way, the automaton guesses an assignment to the n variables of the formula. Then the sequence that encodes a clause is used to read into the stack in order to determine the assignments of the involved variables. In passing the automaton checks whether this clause evaluates to 1. Then the block of # symbols is used to reset the stack pointer to the top of the stack, and after reading \$ the checking procedure for the next clause is restarted. If all clauses evaluate to 1, the whole encoding is accepted, otherwise it is rejected. It is easy to see that the automaton is non-erasing. This shows that already the language family $\mathscr{L}$(1sNENVSA) contains an NP-hard language. □

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56, Article 16 (2009)
2. von Braunmühl, B., Verbeek, R.: Input-driven Languages are Recognized in $\log n$ Space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983)
3. Chervet, P., Walukiewicz, I.: Minimizing Variants of Visibly Pushdown Automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 135–146. Springer, Heidelberg (2007)
4. Chomsky, N.: Formal Properties of Grammars. In: Handbook of Mathematic Psychology, vol. 2, pp. 323–418. Wiley & Sons (1962)
5. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 214–226. Springer, Heidelberg (2010)
6. Dymond, P.W.: Input-driven languages are in $\log n$ depth. Inform. Process. Lett. 26, 247–250 (1988)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. Freeman (1979)
8. Ginsburg, S.: Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland (1975)
9. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. J. ACM 14, 389–418 (1967)
10. Han, Y.S., Salomaa, K.: Nondeterministic state complexity of nested word automata. Theoret. Comput. Sci. 410, 2961–2971 (2009)
11. Hunt, H.: On the complexity of finite, pushdown and stack automata. Math. Systems Theory 10, 33–52 (1976)
12. Kutrib, M., Malcher, A.: Reversible Pushdown Automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 368–379. Springer, Heidelberg (2010)
13. Lange, K.J.: A note on the P-completeness of deterministic one-way stack languages. J. Univ. Comput. Sci. 16, 795–799 (2010)
14. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and its Applications. Springer (1993)
15. Mehlhorn, K.: Pebbling Mountain Ranges and Its Application of DCFL-recongnition. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980)
16. Okhotin, A., Salomaa, K.: State Complexity of Operations on Input-Driven Pushdown Automata. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 485–496. Springer, Heidelberg (2011)
17. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theoret. Comput. Sci. 410, 3290–3302 (2009)
18. Salomaa, A.: Formal Languages. ACM Monograph Series. Academic Press (1973)
19. Shamir, E., Beeri, C.: Checking Stacks and Context-free Programmed Grammars Accept P-complete Languages. In: Loeckx, J. (ed.) ICALP 1974. LNCS, vol. 14, pp. 27–33. Springer, Heidelberg (1974)
20. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. J. ACM 25, 405–414 (1978)

Probabilistic Inference and Monadic Second Order Logic

Marijke Hans L. Bodlaender

P.O. Box 80.089, Department of Computing Sciences, Utrecht University, The Netherlands
h.l.bodlaender@uu.nl

Abstract. This paper combines two classic results from two different fields: the result by Lauritzen and Spiegelhalter [21] that the probabilistic inference problem on probabilistic networks can be solved in linear time on networks with a moralization of bounded treewidth, and the result by Courcelle [10] that problems that can be formulated in counting monadic second order logic can be solved in linear time on graphs of bounded treewidth. It is shown that, given a probabilistic network whose moralization has bounded treewidth and a property P of the network and the values of the variables that can be formulated in counting monadic second order logic, one can determine in linear time the probability that P holds.

1 Introduction

This paper combines two classic results from two different fields from computer science: the result by Lauritzen and Spiegelhalter [21] that the probabilistic inference problem can be solved in linear time for probabilistic network whose moral graph has treewidth bounded by some constant, and the result by Courcelle [10] that problems that can be formulated in counting monadic second order logic can be solved in linear time on graphs of bounded treewidth.

Probabilistic networks (also called *Bayesian networks* or *belief networks*) are the underlying technology of several modern decision support systems. See for more background, e.g., [16,22]. A probabilistic network consists of a directed acyclic graph, and for each vertex in the graph a table of conditional probabilities. Each vertex in the graph represents a statistical variable, which can assume one of a fixed number of values; the table for a vertex gives the probability distribution for the values for that vertex, conditional on the values of the parents of the vertex.

Probabilistic networks are a topic of intense study. A central problem for probabilistic networks is the *inference problem.* One of the most commonly used algorithm to solve this problem is the *clique tree propagation* algorithm of Lauritzen and Spiegelhalter [21]. It consists of the following steps: first the *moral graph* or *moralization* is formed from the probabilistic network. (See Section 2 for the definition of moral graph.) To this moral graph, edges are added such that a triangulated (or chordal) graph is formed. To a triangulated graph G, one can associate a *clique tree*: a tree with every tree node associated to a unique maximal clique in G, with the additional property that for each node of the graph, the cliques to which it belong form a connected subtree of the tree. The clique tree can be used to solve the inference problem, with the time exponential in the sizes of the cliques, but linear when these sizes are bounded by a constant. An alternative way of describing the algorithm of [21] is by means of *tree decompositions*. Doing so, we have an algorithm for the probabilistic inference problem that is linear in the number of variables, but exponential in the *width* of the tree decomposition, in other words: the problem is linear time solvable when the moral graph has its *treewidth* bounded by a constant.

There are many other problems with the property that they are intractable (e.g., NP-hard) for general graphs, but become linear time solvable on graphs with bounded treewidth. A very powerful characterization of a class of such problems is by [10] by the notion of *Monadic Second Order Logic*, or extensions of this notion. In this paper, we use the notion of *Counting Monadic Second Order Logic*and the notion of *regularity*. The result of Courcelle has been extended a number of times (e.g., [1,8,12], see also [9].)

In this paper, we show that the results by Lauritzen and Spiegelhalter and by Courcelle can be combined. Suppose we have a probabilistic network whose moral graph has bounded treewidth. We can state some property of the network and of the values of the variables. Our result shows that if the property can be formulated in a language called (Counting) Monadic Second Order Logic, then we can compute the probability that it holds in linear time; i.e., to compute such a probability is *fixed parameter tractable* when parameterized by treewidth. (The result can also be seen as a variant of results shown by Courcelle and Mosbah [12].) Examples of such CMSOL properties are: do the variables that are true form a connected subgraph of the network, does no directed path in the network have all its variables alternatingly true and false along the path, are there an odd number of variables true, is the subgraph of the network induced by the false variables 3-colorable? It includes many properties that are NP-hard to check for arbitrary networks.

In Section 2, we give some preliminary results and definitions, including a hypergraph model that can be used instead of moral graphs. In Section 3, we sketch the proof of the main result. Some variants, extensions, and applications are discussed in Section 4. Some conclusions are given in Section 5.

2 Preliminaries

2.1 Probabilistic Networks and the Inference Problem

To ease descriptions, we assume all variables in the probabilistic networks that are dealt with to be binary (boolean). Notations in the paper sometimes follow conventions from algorithmic graph theory, and sometimes follow conventions from probabilistic networks theory.

For a set S, a *configuration on* S is a function $S \to \{\text{true, false}\}$. We denote the set of all configurations on S as $\mathcal{C}(S)$. A configuration on a set S is often denoted as x_S; when S is a single vertex v, we write x_v for $x_{\{v\}}$. $x_\emptyset$ denotes the unique configuration on the empty domain.

For sets S and S' and configurations $x_S \in \mathcal{C}(S)$ and $x_{S'} \in \mathcal{C}(S')$, we say that x_S *agrees with* $x_{S'}$, notated $x_S \sim x_{S'}$, if for all $v \in S \cap S' : x_S(v) = x_{S'}(v)$. Given a configuration $x_S \in \mathcal{C}(S)$ and a subset $S' \subseteq S$, we write $x_S(S')$ for the configuration on S' that is a restriction of x_S to the smaller domain: $x_S(S') \in \mathcal{C}(S')$ with $x_S(S') \sim x_S$.

Given a directed graph $G = (V, A)$ with a vertex $v \in V$, $par(v)$ denotes the set of vertices that have an arc to v: $par(v) = \{w \in V \mid \exists (w, v) \in A\}$. $par(v)$ is called the set of *parents* of v.

Let $G = (V, A)$ be a directed graph. The *moral graph*of G is the undirected graph, obtained by adding an edge between each pair of different vertices that are parents of the same vertex, and then dropping all directions of arcs, i.e., the graph $(V, \{\{v, w\} \mid (v, w) \in A \vee (w, v) \in A \vee \exists x \in V : v, w \in par(x)\}$. (The process of obtaining the moral graph of a directed graph is called *moralization*. The term, frequently used in probabilistic network research, comes from the notion to 'marry the parents'.)

A probabilistic network is a pair (G, κ), with $G = (V, A)$ a directed acyclic graph, and κ a set of functions, as follows. For each $v \in V$, we have a function $\kappa_v : \mathcal{C}(\{v\}) \times \mathcal{C}(par(v)) \rightarrow [0, 1]$.

κ_v is meant to describe the conditional probability distribution for v, here conditional on the values of the parents of v, i.e., for configurations x_v, $x_{par(v)}$, $\kappa_v(x_v, x_{par(v)})$ should give the value $Pr(x_v|x_{par(v)})$.

G and κ together define a joint probability distribution on the set of probabilistic variables V in the following way.

A *full configuration* of a probabilistic network (G, κ), $G = (V, A)$ is a configuration $x_V \in \mathcal{C}_V$ on the set V of all vertices in G.

For each full configuration, define the probability of the configuration as the product of the conditional probabilities over all vertices, as follows. Assume $x_V \in \mathcal{C}(V)$.

$$\Pr(x_V) = \prod_{v \in V} \kappa_v(x_V(\{v\}), x_V(par(v))) \tag{1}$$

Throughout this paper, we write x_V as the stochastic variable, that selects one full configuration with the distribution as given by (1) for the considered probabilistic network (G, κ).

For each configuration, its probability now is the sum of the probabilities of all full configurations that agree with it. Assume $x_W \in \mathcal{C}(W)$, $W \subseteq V$.

$$\Pr(x_W) = \sum_{x_V \in \mathcal{C}(V),\ x_V \sim x_W} \Pr(x_V) \tag{2}$$

In applications of probabilistic networks, one generally desires to know the probability that a variable has a certain value, conditional to given values for some other variables. For instance, in a network, modeling a medical application, one may want to know if a patient has a certain disease, given some symptoms. The given values of variables are called the *observations*. The set of *observed* *variables* is denoted by $\mathcal{O}$, and the observed configuration is denoted $x_{\mathcal{O}}$. The PROBABILISTIC INFERENCE problem is to compute the conditional probability $\Pr(x_v \mid x_{\mathcal{O}})$, for a variable $v \in V$ and $x_v \in \mathcal{C}(v)$, or, more generally, the probability distribution for each variable, conditional to $x_{\mathcal{O}}$. To compute $\Pr(x_v \mid x_{\mathcal{O}})$, we can use that $\Pr(x_v \mid x_{\mathcal{O}}) = \Pr(x_v \wedge x_{\mathcal{O}})/\Pr(x_{\mathcal{O}})$.

Computational Model. In the paper, we assume all computations with values of probabilities can be done in constant time. In a practical setting, one could work with standard rounded representations of real numbers. To be precise, we should assume each conditional probability in κ is given as a fraction of two integers. Analysis of the algorithms show that each value remains a fraction of two integers, both expressible with a polynomial number of bits; the algorithms we give involve a linear number of operations with such integers, while each such operation involves a polynomial number of bit operations. The details are not given here.

2.2 A Mixed Hypergraph Model

Instead of using the more standard method of using a moralization of a probabilistic network, we instead associate with each probabilistic network a mixed hypergraph.

A mixed hypergraph is a pair $H = (V, E, A)$, with V a finite set of *vertices*, and E a finite set of *hyperedges*, and A a set of arcs; each hyperedge $e \subseteq V$ is a non-empty subset of the vertices; an arc is an ordered pair of distinct vertices.

To a directed acyclic graph $G = (V, A)$, we associate a mixed hypergraph $H = (V, E, A)$ in the following way. H has the same vertices and arcs as G. For each vertex $v \in V$, we take a hyperedge consisting of v and its parents: $E = \{\{v\} \cup par(v) \mid v \in V\}$. H is the *mixed hypergraph associated with* G. The following lemma shows there is a one-to-one correspondence between the vertices and the hyperedges.

Lemma 1. *Let $G = (V, A)$ be a directed acyclic graph. If $v, w \in V$, $v \neq w$, then $\{v\} \cup par(v) \neq \{w\} \cup par(w)$.*

Besides the hypergraph model of the directed acyclic graph, we also define a set of functions $\kappa_e : \mathcal{C}(e) \to [0, 1]$, one for each hyperedge $e \in E$; these reflect the conditional probability functions κ_v.

Consider an edge $e = \{v\} \cup par(v)$. Let for each configuration on e, $x_e \in \mathcal{C}(e)$:

$$\kappa_e(x_e) = \kappa_v(x_e(\{v\}), x_e(par(v))) \tag{3}$$

We introduce one more notation. Let $x_S \in \mathcal{C}(S)$ be a configuration. Let e be a hyperedge with $e \subseteq S$. Then we write $\kappa(x_S, e) = \kappa(x_e)$ for the configuration x_e, obtained by restricting the function x_S to domain e.

We now can rephrase Equation 1 in terms of the κ_e functions. The next (well known) proposition follows directly from the definitions. See also [25].

Proposition 1. *Let $(G = (V, A), \kappa)$ be a probabilistic network. For each full configuration $x_V \in \mathcal{C}(V)$:*

$$\Pr(x_V) = \prod_{e \in A} \kappa_e(x_V)$$

2.3 Monadic Second Order Logic

The Monadic Second Order Logic language (MSOL) allows us to express properties of (mixed hyper)graphs. The language has the following constructs:

- Quantification over vertices, (hyper)edges, arcs: $\exists v \in V$, $\exists e \in E$, $\forall v \in V$, $\forall e \in E$, $\forall a \in A$,
- Quantification over sets of vertices, sets of (hyper)edges, sets of arcs: $\exists W \subseteq V$, $\exists F \subseteq E$, $\forall W \subseteq V$, $\forall F \subseteq E$, ...
- Membership tests: $v \in W$, $e \in F$.
- Identity tests: $v = w$, $e = f$.
- Adjacency tests: $v \in e$, $\{v, w\} \in E$, $\{v, w\} \in F$, v is tail (head) of a, ...
- Logic operations: $\vee$, $\wedge$, $\Rightarrow$, $\neg$, ...

MSOL is a powerful language. Many graph properties can be expressed in it. E.g, Borie et al. [8] show how many graph properties can be expressed in MSOL. For example, the following property expresses that directed graph $G = (V, A)$ is acyclic:

$$\forall W \subseteq V : \exists v \in W : \neg \exists w \in W : (v, w) \in A$$

Extensions of MSOL can include the following language constructs:

- For fixed constants c_1, c_2: $|W| \bmod c_1 = c_2$, $|F| \bmod c_1 = c_2$. MSOL with these constructs, for all fixed c_1 and c_2 is called Counting MSOL, or CMSOL.
- If the vertices, edges, or arcs of G are labeled with a bounded number of different labels, L the labeling function, we have label tests: $L(v) = c$, $L(e) = e$.

To these, we add one more construct, for the case that G is a probabilistic network.

- Value tests for vertices: $x_v = \text{true}$, $x_v = \text{false}$. (Or: x_v, $\neg x_v$.)

Call the resulting language *CMSOL with value tests*. For example, one can write the property that an even number of variables is true as follows:

$$\exists W : (\forall v : v \in W \leftrightarrow x_v) \wedge (|W| \bmod 2) = 0$$

Given a property in CMSOL with value tests, we are interested in the probability that this property holds for a given probabilistic network.

2.4 Treewidth, Terminal Hypergraphs, and Parse Trees

The notion of treewidth was introduced by Robertson and Seymour in [23]. There are several equivalent notions known, e.g., the treewidth of a graph is exactly one larger than the minimum over all chordal supergraphs of the maximum clique size; and graphs of treewidth at most k are also known as partial k-trees. See [3] for an overview. The definition here is given in terms of mixed hypergraphs.

Definition 1. *A* tree decomposition *of a mixed hypergraph $H = (V, E, A)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of vertices (called* bags*), and T a tree, such that*

– $\bigcup_{i \in I} X_i = V$.
– *For each hyperedge $e \in E$ and each arc $(v, w) \in A$: there exists an $i \in I$ with $e \subseteq X_i$, or $\{v, w\} \subseteq X_i$, respectively.*
– *For all vertices $v \in V$: the set of nodes $\{i \in I \mid v \in X_i\}$ is connected in T.*

The width *of tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum width over all possible tree decompositions of G.*

Instead of using tree decompositions, we use an equivalent framework, where we use an 'algebra' on terminal mixed hypergraphs. A *k-terminal mixed hypergraph* is a 4-tuple $G = (V, E, A, (x_1, \ldots, x_k))$ with V a finite set of vertices, $E \subseteq \mathcal{P}(V)$ a set of hyper edges, $A \subseteq V \times V$ a set of arcs, and $(x_1, \ldots, x_k) \in V^k$ an ordered set of k vertices from V, called the *terminals*. A *terminal mixed hypergraph* is a k-terminal mixed hypergraph for some $k \geq 0$.

We define the following operations on terminal mixed hypergraphs.

– $\textsc{CreateHyperedge}_k()$. Has no arguments. Yields a k-terminal hypergraph with k vertices and one hyperedge, containing all vertices: $(\{v_1, \ldots, v_k\}$, $\{\{v_1, \ldots, v_k\}\}$, $\emptyset$, $(v_1, \ldots, v_k))$.
– $\textsc{CreateArc}()$. As $\textsc{CreateHyperedge}$, but creates a directed arc: $(\{v_1, v_2\}$, $\emptyset$, $\{(v_1, v_2)\}, (v_1, v_2))$.
– $\textsc{DropTerminal}_{k,\ell}(G)$. Has a k-terminal mixed hypergraph as argument, and turns the ℓth terminal into a non-terminal: $(V, E, A, (x_1, \ldots, x_k))$ maps to $(V, E, A, (x_1, \ldots, x_{\ell-1}, x_{\ell+1}, \ldots, x_k))$.
– $\textsc{AddTerminal}_k(G)$. Has a $(k-1)$-terminal mixed hypergraph as argument, and adds a new terminal vertex. Hyperedges and arcs are not affected. So, $(V, E, A, (x_1, \ldots, x_{k-1}))$ maps to $(V \cup \{x_k\}, E, A, (x_1, \ldots, x_k))$.
– $\textsc{Join}_k(G, H)$. Has two k-terminal mixed hypergraphs as argument. Yields the k-terminal mixed hypergraph obtained by taking the union of the arguments and then identifying the corresponding terminals. So, $\textsc{Join}_k((V, E, A, (x_1, \ldots, x_k)), (V', A', E', (x_1, \ldots, x_k))) = (V \cup V', E \cup E', A \cup A', (x_1, \ldots, x_k))$, with $V \cap V' = \{x_1, \ldots, x_k\}$.

Let $\mathcal{O}_k$ be the set of operations, containing for all k', ℓ, $1 \leq \ell \leq k' \leq k$, the operations $\textsc{CreateHyperedge}_{k'}$, $\textsc{CreateArc}_{k'}$, $\textsc{DropTerminal}_{k',\ell}$, $\textsc{AddTerminal}_{k'}$, $\textsc{Join}_{k'}$.

Lemma 2. *Let $(G = (V, A), \kappa)$ be a probabilistic network, let H be the moral graph of G, and let $H' = (V, E', A)$ be the associated mixed hypergraph of G. The following statements are equivalent.*

1. *H has treewidth at most $k - 1$.*
2. *H is subgraph of a triangulated graph with maximum clique size k.*
3. *H' has treewidth at most $k - 1$.*
4. *The 0-terminal mixed hypergraph $(V, E', A, \emptyset)$ can be constructed using the operations from $\mathcal{O}_k$.*

This can be shown using proofs and techniques from [26], see also [3]. Note that the notion is a minor twist on the well known notion of *nice tree decompositions*, see e.g., [3,19].

If we have a method to construct a terminal mixed hypergraph with operations from $\mathcal{O}_k$, we can express this in a *parse tree*. Each node of such a tree is labeled with one operation from $\mathcal{O}_k$. To each tree node i, we associate a terminal mixed hypergraph $G_i = (V_i, E_i, A_i, (x^i_1, \ldots, x^i_{k'})$; the terminal mixed hypergraph associated to the node is the graph obtained by applying its operation to the terminal mixed hypergraphs associated with its children. We assume that the root r of the parse tree is a 0-terminal mixed hypergraph, otherwise, use some DropTerminal operations at the top of the tree. If $G_r = (V_r, E_r, A_r, \emptyset)$, (V_r, E_r, A_r) is the mixed terminal hypergraph represented by the parse tree.

It is well known (see e.g., [6]) that if W is a clique in G, then a tree decomposition of G has a bag that contains all vertices in W. .Note that for each vertex v, $v \cup par(v)$ forms a clique in the moral graph and thus, for each tree decomposition of the moral graph, there is a bag that contains $v \cup par(v)$.

2.5 Regular Properties and Finite State Tree Automata

Many of the linear time algorithms on graphs of bounded treewidth can be expressed as a *finite state tree automaton*, a generalization of the classic finite state automaton. Such an automaton can be written as a 4-tuple $(S, S_A, t, \Box)$, with S a finite set of *states*, $S_A \subseteq S$ a set of *accepting states*, $\Box$ a special symbol, and t a *state transition function*: $t : (S \cup \{\Box\}) \times (S \cup \{\Box\}) \times \mathcal{O}_k \rightarrow S$. The automaton works on a binary tree with nodes labeled with elements from $\mathcal{O}_k$ (and in particular: on a parse tree), in the following way. Each element i of the tree is associated a state $s(i)$ with $s(i) = t(s_L, s_R, o(i))$, with s_L (s_R) the state $s(j)$ of the left (right) child j of i, $s(i) = \Box$ if i has no left (right) child, and $o(i)$ the operation from $\mathcal{O}_k$ with which i is labeled. Note that these states can be computed in bottom-up order (e.g., in postorder) in the tree. We say an automaton *accepts* the labeled tree, if the state of the root belongs to the set S_A.

A property P of graphs (or, of mixed hypergraphs) is *regular*, if for each k, there is a finite state tree automaton M_k, such that for each G: $P(G)$ holds, if and only if M_k accepts all parse trees of G with operations from $\mathcal{O}_k$, if and only if M_k accepts at least one such parse tree. Courcelle's theorem can be stated as follows.

Theorem 1. *[10] A graph property in CMSOL is regular.*

Courcelle conjectured that all regular properties belong to CMSOL; for some special cases, the conjecture has been shown by Kabanets [17] and Kaller [18]. Regularity implies that for each k, there is a linear time algorithm that, given a (mixed hyper)graph G of treewidth at most k, decides if the property holds for G or not: the parse tree with operations in $\mathcal{O}_k$ can be constructed in linear time, and then the automaton M is run in linear time on the parse tree; decide 'yes', if M ends in an accepting state.

We need to look at a minor variant of Theorem 1 for graph expressions with one free vertex set variable, i.e., properties of the form $P(G,W)$, $G=(V,E,A)$ a mixed hyper graph, $W \subseteq V$ a set of vertices. Our finite state tree automaton gets as input a labeled tree, *and* for each node in the tree that gives a new vertex (i.e., one that is not used by any node that is a descendant in the tree) whether this vertex belongs to the vertex set we denote by W. The state transition function gets a fourth argument, that is empty, except for the AddTerminal, CreateArc, and CreateHyperedge$_{k'}$ operations, where it gets a series of 1, 2, or k' booleans telling whether the new vertices belong to set W or not.

Theorem 2. *[10] Let P be an expression in CMSOL with free set variable W. Then P is regular.*

We write as shorthand notation: $\{x_V\} = \{v \in V \mid x_V(v) = \text{true}\}$, i.e., the set of variables with value true in configuration x_V. For a probabilistic network (G,κ) and (regular) property P with free vertex set variable W, we can ask for the probability that $P(G, x_V\}$ holds. Examples of such properties are given in the introduction. We can write:

$$\Pr(P(G,\{x_V\}) = \sum_{W\subseteq V,\ P(G,W)} \Pr(x_V^W) \tag{4}$$

with x_V^W the full configuration with $\{x_V\} = W$. This generalizes (2).

3 Main Results

The main result of this paper is given now. It shows that we can combine the results of Courcelle (Theorem 2) and [21]. Note that the treewidth of the moral graph of G is at least the treewidth of G.

Theorem 3. *Let P be a property of (mixed hyper)graphs with one free vertex set variable. Suppose P is regular. For each constant k, there is a linear time algorithm that, when given a probabilistic network (G,κ) with the treewidth of the moral graph of G at most k, computes the probability of $P(G,\{x_V\}\}$.*

Proof. Let k be fixed. Assume we have the finite state tree automaton $M = (S, S_A, t, \Box)$ for P and $\mathcal{O}_{k+1}$. We first build in linear time a parse tree T with operations in $\mathcal{O}_{k+1}$ for the hypergraph associated with (G,κ). Then, we compute for each node in T a table called Q_i. These tables can be computed in bottom up order in the tree. Given the table of the root node, the requested value can be computed.

For each node i in T, we denote its associated terminal graph $G_i = (V_i, E_i, A_i, (x^i_1, \ldots, x^i_{k_i}))$, and write $X_i = \{x^i_1, \ldots, x^i_{k_i}\}$. Q_i maps each pair (x_{X_i}, s) to a real value in the interval $[0, 1]$, with x_{X_i} a configuration on X_i, and $s \in S$ a state, such that

$$Q_i(C_{X_i}, s) = \sum \prod_{e \in E_i} \kappa(x_{V_i,e})$$

where we sum over all configurations x_{V_i} on V_i that agree with x_{X_i} and that have the property that when we run machine M on the subtree with root i with $W = \{x_{V_i}\}$ then M gives state s in i. Tables have size $2^{|X_i|} \cdot |S| \leq 2^{k+1} \cdot |S| = O(1)$.

Claim. Let i be a node in the parse tree. Given tables Q_j for all children j of i, we can compute the table Q_i in $O(1)$ time.

Proof. i is labeled with some operation from $\mathcal{O}_{k+1}$. For each type of operation, we must show the claim separately. We consider two more interesting cases, and omit the others here.

Join. Suppose i is labeled $\textsc{Join}_{k'}$ and has children j_1 and j_2. $X_i = X_{j_1} = X_{j_2} = V_{j_1} \cap V_{j_2}$. For each configuration x_{X_i} and state s:

$$Q_i(x_{X_i}, s) = \sum Q_{j_1}(x_{X_i}, s_1) \cdot Q_{j_2}(x_{X_i}, s_2) \tag{5}$$

where the sum is taken over all pairs (s_1, s_2) with $t(s_1, s_2, \textsc{Join}_{k'}) = s$. There are $O(1)$ such pairs, and thus each of the $O(1)$ values $Q_i(x_{X_i}, s)$ can be computed in $O(1)$ time.

AddTerminal. Suppose j is the child of node i, i is labeled $\textsc{AddTerminal}'_k$, with z the added new vertex. Consider a configuration x_{X_i} and a state s. We look at the case that z has value *true* in x_{X_i}, the other case is similar. Now, one can show

$$Q_i(x_{X_i}, s) = \sum Q_j(x_{X_i - \{z\}}, s') \tag{6}$$

where $x_{X_i - \{z\}}$ is the restriction of x_{X_i} to domain $X_i - \{z\}$, and the sum is taken over all $s' \in S$ with $t(s, \square, \textsc{AddTerminal}'_k) = s$. This implies that Q_i can be computed in $O(1)$ time, given Q_j. □

Claim. Let r be the root of the parse tree of G.

$$\Pr(P(G, \{v \mid v = \text{true}\})) = \sum_{s \in S_A} Q_r(x_\emptyset, s)$$

Proof. G_r has 0 terminals, so there is a unique configuration $x_{X_r} = x_\emptyset$.

$\Pr(P(G, \{x_V\}))$ equals

$$\sum \Pr(x_V) = \sum \prod_{e \in E} \kappa(x_{V_i}, e)$$

where the sum is taken over all configurations x_V on $V = V_r$ where $P(G, \{x_V\})$ holds. Each such configuration trivially agrees with $x_\emptyset$. As M is the finite state tree automaton for P, these configurations are exactly those where M accepts when $W = \{x_V\}$. So, the sum equals

$$\sum_{s \in S_A} \sum \prod_{e \in E} \kappa(x_{V_i}, e)$$

where the second sum is taken over the configurations x_V on V that have M end in state s when W is as above. This equals $\sum_{s \in S_A} Q_r(x_\emptyset, s)$. □

So, working bottom-up in the parse tree, we can compute in linear time all tables Q_i, and then compute the requested answer in $O(1)$ time using the table of the root of the parse tree. □

4 Extensions

We briefly mention a few extensions of the result.

4.1 Non-binary Variables

While most results were described in terms of binary variables, the same results hold when variables are non-binary but can attain a number of values bounded by some constant.

4.2 Conditional Probabilities

Suppose we want to compute the probability that property P holds, conditional to the observations $x_\mathcal{O}$. The value can be computed with the same method. We use that $\Pr(P(G, x_V)|x_\mathcal{O}) = \Pr(P(G, x_V) \wedge x_\mathcal{O}) / \Pr(x_\mathcal{O})$ and that regularity and CMSOL are closed under conjunction, and compute $\Pr(P(G, x_V) \wedge x_\mathcal{O})$ and $\Pr(x_\mathcal{O})$ separately.

4.3 Different Types of Edges

To the mixed hypergraph model, we can add additional edges and arcs that are not part of the probabilistic network, but can be used in CMSOL queries on the network. Assuming that we work with bounded width tree decompositions that also fulfill the property that for each of the different types of edges there are bags that contain the endpoints, we can also answer compute the probability of CMSOL queries on properties about the graph formed by these additional edges in linear time.

4.4 Different Models

Possibly, the results can also be applied to different probabilistic models, whose structure can be modeled by hypergraphs. See e.g. [13].

5 Conclusions

Examples. The results in this paper can be applied to a large number of problems, as many graph properties can be formulated in CMSOL. See e.g., [8] for many examples of CMSOL graph properties. Some examples are:

– 'Suggested causality': for two variables x and y: what is the probability that x and y hold *and* there is a path from x to y of variables that all are true?
– Independence: what is the probability that no true variables are adjacent?
– In graphical games: some vertices represent an agent. Agents have states, which are modeled by a probabilistic network. For a collection of pairs of agents F, we ask: what is the probability that no two pair in F of agents have the same state? Or: can we partition the agents into three (or any other constant) number of groups, such that adjacent agents with the same state belong to a different group?

Time and space considerations. The time and space used by the algorithm of Theorem 3 is approximately the product of the time, respectively the space, of the Lauritzen Spiegelhalter algorithm and the number of states of the tree automaton. In some cases this number of states is large, in particular, when the property whose probability is to be computed is NP-hard when the treewidth is not bounded. In some other cases, the number of states can be reasonably small. For instance, when P is the property that an even number of variables has the value true, then a machine with only two states suffices. So, while in some cases, the algorithm implied by Theorem 3 is not practical, in other cases, we get algorithms that are sufficiently efficient.

It is also often possible to use some optimizations that decrease the running time. For instance, many machines will have a state s_r that will never lead to an accepting state later. Here, we can see that we do not need to compute values of the form $Q_i(c, s_r)$ for any node i and any configuration c. (For notation, see the proof of Theorem 3.) More Myhill-Nerode type optimizations (compare [15]) are also possible. For instance, computing the probability that there is at most one true variable needs tables with $\ell + 3$ entries per node in the parse tree, ℓ the treewidth.

Finally, in practical cases, one can try to design more efficient algorithms by hand. The situation can resemble the experiences with the use of Courcelle's theorem: Consider some practical problem P, which can be formulated in monadic second order logic. Courcelle's theorem directly tells us that the problem can be solved in linear time for graphs of bounded treewidth. However, a direct use of the machinery behind the proof of Courcelle's theorem would most probably give an algorithm that is too slow in practice, due to large hidden constants in the

O-notation. However, in most cases, algorithms that are tailor made for problem P will have much better constant factors, and may be practical for small values of treewidth (see e.g. [20].)

A direct application of Courcelle's theorem may also yield an automaton that is too large to effectively build.A recent technique by Courcelle and Durand [11] helps to (partially) overcome such problems. It is interesting to investigate if the techniques from [11] can give practical improvements for implementations of Theorem 3.

If a tree decomposition of bounded width is not given with the input, it is possible to find one in linear time (assuming the width bound is a constant): run the algorithm of [2] on the undirected graph, obtained by dropping directions of edges and replacing hyperedges by a clique. Unfortunately, this algorithm is not practical, even for small values of the treewidth. However, there are good heuristics that often perform very well, see [4,5] for overviews.

Final conclusions. In this paper, we have shown that two famous results from different fields of computer science can be combined: the algorithm for probabilistic inference by Lauritzen and Spiegelhalter, and the result by Courcelle that problems in CMSOL can be solved in linear time when the treewidth of the graph is bounded. The formalism chosen in this paper to present the results may depart from what is sometimes usual; a description in other formalisms (tree decompositions or clique trees) is also possible, but seems to require more clumsy details.

The result allows us to compute the probability of several properties of the network and the values of the variables in linear time. For some properties, the constant factor hidden in the 'O' may yield it impractical, but for other properties, one can expect that the resulting algorithm is indeed sufficiently efficient.

An interesting question is whether other problems that were studied for probabilistic networks have a similar CMSOL variant. Other interesting theoretical questions include:

- Is Inference on probabilistic networks with moral graphs of bounded treewidth solvable in logspace? (Compare [14].)
- Inference is $\#P$-hard [24]. What complexities have computing the probabilities of CMSOL properties on general probabilistic networks (i.e., without bounds on treewidth)?
- Are there other graph problems for which a variant of Theorem 3 holds? One possible direction may be to look at optimization problems, e.g., with the finite integer index property [7]. Also, when we allow polynomial running time instead of linear, it is to be expected that a larger class of problems can be handled.

Acknowledgement. I thank the referees for several very helpful comments, and colleagues from the Decision Support Systems and the Algorithmic Systems groups at the Department of Information and Computing Science of Utrecht University for useful discussions.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms 12, 308–340 (1991)
2. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
3. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
4. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations I. Upper bounds. Information and Computation 208, 259–275 (2010)
5. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations II. Lower bounds. Information and Computation 209, 1103–1119 (2011)
6. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. SIAM Journal on Discrete Mathematics 6, 181–188 (1993)
7. Bodlaender, H.L., van Antwerpen-de Fluiter, B.: Reduction algorithms for graphs of small treewidth. Information and Computation 167, 86–119 (2001)
8. Borie, R.B., Parker, R.G., Tovey, C.A.: Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. Algorithmica 7, 555–581 (1992)
9. Borie, R.B., Parker, R.G., Tovey, C.A.: Solving problems on recursively constructed graphs. ACM Computing Surveys 41(4) (2008)
10. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
11. Courcelle, B., Durand, I.A.: Fly-Automata, Their Properties and Applications. In: Bouchou-Markhoff, B., Caron, P., Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2011. LNCS, vol. 6807, pp. 264–272. Springer, Heidelberg (2011)
12. Courcelle, B., Mosbah, M.: Monadic second-order evaluations on tree-decomposable graphs. Theoretical Computer Science 109, 49–82 (1993)
13. Daskalakis, C., Papadimitriou, C.H.: Computing pure Nash equilibria in graphical games via Markov random fields. In: Feigenbaum, J., Chuang, J.C.-I., Pennock, D.M. (eds.) Proceedings 7th ACM Conference on Electronic Commerce EC-2006, pp. 91–99. ACM (2006)
14. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle, pp. 143–152 (2010)
15. Fellows, M.R., Langston, M.A.: An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, FOCS 1989, pp. 520–525 (1989)
16. Jensen, F.V., Nielsen, T.D.: Bayesian Networks and Decision Graphs, 2nd edn. Information Science and Statistics. Springer (2007)
17. Kabanets, V.: Recognizability Equals Definability for Partial k-paths. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 805–815. Springer, Heidelberg (1997)
18. Kaller, D.: Definability equals recognizability of partial 3-trees and k-connected partial k-trees. Algorithmica 27, 348–381 (2000)
19. Kloks, T.: Treewidth. Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
20. Koster, A.M.C.A., van Hoesel, S.P.M., Kolen, A.W.J.: Solving partial constraint satisfaction problems with tree decomposition. Networks 40(3), 170–180 (2002)
21. Lauritzen, S.J., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. The Journal of the Royal Statistical Society. Series B (Methodological) 50, 157–224 (1988)

22. Pearl, J.: Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, Palo Alto (1988)
23. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
24. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82, 273–302 (1996)
25. van der Gaag, L.C.: Probability-Based Models for Plausible Reasoning. PhD thesis, University of Amsterdam (1990)
26. Wimer, T.V.: Linear Algorithms on k-Terminal Graphs. PhD thesis, Dept. of Computer Science, Clemson University (1987)

Cinderella versus the Wicked Stepmother

Marijke Hans L. Bodlaender[1], Cor A.J. Hurkens[2], Vincent J.J. Kusters[2], Frank Staals[2], Gerhard J. Woeginger[2], and Hans Zantema[2]

[1] Dept. of Information and Computing Sciences, Universiteit Utrecht, Netherlands
[2] Dept. of Mathematics and Computer Science, TU Eindhoven, Netherlands

Abstract. We investigate a combinatorial two-player game, in which one player wants to keep the behavior of an underlying water-bucket system stable whereas the other player wants to cause overflows. This game is motivated by data management applications in wireless sensor networks. We construct optimal strategies and characterize optimal bucket sizes for many instances of this game.

1 Introduction

Motivated by a data management application in wireless sensor networks, Bender & al [1] study the *minimum-backlog problem* which is a two-player game on an undirected graph. The vertices of the graph contain buckets (which model buffers) that can store water (which models data). In every time step the adversary distributes exactly one liter of water over the buckets. The player then moves from his current vertex to an adjacent one and empties the corresponding bucket. The player's objective is to minimize the maximum amount of water in any bucket at any time, or in other words, to prevent the buckets from overflowing while using the smallest possible bucket size. Bodlaender & al [2] discuss another variant where in every time step the player can empty a subset of buckets standing in an arbitrary independent set in the graph. Polishchuk & Suomela [7] investigate the variant of the minimum-backlog problem where the underlying metric space is not a graph but the Euclidean plane. Chrobak & al [3] discuss related scenarios in undirected graphs where data/water arrives continuously over time and where the player can (continuously) empty an entire independent set of buckets/buffers; if the player spends t time units on a bucket set, then the contents of each such bucket is decreased by t. Note that in [1] the graph structure constrains the route taken by the player, whereas in [3] the graph structure constrains the sets of buckets that the player can empty simultaneously.

In the current paper we will concentrate on discrete scenarios where data/water arrives in rounds and where in every round the player can empty certain subsets of the buckets. One of the simplest cases of our game is as follows [8,6].

"Five empty buckets of capacity b stand in the corners of a regular pentagon. Cinderella and her wicked Stepmother play a game that goes through a sequence of rounds: at the beginning of every round, the Stepmother takes one liter of water from the nearby river, and distributes it arbitrarily over the five buckets.

Then Cinderella chooses a pair of neighboring buckets, empties them into the river, and puts them back into the pentagon. Then the next round begins. The Stepmother's goal is to make one of these buckets overflow. Cinderella's goal is to prevent this. For which bucket sizes b can the Stepmother eventually enforce a bucket overflow? And for which bucket sizes can Cinderella keep the game running forever?"

We study a general bucket game BG(n, c) for integers n and c with $1 \leq c \leq n-1$, where there are $n \geq 2$ buckets standing in a circle. Throughout we will use the term Cinderella to denote the player and the wicked Stepmother to denote the adversary. In every round the Stepmother first distributes one liter over the n buckets, and then Cinderella empties an arbitrary group of c consecutive buckets. The Stepmother wants to reach a bucket overflow, and Cinderella wants to avoid this. Clearly the above Cinderella puzzle coincides with BG$(5, 2)$. We define $F'(n, c)$ as the infimum of all bucket sizes for which Cinderella can keep the game running forever, and we furthermore introduce the quantity $F(n, c) = F'(n, c)-1$. If Cinderella consistently avoids overflows for buckets of size $F'(n, c)$, then at the end of every round she will only leave buckets with contents $F(n, c)$ or less.

Summary of Results. Table 1 lists the values $F(n, c)$ for all games with $n \leq 12$ buckets. For every single entry in this table with $n \leq 10$ we have proofs that were constructed by humans (and that are presented in this paper) as well as computerized proofs (that have been done with the SMT solver YICES [5]). For some of the entries in the lines $n = 11$ and $n = 12$, we only have computer proofs.

The entries in the table might seem somewhat chaotic at first sight. But taking a second look, the reader perhaps notices that the topmost numbers 1, 1/2, 1/3, ..., 1/11 in the columns are the reciprocals of the positive integers. This indeed is a (fairly shallow) mathematical fact which we present in Section 3, and which says that $F(c+1, c) = 1/c$ for all $c \geq 1$. Next let us discuss the values $F(c+2, c)$ immediately below the topmost numbers, which are

$$3/2, \quad 1, \quad 5/9, \quad 1/2, \quad 7/20, \quad 1/3, \quad 9/35, \quad 1/4, \quad 11/54, \quad 1/5.$$

We note that the values in the even positions again are the reciprocals of integers. Indeed Section 2 shows that the function values $F(n, c)$ only depend on the ratio n/c, which for even $c = 2s$ implies $F(2s + 2, 2s) = F(s + 1, s) = 1/s$. Section 3 shows that the remaining values in the odd positions satisfy $F(c + 2, c) = (2c + 4)/(c^2 + 3c)$. By stepping further down in the columns, we meet the values $F(c + 3, c)$ which read

$$11/6, \quad 1, \quad 1, \quad 17/30, \quad 1/2, \quad 1/2, \quad 69/196, \quad 1/3, \quad 1/3.$$

The values $F(3s + 3, 3s) = F(s + 1, s) = 1/s$ are of course once again the reciprocals of the integers. Section 3 shows that also the values $F(3s+2, 3s-1) = 1/s$ are such reciprocals, and it fully explains the (more complicated) structure of the remaining values $F(3s + 1, 3s - 2)$.

Table 1. Summary of the values $F(n, c)$ for $n \leq 12$ buckets. For the entries with $n \leq 10$, we even have proofs constructed by humans.

$n \backslash c$	1	2	3	4	5	6	7	8	9	10
2	1	–	–	–	–	–	–	–	–	–
3	3/2	1/2	–	–	–	–	–	–	–	–
4	11/6	1	1/3	–	–	–	–	–	–	–
5	25/12	1	5/9	1/4	–	–	–	–	–	–
6	137/60	3/2	1	1/2	1/5	–	–	–	–	–
7	49/20	3/2	1	17/30	7/20	1/6	–	–	–	–
8	363/140	11/6	1	1	1/2	1/3	1/7	–	–	–
9	761/280	11/6	3/2	1	299/525	1/2	9/35	1/8	–	–
10	7129/2520	25/12	3/2	1	1	5/9	69/196	1/4	1/9	–
11	7381/2520	25/12	3/2	1	1	77/135	1/2	1/3	11/54	1/10
12	83711/27720	137/60	11/6	3/2	1	1	5/9	1/2	1/3	1/5

Moving further down in Table 1, we eventually hit an area that entirely consists of 1-entries. The uppermost 1-entry in every column is $F(2c, c) = F(2, 1) = 1$, and these entries form the so-called *half-diagonal* of the table (the diagonal where c is half of n). This half-diagonal is a natural separation line, and it turns out that the combinatorics of the games below the half-diagonal behaves quite differently from the combinatorics of the games above the half-diagonal. Going even further down, we see that the lowermost 1-entry in every column is $F(3c-1, c) = 1$. In other words $F(n, c) = 1$ holds whenever $2c \leq n \leq 3c-1$, and this is a mathematical theorem which we establish in Section 5. This theorem actually is our main result, and its proof is long and involved and uses quite delicate invariants.

What else is going on below the half-diagonal? The first column lists the harmonic numbers $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$. The second column (below the half-diagonal) seems to list again the harmonic numbers, but this time with every term occurring twice. And also the third column (below the half-diagonal) seems to list the harmonic numbers, with every term occurring thrice. And so on. We settle the behavior of the first column in Section 5, and we furthermore derive some partial results on the other columns. Many questions remain open.

Also the global structure of Table 1 shows many interesting properties. Of course, the values in every row form a non-increasing sequence (since c increases and Cinderella becomes more powerful), and for similar reasons the values in every column form a non-decreasing sequence. In fact an even stronger property holds true (see Section 2): the function $F(n,c)$ is non-decreasing in the ratio n/c. Here is one application of this fact: from $F(3,2) = F(8,5) = 1/2$ and from $3/2 < 11/7 < 8/5$ we immediately deduce $F(11,7) = 1/2$.

Organization of the Paper. Section 2 states simple observations, summarizes the notation, and explains the general setup of our proofs. Section 3 deals with the games above the half-diagonal, and Section 5 deals with the games below the half-diagonal. Section 6 gives some conclusions.

2 Preliminaries, Notations, and Conventions

The n buckets in any fixed game BG(n,c) are ordered along the circle and denoted $1, 2, \ldots, n$. The numbering of buckets is always taken modulo n, so that k and $n+k$ denote the same bucket. We use $d(i,j) = \min\{|i-j|, n-|i-j|\}$ to denote the *distance* between buckets i and j along the circle. If $d(i,j) \geq c$, then Cinderella can not simultaneously empty i and j within a single round. A subset S of buckets is called *independent*, if it does not contain two adjacent buckets. The family $\mathcal{I}$ consists of all independent bucket subsets.

The contents of the buckets at a particular moment in time are often summarized in a vector $x = (x_1, \ldots, x_n)$ where x_i denotes the current contents of bucket i. For a subset S of the buckets, we use $x(S) = \sum_{i \in S} x_i$. To keep the notation simple, we write $x(i,j)$ short for $x(\{i,j\})$ and $x(i,j,k)$ short for $x(\{i,j,k\})$, and we use x_i and $x(i)$ interchangeably.

The following two lemmas imply that function $F(n,c)$ only depends on the ratio n/c, and that it is non-decreasing in this ratio.

Lemma 1. *$F(\lambda n, \lambda c) = F(n,c)$ for all integers $\lambda \geq 1$.*

Proof. Consider an arbitrary strategy for the Stepmother for BG(n,c). The Stepmother can emulate this strategy in BG$(\lambda n, \lambda c)$ by using the buckets $\lambda, 2\lambda, \ldots, n\lambda$. This yields $F(\lambda n, \lambda c) \geq F(n,c)$. Vice versa, Cinderella can carry over strategies from BG(n,c) to BG$(\lambda n, \lambda c)$. She cuts the circle into n intervals with λ buckets, treats every interval as a super-bucket, and uses her strategy for BG(n,c) on the super-buckets. This yields $F(\lambda n, \lambda c) \leq F(n,c)$. □

Lemma 2. *(Monotonicity lemma) $F(n_1,c_1) \leq F(n_2,c_2)$ whenever $n_1/c_1 \leq n_2/c_2$.*

Proof. This follows from $F(n_1,c_1) = F(n_1n_2, c_1n_2) \leq F(n_1n_2, c_2n_1) = F(n_2,c_2)$. □

By the definition of $F'(n,c)$, Cinderella wins the game BG(n,c) if the bucket size is strictly larger than $F'(n,c)$, and the Stepmother wins the game if the bucket size is strictly smaller than $F'(n,c)$. What happens at the threshold $F'(n,c)$?

Lemma 3. *If BG(n,c) is played with buckets of size $F'(n,c)$, then Cinderella can keep the game running forever.*

Proof. Let $G(n, c, R)$ denote the maximum amount of water that the Stepmother can accumulate in some bucket within the first R rounds of game $\mathrm{BG}(n, c)$; it can be proved by an inductive argument that this maximum indeed exists.

Now suppose that the Stepmother could enforce an overflow for buckets of size $F'(n, c)$. Then she can enforce this overflow after a finite number R of rounds, which means $G(n, c, R) > F'(n, c)$. But then within R rounds the Stepmother could as well enforce overflows for any bucket size between $F'(n, c)$ and $G(n, c, R)$, which conflicts with the definition of $F'(n, c)$. $\square$

Lower Bounds from Balancing Stepmothers. Our lower bound arguments for $F(n, c)$ use a special adversary which we call *balancing Stepmother.* A balancing Stepmother balances the water levels in certain buckets, and works in two phases. During the first phase, the Stepmother always distributes her liter in such a way that all n buckets are filled to the same level. This common filling level is $1/n$ in the first round, and in later rounds increases and converges to $1/c$. The first phase ends, when the common filling level exceeds $1/c - \varepsilon$ (where ε is a tiny positive real number that can be made arbitrarily close to 0). The set of $n - c$ buckets that are filled to level $L_1 \approx 1/c$ at the end of the last round of the first phase is denoted by S_1.

In the second phase, we will usually ignore the dependence of our bounds on ε, so that the presentation remains simple and our formulas stay clean. The second phase goes through $n - c - 1$ further rounds. At the beginning of the r-th one of these rounds ($r = 1, \ldots, n - c - 1$), there are $(n - c) - r + 1$ buckets filled to the same level L_r that Cinderella could not empty in the preceding round; these buckets form the set S_r. The balancing Stepmother then picks an appropriate set $T_r \supseteq S_r$ of buckets, such that in the current round Cinderella must leave at least $(n - c) - r$ buckets in T_r untouched. All buckets in T_r are then filled to the same level $L_{r+1} = (|S_r|\, L_r + 1)/|T_r|$. At the end of the last round $n - c - 1$, there remains a single non-empty bucket whose contents L_{n-c-1} forms the resulting lower bound.

Upper Bounds from Invariants. Our upper bound arguments for $F(n, c)$ are based on appropriate systems of invariants that (i) can be maintained by Cinderella, and that (ii) imply that every bucket contents remains below $F(n, c)$. A typical invariant system bounds the contents of every bucket by $x_i < F(n, c)$, and furthermore bounds the overall contents of certain groups of buckets. All invariants are trivially satisfied at the beginning of the first round when all buckets are empty. In our proofs we usually assume inductively that these invariants are satisfied at the beginning of some fixed round (just before the Stepmother moves), and then show that Cinderella can re-establish them at the end of the round. In doing this, we always let x_i denote the contents of bucket i at the beginning of the round, and we always let y_i denote the contents of bucket i after the Stepmother has moved.

3 Above the Half-Diagonal

By definition the games BG(n, c) above the half-diagonal satisfy $n < 2c$. It is not hard to see that all these games satisfy $F(n, c) < 1$ (as at the end of her move, Cinderella can keep the total amount of water in the system below $n/c - 1$). We fully understand the games BG$(c+1, c)$, BG$(c+2, c)$, and BG$(c+3, c)$.

Theorem 1. *$F(c+1, c) = 1/c$ holds for all $c \geq 1$.*

Proof. (Upper bound) As invariant, Cinderella always leaves a single bucket untouched whose contents is below $1/c$. The Stepmother adds one liter to the system and increases the total amount of water to less than $(c+1)/c$. By averaging, one of the $c+1$ buckets has contents below $1/c$, and that's the bucket that Cinderella does not touch in her move.

(Lower bound) In her first phase, the balancing Stepmother brings the contents of all buckets arbitrarily close to $1/c$. □

Theorem 2. *$F(c+2, c) = (2c+4)/(c^2+3c)$ holds for all odd $c \geq 1$, and $F(c+2, c) = 2/c$ holds for all even $c \geq 2$.*

Theorem 3. *The values $F(c+3, c)$ behave as follows for $c \geq 1$.*

(i) $F(3s+2, 3s-1) = 1/s$
(ii) $F(3s+3, 3s) = 1/s$
(iii) $F(3s+4, 3s+1) = \dfrac{(s+1)(6s+11)}{(s+2)(2s+3)(3s+1)}$

Theorem 4. $F(9, 5) = 299/525$.

The proofs of Theorems 2, 3 and 4 are to be found in the subsections below. It is easily verified that these theorems (together with monotonicity) imply all entries for $n \leq 10$ above the half-diagonal of Table 1.

We think that also all values $F(c+4, c)$ are within reach and could be fully characterized, if one invests sufficient time and energy. We actually determined many values $F(n, c)$ above the half-diagonal with the help of computer programs. For instance we know that $F(13, 9) = 37/105$ and $F(17, 13) = 1961/7605$, and it took us hours of computation time to establish $F(16, 11) = 252/715$. We see many patterns and regularities in the data, but we can not find a unifying conjecture that would systematically cover all possible cases; certain divisibility properties seem to kick in and totally mess up the structure. Our data suggests the following conjecture (which is fairly weak and only covers a small part of the unknown area).

Conjecture 1. $F(n, c) = 1/2$ holds for all n and c with $3/2 \leq n/c < 5/3$.

3.1 The Proof of Theorem 2

Since monotonicity settles the cases with even c, we only discuss the games where c is odd (and $n = c + 2$).

(Upper bound) At the end of every round, Cinderella leaves two non-empty buckets (say buckets 1 and 2) whose loads x_1 and x_2 satisfy the following two invariants:

$$x(1,2) \; < \; 2/c \tag{1a}$$

$$x_1, \; x_2 \; < \; L_2 \; := \; (2c+4)/(c^2+3c) \tag{1b}$$

Then the Stepmother moves and yields bucket contents $y_1, \dots, y_{c+2}$. Cinderella maintains the invariants by leaving a pair $j, j+1$ of neighboring buckets with smallest total contents. Since the Stepmother only adds a single liter, invariant (1a) implies

$$\sum_{i=1}^{c+2} y_i \; < \; (c+2)/c. \tag{2}$$

By averaging we get $y(j,j+1) \le (2\sum y_i)/(c+2) < 2/c$, which ensures invariant (1a). Next, suppose for the sake of contradiction that $y_j \ge L_2$. Partition the remaining $c+1$ buckets (except bucket j) into $(c+1)/2$ pairs of neighboring buckets. The total contents of every such pair is at least $y(j,j+1) \ge L_2$, which implies $\sum_{i=1}^{c+2} y_i \ge \frac{1}{2}(c+3)\, L_2 = (c+2)/c$, and thus contradicts (2). Hence $y_j < L_2$, and an analogous argument yields $y_{j+1} < L_2$.

(Lower bound) The first phase of the balancing Stepmother ends with two buckets (say 1 and 2) of contents very close to $1/c$. In the second phase, the Stepmother chooses set T_1 to contain buckets 1 and 2 together with all buckets with even numbers; note that $|T_1| = (c+3)/2$. Then all buckets in T_1 are brought to level at least $(2/c+1)/|T_1| = L_2$. Since Cinderella cannot simultaneously empty all buckets in T_1, we get $F(c+2,c) \ge L_2$.

3.2 The Proof of Theorem 3.(i) and (ii)

Monotonicity and Theorem 1 yield the lower bound $F(3s+2, 3s-1) \ge F(3s+3, 3s) = F(s+1, s) = 1/s$. Hence we will concentrate on the upper bound for the game with $n = 3s+2$ buckets and $c = 3s-1$.

At the end of some fixed round Cinderella leaves three adjacent buckets, say the buckets $3, 4, 5$. She always maintains the following two invariants.

$$x_4 \; < \; 1/s \tag{3a}$$

$$x(3,5) \; < \; 1/s \tag{3b}$$

The Stepmother adds one liter to the system and brings the contents to $y_1, y_2, \dots, y_{3s+2}$. A *triple* is a group of three consecutive buckets i, $i+1$, $i+2$ in the circle. A triple is called *good*, if $y(i, i+1, i+2) < 1/s$. By emptying all buckets outside a good triple, Cinderella can maintain the invariants. Hence we assume from now on that there is no good triple.

We denote by W the total amount of water in all buckets except bucket 4. Invariant (3b) implies $W < 1 + 1/s$. Since there are no good triples, we have

$y(1,2,3) \geq 1/s$ and $y(5,6,7) \geq 1/s$. By subtracting these two inequalities from $W < 1 + 1/s$, we get

$$\sum_{i=8}^{3s+2} y_i \; < \; 1 - 1/s. \tag{4}$$

Next suppose for the sake of contradiction that $y_{3i+2} \geq 1/s$ holds for some i with $2 \leq i \leq s$. Then the $3i-6$ buckets $8, 9, \ldots, 3i+1$ and the $3s-3i$ buckets $3i+3, 3i+4, \ldots, 3s+2$ can be divided into $s-2$ non-good triples. Therefore the overall amount of water in these $s-2$ triples together with y_{3i+2} would be at least $(s-1)/s$, which contradicts (4). This contradiction implies $y_{3i+2} < 1/s$ for $2 \leq i \leq s$. Furthermore we assume $y(3i+1, 3i+3) \geq 1/s$, since otherwise Cinderella could easily maintain the invariants by emptying all buckets except the triple $3i+1, 3i+2, 3i+3$. Summing these $s-1$ inequalities for $2 \leq i \leq s$ yields

$$\sum_{i=2}^{s} y_{3i+1} + \sum_{i=2}^{s} y_{3i+3} \; \geq \; 1 - 1/s. \tag{5}$$

If $y(6,8) \geq 1/s$, then (5) yields that the Stepmother has added her entire liter to the buckets outside the triple $3,4,5$, and Cinderella can maintain all invariants by reverting the system to the preceding state. Hence we assume from now on $y(6,8) < 1/s$, and a symmetric argument yields $y(3s+2, 2) < 1/s$. If $y_7 < 1/s$ or $y_1 < 1/s$, then Cinderella maintains the invariants by emptying everything except the triple $6,7,8$, respectively by emptying everything except the triple $3s+2, 1, 2$. Hence we assume from now $y_7 \geq 1/s$ and $y_1 \geq 1/s$.

Finally note that the $3s-5$ buckets $8, 9, 10, \ldots, 3s+2$ contain $s-2$ pairwise disjoint triples, each of which is non-good and has total contents at least $1/s$. Together with $y_1 \geq 1/s$ and $y_7 \geq 1/s$ this shows that the Stepmother must have added her entire liter to the buckets $7, 8, 9, 10, \ldots, 3s+2$, and 1. By emptying these buckets, Cinderella reverts the system to the preceding state and maintains all invariants.

3.3 The Proof of Theorem 3.(iii)

We discuss the game with $n = 3s+4$ and $c = 3s+1$. For $s \geq 1$, we introduce three parameters L_1, L_2, L_3 by

$$L_1 = \frac{1}{3s+1}$$

$$L_2 = \frac{3s+4}{(2s+3)(3s+1)}$$

$$L_3 = \frac{(s+1)(6s+11)}{(s+2)(2s+3)(3s+1)}$$

Note that these three parameters satisfy

$$L_2 \;=\; \frac{3L_1+1}{2s+3} \quad \text{and} \quad L_3 \;=\; \frac{2L_2+1}{s+2}. \tag{6}$$

Furthermore, we have

$$L_1 \;\le\; L_2 \;\le\; L_3 \;\le\; 2L_2 \;\le\; 3L_1, \tag{7}$$

and finally there is the useful inequality

$$3L_1+1 \;\le\; (s+1)\,L_3+2L_2. \tag{8}$$

(Upper bound) At the end of some fixed round Cinderella leaves three non-empty buckets, say buckets $1, 2, 3$. She maintains the following three invariants.

$$x_i \;<\; L_3 \qquad \text{for } 1 \le i \le 3 \tag{9a}$$

$$x(i,j) \;<\; 2L_2 \qquad \text{for } 1 \le i < j \le 3 \tag{9b}$$

$$x(1,2,3) \;<\; 3L_1 \tag{9c}$$

Then the Stepmother moves, and raises the bucket contents to $y_1, y_2, \ldots, y_{3s+4}$. By invariant (9c) the overall amount of water in the system is bounded by

$$\sum_{j=1}^{3s+4} y_j \;<\; 3L_1+1. \tag{10}$$

A bucket i with $y_i \ge L_3$ is called *large*. A *triple* is a group of three consecutive buckets i, $i+1$, $i+2$ in the circle. A triple is called *good*, if (i) none of its buckets is large and (ii) $y(i,i+1,i+2) < 2L_2$. If Cinderella empties all buckets outside a good triple, she automatically maintains the invariants. Hence we assume from now on that there is no good triple.

Lemma 4. *If there is no good triple, then there also are no large buckets.*

Proof. We distinguish several cases on the number ℓ of large buckets. The overall amount of water in the system is at least ℓL_3, and below $3L_1+1$ by (10). By using (8) and (7) this yields

$$\ell L_3 \;<\; 3L_1+1 \;\le\; (s+1)\,L_3+2L_2 \;\le\; (s+3)\,L_3.$$

Therefore $\ell \le s+2$. If $\ell = s+2$ and at most two of the buckets $1, 2, 3$ are large, then (9b) implies that the overall amount W of water in the large buckets satisfies

$$(s+2)L_3 \;\le\; W \;<\; 2L_2+1,$$

which contradicts (6). If $\ell = s+2$ and all three buckets $1, 2, 3$ are large, then these $s+2$ large buckets divide the $2s+2$ non-large buckets into at most s non-empty intervals along the circle. One of these intervals contains at least three non-large buckets, and hence a non-good triple whose total contents is at least $2L_2$. Then the overall amount W of water in the large buckets plus the water in this non-good triple satisfies

$$(s+2)L_3 + 2L_2 \;\leq\; W \;<\; 3L_1 + 1,$$

which contradicts (8). In the remaining cases we have $\ell \leq s+1$.

If $\ell \geq 1$, the large buckets divide the $3s+4-\ell$ non-large buckets into ℓ intervals along the circle. If an interval consists of k non-large buckets, we can find $\lfloor k/3 \rfloor$ pairwise disjoint triples in this interval. It can be seen that altogether we find at least $s+2-\ell$ pairwise disjoint triples in all ℓ intervals. Each of these triples is non-good and has total contents at least $2L_2$. By applying (10) the total contents W of all buckets satisfies

$$\ell L_3 + 2L_2(s+2-\ell) \;\leq\; W \;<\; 3L_1 + 1.$$

Since $L_3 - 2L_2 \leq 0$, the expression in the left hand side is decreasing in ℓ. Together with $\ell \leq s+1$ this yields $(s+1)\,L_3 + 2L_2 < 3L_1 + 1$, which contradicts (8). This leaves $\ell = 0$ as the only possible case. □

By the lemma there is no large bucket, and we see that all buckets a priori satisfy invariant (9a). Consider a fixed bucket i, and divide the remaining $3s+3$ buckets into $s+1$ non-good triples. Then

$$y_i + (s+1)\cdot 2L_2 \;\leq\; \sum_{j=1}^{3s+4} y_j \;<\; 3L_1 + 1 \;=\; (2s+3)\,L_2,$$

which implies $y_i < L_2$. Hence any pair of buckets satisfies invariant (9b). By averaging, there exists a triple of buckets whose total contents is

$$y_j + y_{j+1} + y_{j+2} \;<\; 3\cdot\frac{3L_1+1}{3s+4} \;=\; 3L_1.$$

Cinderella empties all buckets except this triple, and thereby also fulfills invariant (9c).

(Lower bound) The first phase of the balancing Stepmother ends with three buckets (say buckets $1, 2, 3$) having contents very close to L_1. The second phase goes through two further rounds.

In the first of these rounds, the Stepmother selects the set T_1 to contain all buckets except the buckets $3i+1$ with $i = 1, \ldots, s+1$. Then T_1 contains $2s+3$ buckets which the Stepmother all brings to contents L_2. Cinderella leaves a set S_2 of two buckets with contents L_2; these two buckets are either adjacent (say 1 and 2) or separated by a single other bucket (say buckets $3s+4$ and 2).

In the second round, the Stepmother selects the set T_2 to contain the two buckets in set S_2 together with the buckets $3i+2$ with $i = 1, \ldots, s$. Then T_2 consists of $s+2$ buckets which the Stepmother all brings to contents L_3. Cinderella must leave one bucket with contents L_3 at the end of the round.

4 The Proof of Theorem 4

(Upper bound) Assume that in the game BG(9, 5), Cinderella leaves a bucket configuration that satisfies the following four invariants.

$$x_i < 299/525 \approx 0.569 \qquad \text{for } 1 \le i \le 9 \tag{11a}$$

$$x(S) < 124/175 \approx 0.708 \qquad \text{for all } S \text{ with } |S| = 2 \tag{11b}$$

$$x(S) < 27/35 \approx 0.771 \qquad \text{for all } S \text{ with } |S| = 3 \tag{11c}$$

$$x(S) < 4/5 = 0.800 \qquad \text{for all } S \text{ with } |S| = 4 \tag{11d}$$

The Stepmother moves and raises the bucket contents from $x_1, \ldots, x_9$ to $y_1, \ldots, y_9$. Note that $\sum_{i=1}^{9} y_i < 9/5$ by (11d). A *quadruple* is a set of four consecutive buckets in the circle. A quadruple is called *good*, if its four buckets satisfy (11a)–(11d). If there is a good quadruple, then Cinderella can maintain the invariants by emptying all buckets outside the quadruple.

Lemma 5. *If $y_j \ge 299/525$ for some j, then Cinderella can maintain the invariants.*

Proof. A bucket j with $y_j \ge 299/525 =: L$ is called *large*. If the Stepmother leaves three large buckets i, j, k, then $x(i, j, k) \ge y(i, j, k) - 1 \ge 3L - 1 = 124/175$. This implies that x_i, x_j, x_k all are non-zero, since otherwise two of these buckets would have violated (11b). Hence i, j, k all belong to the quadruple that Cinderella did not touch in the preceding round. If Cinderella empties this quadruple (together with some fifth bucket), the remaining volume of water decreases to $9/5 - 3L < 124/175$, and all invariants are maintained.

If the Stepmother leaves two large buckets i and j, then Cinderella empties these large buckets (together with three other buckets). The remaining volume of water decreases to $9/5 - 2L < 124/175$, and all invariants are maintained.

Finally assume that the Stepmother leaves a single large bucket, which without loss of generality is bucket 1. Then $y(2, 3, 4, 5) + y(6, 7, 8, 9) \le 9/5 - L < 2 \cdot 124/175$, which implies that one of the quadruples $2, 3, 4, 5$ and $6, 7, 8, 9$ must be good. □

Lemma 6. *If $y_j \ge 62/175$ for some j, then Cinderella can maintain the invariants.*

Proof. By the preceding lemma we assume $y_i < 299/525$ for all i. We assume furthermore that bucket 1 with $y_1 \ge 62/175$ is the fullest bucket, and that the quadruples $2, 3, 4, 5$ and $6, 7, 8, 9$ both are non-good (so that the total contents of either quadruple is at least 124/175). If the quadruple $2, 3, 4, 5$ violates (11c) or (11d), then we would get the contradiction

$$y_1 + y(2, 3, 4, 5) + y(6, 7, 8, 9) \ \ge\ 62/175 + 27/35 + 124/175 \ >\ 9/5.$$

Hence the quadruple $2, 3, 4, 5$ contains two buckets b_1, b_2 that violate (11b) with $y(b_1, b_2) \ge 124/175$. Symmetric arguments show that the quadruple $6, 7, 8, 9$ contains two buckets b_3, b_4 with $y(b_3, b_4) \ge 124/175$.

Let $T = \{1, b_1, b_2, b_3, b_4\}$ and note $x(T) \geq y(T) - 1 \geq 27/35$. Now (11c) implies that T contains all the four buckets that Cinderella did not touch in the preceding round. By emptying this quadruple (together with some fifth bucket), the remaining volume of water goes below 124/175 and all invariants are maintained. □

By the above lemmas we assume from now on $y_i < 62/175$ for all i, so that invariants (11a) and (11b) become harmless. Consider an arbitrary bucket k, and consider the partition of the remaining eight buckets into two quadruples T_1 and T_2, so that

$$y_k \; < \; 9/5 - y(T_1) - y(T_2). \tag{12}$$

We may assume that both quadruples T_1 and T_2 are non-good. Then the lower bounds $y(T_1), y(T_2) \geq 27/35$ and (12) together yield $y_k < 9/35$. Since k was an arbitrary bucket, this means that every bucket triple satisfies (11c), which also makes invariant (11c) harmless. Since T_1 and T_2 are non-good, we now conclude $y(T_1), y(T_2) \geq 4/5$. But then (12) yields $y_k < 1/5$ for all k, and any move of Cinderella will maintain all invariants. This completes the proof.

(Lower bound) The first phase of the balancing Stepmother ends with four consecutive buckets (say buckets $1, 2, 3, 4$) having contents very close to $1/5$. The second phase goes through three further rounds.

In the first of these rounds, the Stepmother uses set $T_1 = \{1, 2, 3, 4, 6, 7, 8\}$ with all buckets except 5 and 9. The Stepmother brings every bucket in T_1 to contents 9/35. Cinderella leaves a set of four buckets, at least three of which are in T_1. These three buckets are either adjacent (say $2, 3, 4$ in this first case) or separated by a single empty bucket (say $3, 4, 6$ in the second case).

In the second round the Stepmother selects the set T_2 to contain five buckets; in the first case she uses $T_2 = \{2, 3, 4, 7, 8\}$ and in the second case $T_2 = \{3, 4, 6, 7, 8\}$. The Stepmother brings every bucket in T_2 to contents 62/175. Cinderella leaves a set of four buckets, at least two of which are in T_2. We rename the buckets so that 1 and $b \in \{2, 3, 4\}$ keep their contents 62/175.

In the third round the Stepmother uses $T_3 = \{1, b, 6\}$, and fills these three buckets up to level 299/525. Cinderella must leave at least one such bucket with contents 299/525 at the end of the round.

5 Below the Half-Diagonal

By definition the games BG(n, c) below the half-diagonal satisfy $n \geq 2c$. For these games the harmonic sums $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{k}$ seem to play a major role. The following theorem has been observed before by Dietz & Sleator [4] and Chrobak & al. [3].

Theorem 5. *$F(n, 1) = H_{n-1}$ holds for all $n \geq 2$.*

Proof. (Upper bound) Let x_i denote the contents of bucket i at the beginning of some round. We argue that Cinderella can maintain the following invariants.

$$x(T) \; < \; (1 + H_{n-1} - H_{|T|})\, |T| \qquad \text{for all bucket sets } T \tag{13}$$

The Stepmother raises the bucket contents to $y_1, \ldots, y_n$, and we assume that $y_n \geq y_i$ for all i. Then for any bucket set $T \subseteq \{1, \ldots, n-1\}$ we have

$$\begin{aligned} \frac{1}{|T|}\, y(T) &\leq \frac{1}{|T|+1}\,(y(T)+y_n) \;\leq\; \frac{1}{|T|+1}\,(x(T)+x_n+1) \\ &< 1 + H_{n-1} - H_{|T|+1} + \frac{1}{|T|+1} \;=\; 1 + H_{n-1} - H_{|T|}. \end{aligned}$$

Therefore Cinderella can maintain the invariants by emptying the fullest bucket n. By applying (13) to a single bucket set $T = \{i\}$, we get that all buckets satisfy $x_i < H_{n-1}$.

(Lower bound) In the first phase, the balancing Stepmother brings the filling level of all buckets very close to 1. The first phase terminates with a set S_1 of $n-1$ buckets with contents $L_1 \approx 1$. In the second phase, the Stepmother always chooses $T_r := S_r$ as the set of the $n-r$ currently fullest buckets (which Cinderella could not empty in the preceding round), and fills all of them to level $L_{r+1} = 1 + H_{n-1} - H_{n-r-1}$. Then at the end of round $n-2$ Cinderella has left a bucket of contents H_{n-1}. □

Theorem 6. *(i)* $F(7,2) = 3/2$ *and (ii)* $F(9,2) = 11/6$.

Theorem 7. $F(n,c) = 1$ *holds for all* n *and* c *with* $2 \leq n/c < 3$.

The proof of Theorem 6 and the (long and technical) proof of Theorem 7 can be found in the full version of this paper. Note that the theorems in this section together with the monotonicity property imply all entries for $n \leq 10$ below the half-diagonal of Table 1. Furthermore Theorem 7 covers the cases with $\lfloor n/c \rfloor = 2$ for the following clean and natural conjecture.

Conjecture 2. $F(n,c) = F(\lfloor n/c \rfloor, 1)$ holds for all n and c with $2 \leq n/c$.

If true, then this conjecture (in combination with Theorem 5) would determine all values of $F(n,c)$ below the half-diagonal. Note that the monotonicity Lemma 2 yields $F(n,c) \geq F(\lfloor n/c \rfloor, 1)$, and that therefore the hard part of the conjecture is to come up with the right systems of invariants. Unfortunately, we have no idea how to settle Conjecture 2. In fact, we cannot even settle the special case $F(13,2) = 137/60$. The games BG$(n,2)$ with odd $n \leq 11$ can be handled by certain types of invariant systems that we understand very well; all these systems are built around subsets of pairwise non-adjacent buckets that follow a certain pattern. With the help of YICES we can prove that the most natural generalization of this pattern to BG$(13,2)$ will not work out, since there exist situations where Cinderella cannot maintain the corresponding invariants.

6 Final Remarks

We have settled all bucket games BG(n,c) with $n \leq 12$. Some of our smaller results started to grow together, and eventually resulted in general theorems that cover large families of games (as for instance the families in Theorem 3 and Theorem 7). There remain many open questions, and in particular there remains our tantalizing Conjecture 2.

All our lower bounds have been derived by a suitable balancing Stepmother strategy (sometimes in combination with monotonicity). For many games, we performed extensive computer experiments and used backtracking algorithms (written in Haskell) to detect the strongest balancing Stepmothers; this boils down to checking a huge but finite number of cases.

Question 1. Does every value $F(n, c)$ result from an adversary argument with a balancing Stepmother (in combination with monotonicity)?

A positive answer to Question 1 would also imply the truth of the following conjecture.

Conjecture 3. The function $F(n, c)$ only takes rational values, and is Turing-computable.

For some of the considered games it was far from clear how to choose the right system of invariants, and several attempts were required before finding the right choice. For experimenting with such invariants it was convenient to use an SMT solver (Satisfiability Modulo Theories) which checks the satisfiability of any Boolean formula on linear inequalities. Note that this goes far beyond linear programming, as in linear programming the set of constraints is the conjunction of a set of linear inequalities, whereas in SMT any combination of disjunctions and conjunctions is allowed. Now I is an invariant system for Cinderella if the formula

$$I \;\wedge\; \left(\sum_i y_i = 1 + \sum_i x_i\right) \;\wedge\; \left(\bigwedge_i y_i \geq x_i\right) \;\wedge\; \bigvee_j \neg I_j$$

is unsatisfiable. Here the formula I_j $(1 \leq j \leq n)$ is obtained from I as follows. For $i = j+1, \ldots, j+c$ every occurrence of x_i is replaced by 0, and for the remaining indices i every occurrence of x_i is replaced by y_i. Our approach was to check this by the SMT solver YICES for several candidates for I. If this formula is unsatisfiable, we have proved the invariance and thereby derived an upper bound on $F(n, c)$; if it is satisfiable then YICES provides the corresponding values of x_i and y_i that can be interpreted as a counterexample for the invariance. Internally, YICES works with rational numbers in unbounded precision, and typically the proof trees consist of thousands of indigestible case distinctions, but are found within at most a few seconds.

References

1. Bender, M.A., Fekete, S.P., Kröller, A., Mitchell, J.S.B., Liberatore, V., Polishchuk, V., Suomela, J.: The minimum-backlog problem. In: Proceedings of the International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS 2007), pp. 1–18 (2007)

2. Bodlaender, M.H.L., Hurkens, C.A.J., Woeginger, G.J.: The Cinderella Game on Holes and Anti-holes. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 71–82. Springer, Heidelberg (2011)
3. Chrobak, M., Csirik, J.A., Imreh, C., Noga, J., Sgall, J., Woeginger, G.J.: The Buffer Minimization Problem for Multiprocessor Scheduling with Conflicts. In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 862–874. Springer, Heidelberg (2001)
4. Dietz, P.F., Sleator, D.D.: Two algorithms for maintaining order in a list. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987), pp. 365–372 (1987)
5. Dutertre, B., de Moura, L.: Yices – A high-performance SMT solver. Downloadable from http://yices.csl.sri.com
6. Hurkens, A.J.C., Hurkens, C.A.J., Woeginger, G.J.: How Cinderella won the bucket game (and lived happily ever after). Mathematics Magazine 84, 285–290 (2011)
7. Polishchuk, V., Suomela, J.: Optimal Backlog in the Plane. In: Fekete, S. (ed.) ALGOSENSORS 2008. LNCS, vol. 5389, pp. 141–150. Springer, Heidelberg (2008)
8. Woeginger, G.J.: Combinatorics problem C5. In: Problem Shortlist of the 50th International Mathematical Olympiad, Bremen, Germany, pp. 33–35 (2009)

Worst- and Average-Case Privacy Breaches in Randomization Mechanisms*

Michele Boreale[1] and Michela Paolini[2]

[1] Università di Firenze, Italy
michele.boreale@unifi.it
[2] IMT - Institute for Advanced Studies, Lucca, Italy

Abstract. In a variety of contexts, randomization is regarded as an effective technique to conceal sensitive information. We model randomization mechanisms as information-theoretic channels. Our starting point is a semantic notion of security that expresses absence of any privacy breach above a given level of seriousness ϵ, irrespective of any background information, represented as a prior probability on the secret inputs. We first examine this notion according to two dimensions: worst vs. average case, single vs. repeated observations. In each case, we characterize the security level achievable by a mechanism in a simple fashion that only depends on the channel matrix, and specifically on certain measures of "distance" between its rows, like norm-1 distance and Chernoff Information. We next clarify the relation between our worst-case security notion and differential privacy (DP): we show that, while the former is in general stronger, the two coincide if one confines to background information that can be factorised into the product of independent priors over individuals. We finally turn our attention to expected utility, in the sense of Ghosh et al., in the case of repeated independent observations. We characterize the exponential growth rate of any reasonable utility function. In the particular case the mechanism provides ϵ-DP, we study the relation of the utility rate with ϵ: we offer either exact expressions or upper-bounds for utility rate that apply to practically interesting cases, such as the (truncated) geometric mechanism.

Keywords: Foundations of security, quantitative information flow, differential privacy, utility, information theory.

1 Introduction

In a variety of contexts, randomization is regarded as an effective means to conceal sensitive information. For example, anonymity protocols like Crowds [24] or the Dining Cryptographers [11] rely on randomization to "confound" the adversary as to the true actions undertaken by each participant. In the field of Data Mining, techniques have been proposed by which datasets containing personal information that are released for business or research purposes are perturbed with noise, so as to prevent an adversary

* Work partially supported by the EU funded project ASCENS. Corresponding author: Michele Boreale, Università di Firenze, Dipartimento di Sistemi e Informatica, Viale Morgagni 65, I-50134 Firenze, Italy.

from re-identifying individuals or learning sensitive information about them (see e.g. [15] and references therein).

In the last few years, interest in the theoretical principles underlying randomization-based information protection has been steadily growing. Two major areas have by now clearly emerged: *Quantitative Information Flow* (QIF) [8,19,5,6,9,10,26] and *Differential Privacy* (DP) [13,14,21,22,16,17]. As discussed in [4], QIF is mainly concerned with quantifying the degree of protection offered against an adversary trying to guess the whole secret; DP is rather concerned with protection of individual bits of the secret, possibly in the presence of background information, like knowledge of the remaining bits. The areas of QIF and DP have grown separately for some time: only very recently researchers have begun investigating the relations between these two notions [1,2,3,4].

The present paper is an attempt at distilling and systematizing the notions of security breach underlying QIF and DP. We view a randomization mechanism as an information-theoretic channel with inputs in $\mathcal{X}$ and outputs in $\mathcal{Y}$. The starting point of our treatment is a semantical notion of breach. Assume $\mathcal{X}$ is a finite set of items containing the secret information X, about which the adversary has some background knowledge or belief, modeled as a prior probability distribution $p(x)$. Consider a predicate $Q \subseteq \mathcal{X}$ – in a dataset about individuals, one may think of Q as gender, or membership in a given ethnical group etc. The mere fact that X is in Q or not, if ascertained, may convey sensitive information about X. Henceforth, any observation $y \in \mathcal{Y}$ that causes a significant change in the adversary's posterior belief about $X \in Q$ must be regarded as dangerous. In probabilistic terms, Q is a *breach* if, for some prior probability on $\mathcal{X}$, the posterior probability of Q after interaction with the randomization mechanism exhibits a significant change, compared to its prior probability. We decree a randomization mechanism as secure at level ϵ, if it exhibits *no breach* of level $> \epsilon$, independently of the prior distribution on the set of secret data $\mathcal{X}$. The smaller ϵ, the more secure the mechanism. This simple idea, or variations thereof, has been proposed elsewhere in the Data Mining literature – see e.g. [15]. Here, we are chiefly interested in analyzing this notion of breach according to the following dimensions.

1. Worst- vs. average-case security. In the worst-case approach, one is interested in bounding the level of any breach, independently of how likely the breach is. In the average-case, one takes into account the probability of the observations leading to the breach.
2. Single vs. repeated, independent executions of the mechanism.
3. Expected utility of the mechanism and its asymptotic behavior, depending on the number of observations and on a user-defined loss function.

To offer some motivations for the above list, we observe that worst-case is the type of breach considered in DP, while average-case is the type considered in QIF. In the worst-case scenario, another issue we consider is resistance to background information. In the case of DP, this is often stated in the terms that [13]: *Regardless of external knowledge, an adversary with access to the sanitized database draws the same conclusions whether or not my data is included*, and formalized as such [17]. We investigate how this relates to the notion of privacy breach we consider, which also intends to offer protection against arbitrary background knowledge.

Concerning the second point, a scenario of repeated observations seems to arise quite naturally in many applications. For instance, an online, randomized data-releasing mechanism might offer users the possibility of asking the same query a number of times. This allows the user to compute more accurate answers, but also poses potential security threats, as an adversary could remove enough noise to learn valuable information about the secret. This is an instance of the *composition* attacks which are well known in the context of DP, where they are thwarted by allotting each user or group of users a *privacy budget* that limits the overall number of queries to the mechanism; see e.g. [21,16]. For another example, in a de-anonymization scenario similar to [23], [6] shows that gathering information about a target individual can be modeled as collecting multiple observations from a certain randomization mechanism. In general, one would like to assess the security of a mechanism in these situations. In particular, one would like to determine exactly *how fast* the level of any potential breach grows, as the number n of independent observations grows.

The third point, concerning utility, has been the subject of intensive investigation lately (see related work paragraph). Here, we are interested in studying the growth of expected utility in the model of Ghosh et al. [18] as the number of independent observations grows, and to understand how this is related to security.

In summary, the main results we obtain are the following.

- In the scenario of a single observation, both in the average and in the worst case, we characterize the security level (absence of breach above a certain threshold) of the randomization mechanism in a simple way that only depends on certain row-distance measures of the underlying matrix.
- We prove that our notion of worst-case security is stronger than DP. However, we show the two notions coincide when one confines to background information that factorises as the product of independent measures over all individuals. This, we think, sheds further light on resistance of DP against background knowledge.
- In the scenario of repeated, independent observations, we determine the exact asymptotic growth rate of the (in)security level, both in the worst and in the average case.
- In the scenario of repeated, independent observations, we determine the exact asymptotic growth rate of any reasonable expected utility. We also give bounds relating this rate to ϵ-DP, and exact expressions in the case of the geometric mechanisms. In this respect, we argue that the geometric mechanism is superior to its *truncated* version [18].

Related work. There is a large body of recent literature on QIF [8,19,5,6] and DP [13,14]. The earliest proposal of a worst-case security notion is, to the best of our knowledge, found in [15]. As mentioned, the investigation of the relations between QIF and DP has just begun. Both [4] and [2,3] discuss the implication of ϵ-DP on information leakage guarantees, and vice-versa, in the case of a single observation. In the present work, we propose and characterize both worst- and average-case semantic notions of privacy breach, encoding resistance to arbitrary side-information, and clarify their relationships with QIF and DP. We also study the asymptotic behavior of privacy breaches depending on the number of observations.

The notion of utility has been the subject of intensive investigation in the field of DP, see e.g. [22,18,1,2,3] and references therein. A general goal is that of designing mechanisms achieving optimal expected utility given a certain security level ϵ. Ghosh et al. [18] propose a model of expected utility based on user preferences, and show that both the geometric mechanism and its truncated version achieve universal optimality. Here we provide the growth rate of utility, and we highlight a difference between a mechanism and its truncated version, in the presence of repeated observations. Alvim et al. [1] have shown the tight connection between utility and Bayes risk, hence information leakage, in the case of a single observation. A different, somewhat stronger notion of utility, called *accuracy*, is considered by McSherry and Talwar [22]. They do not presuppose any user-specific prior over the set of possible answers; rather, they show that, in the exponential mechanism they propose, for any database, the expected *score* of the answer comes close to the maximum.

Structure of the Paper. The rest of the paper is organized as follows. In Section 2 we review some terminology and basic concepts about Bayesian hypothesis testing and information leakage. Section 3 characterizes the semantic security of randomization mechanisms, both in the worst and in the average case, but limited to a single observation on the part of the adversary. Section 4 discusses the relation between DP and our worst-case security. Section 5 discusses the asymptotic behavior of the security level in the case of n independent observations where the secret input remains fixed, again both in the worst and in the average case. In the worst case, we also offer a result characterizing the probability, depending on n, that *some* sequence of observations causes a breach. In Section 6 we deal with utility in the case of repeated observations. Section 7 discusses further work and draws some concluding remarks. Due to space limitations proofs have been omitted; they can be found in a full version available online [7].

2 Preliminaries

We review some notation and basic concepts about Bayesian hypothesis testing and information leakage.

2.1 Basic Terminology

Let $\mathcal{X}$ be a finite nonempty set. A probability distribution on $\mathcal{X}$ is a function $p : \mathcal{X} \to [0,1]$ such that $\sum_{x \in \mathcal{X}} p(x) = 1$. The *support* of p is defined as $\mathrm{supp}(p) \triangleq \{x \in \mathcal{X} | p(x) > 0\}$. For any $Q \subseteq \mathcal{X}$ we let $p(Q)$ denote $\sum_{x \in Q} p(x)$. Given $n \geq 0$, we let $p^n : \mathcal{X}^n \to [0,1]$ denote the n-th extension of p, defined as $p^n(x_1, \ldots, x_n) \triangleq \prod_{i=1}^{n} p(x_i)$; this is in turn a probability distribution on the set X^n. When $Q \subseteq \mathcal{X}^n$ and n is clear from the context, we shall abbreviate $p^n(Q)$ as just $p(Q)$. For $n = 0$, we set $p^0(\epsilon) = 1$, where ϵ denotes the empty tuple. $\Pr(\cdot)$ will generally denote a probability measure defined on some probability space (understood from the context). Given a random variable X taking values in $\mathcal{X}$, we write $X \sim p(x)$ if X is distributed according to $p(x)$, that is for each $x \in \mathcal{X}$, $\Pr(X = x) = p(x)$. We shall only consider discrete random variables. Suppose we are given random variables X, Y,... taking values in $\mathcal{X}$, $\mathcal{Y}$,... and defined on the same probability space. We shall use abbreviations such as $p(y|x)$ for $\Pr(Y = y|X = x)$, $p(y|Q)$ for $\Pr(Y = y|X \in Q)$, and so on, whenever no confusion arises about the involved random variables X and Y. Finally, when notationally convenient, we shall denote the conditional probability distribution on $\mathcal{Y}$ $p(\cdot|x)$ ($x \in \mathcal{X}$) as $p_x(\cdot)$. Randomization mechanisms are information-theoretic channels. The use of this concept in the field of QIF has been promoted by Chatzikokolakis, Palamidessi and collaborators [9,10,8]; the systems amenable to this form of representation are sometimes referred to as *information hiding systems* (see also [5,6]).

Definition 1 (randomization mechanism). *A* randomization mechanism *is a triple $\mathcal{R} = (\mathcal{X}, \mathcal{Y}, p(\cdot|\cdot))$, composed by a finite set of* inputs $\mathcal{X}$ *representing the secret information, a finite set of* observables $\mathcal{Y}$ *representing the observable values, and a* conditional probability matrix*, $p(\cdot|\cdot) \in [0,1]^{\mathcal{X}\times\mathcal{Y}}$, where each row sums up to 1.*

The entry of row x and column y of the channel's matrix will be written as $p(y|x)$, and represents the probability of observing y, given that x is the (secret) input of the system. For each x, the x-th row of the matrix is identified with the probability distribution on $\mathcal{Y}$ given by $y \mapsto p(y|x)$, which is denoted by p_x. We say $\mathcal{R}$ is *non-degenerate* if $x \neq x'$ implies $p_x \neq p_{x'}$, and *strictly positive* if $p(y|x) > 0$ for each x and y. Note that $p(\cdot)$ on $\mathcal{X}$ and the conditional probability matrix $p(y|x)$ together induce a probability distribution q on $\mathcal{X} \times \mathcal{Y}$ defined as $q(x,y) \triangleq p(x) \cdot p(y|x)$, hence a pair of discrete random variables $(X,Y) \sim q(x,y)$, with X taking values in $\mathcal{X}$ and Y taking values in $\mathcal{Y}$. Of course, one has $X \sim p(x)$ and, for each $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ s.t. $p(x) > 0$, $\Pr(Y = y|X = x) = p(y|x)$.

2.2 Bayesian Hypothesis Testing, Min-entropy, Leakage

Assume we are given a randomization mechanism $\mathcal{R} = (\mathcal{X}, \mathcal{Y}, p(\cdot|\cdot))$ and an a priori distribution $p(x)$ on $\mathcal{X}$. Assume an attacker wants to identify X on the basis of the observation Y, where, as explained above, $(X,Y) \sim p(x) \cdot p(y|x)$. This scenario can be formalized in terms of Bayesian hypothesis testing, as follows. The attacker's strategy is represented by a *guessing function* $g : \mathcal{Y} \to \mathcal{X}$. The *success probability after 1 observation* (relative to g) is defined as by

$$P^{(g)}_{succ} \triangleq \Pr(g(Y) = X)\,. \tag{1}$$

Correspondingly, the error probability is $P^{(g)}_e \triangleq 1 - P^{(g)}_{succ}$. It is well-known (see e.g. [12]) that optimal strategies, that is strategies maximizing the success probability, are those obeying the following *Maximum A Posteriori* (MAP) criterion: for each $y \in \mathcal{Y}$ and $x \in \mathcal{X}$ $g(y) = x$ implies $p(y|x)p(x) \geq p(y|x')p(x')$ $\forall x' \in \mathcal{X}$.In what follows, we shall always assume that g is MAP and consequently omit the superscript $^{(g)}$. The quantity P_{succ} admits a number of equivalent formulations. For example, it is straightforward to check that (cf. e.g. [26,5,6]; the sums below run over y of positive probability)

$$P_{succ} = \sum_y p(y) \max_x p(x|y) \tag{2}$$

$$= \sum_y \max_x p(y|x)p(x)\,. \tag{3}$$

Equation (2) shows clearly that P_{succ} results from an *average* over all observations $y \in \mathcal{Y}$. This equation also establishes a connection with Rényi's *min-entropy* [25]. This, for a random variable $X \sim p(x)$, is defined thus (in the following, all the log's are taken with base 2): $H_\infty(X) \overset{\triangle}{=} -\log \max_x p(x)$ .Conditional min-entropy of X given Y is defined as: $H_\infty(X|Y) \overset{\triangle}{=} -\log \sum_y p(y) \max_x p(x|y)$ .Therefore from (2)

$$P_{succ} = 2^{-H_\infty(X|Y)} . \tag{4}$$

Success probability is the key to defining *information leakage* of X given Y. This quantity expresses, in bits, how much, on the average, one observation increases the success probability of the attacker. The intuition is that a gain of one bit of leakage corresponds to doubling the a priori success probability: $\mathcal{L}(X;Y) \overset{\triangle}{=} H_\infty(X) - H_\infty(X|Y) = \log \frac{P_{succ}}{\max_x p(x)}$.

2.3 Asymptotic Behavior

The scenario of a single observation generalizes to the case of several, say n, independent observations as follows. Given a prior $p(x)$ and fixed any $n \geq 0$, the adversary gets to know the observations corresponding to n independent executions of the mechanism $\mathcal{R}$, say $y^n = (y_1, ..., y_n) \in \mathcal{Y}^n$, throughout which the secret state x is kept fixed. Formally, the adversary knows a random vector of observations $Y^n = (Y_1, ..., Y_n)$ such that, for each $i = 1, ..., n$, Y_i is distributed like Y and the individual Y_i are *conditionally independent given* X. That is, the following equality holds true for each $y^n \in \mathcal{Y}^n$ and $x \in \mathcal{X}$ s.t. $p(x) > 0$ $\Pr(Y^n = (y_1, \ldots, y_n) \,|\, X = x) = \prod_{i=1}^n p(y_i|x)$ .We will often abbreviate the right-hand side of the last expression as $p(y^n|x)$. Again, for any n, the attacker's strategy is modeled by a guessing function $g : \mathcal{Y}^n \to \mathcal{X}$; the optimal strategy, that we will assume throughout the paper, is when g is MAP. The corresponding success and error probabilities, which depend on n, will be denoted by P^n_{succ} and P^n_e, respectively[1]. It is quite expected that, as $n \to +\infty$, $P^n_{succ} \to 1$, and this is indeed the case, under very mild conditions. What is important, though, is to characterize *how fast* the probability of success approaches 1. Intuitively, we want be able to determine an exponent $\rho \geq 0$ such that, for large n, $P^n_{succ} \approx 1 - 2^{-n\rho}$. To this purpose, we introduce some concepts in what follows.

Let $\{a_n\}_{n\geq 0}$ be a sequence of nonnegative reals. Assume that $\tau = \lim_{n\to+\infty} a_n$ exists and that $a_n \leq \tau$ for each n. We define the *rate* of $\{a_n\}_{n\geq 0}$ as follows:

$$\text{rate}(\{a_n\}) \overset{\triangle}{=} \lim_{n\to+\infty} -\frac{1}{n}\log(\tau - a_n) \tag{5}$$

provided this limit exists[2]. When $\text{rate}(\{a_n\}) = \rho$ we also say that a_n *reaches* τ *at rate* ρ, and write this as $a_n \doteq \tau - 2^{-n\rho}$ .Intuitively, for large values on of n, this $\doteq$ can be interpreted as $\approx$. The above definition is modified as expected for the case when $a_n \geq \tau$ for each n: we set $\text{rate}(\{a_n\}) \overset{\triangle}{=} \lim_{n\to+\infty} -\frac{1}{n}\log(a_n - \tau)$ and write $a_n \doteq \tau + 2^{-n\rho}$if

[1] For the case $n = 0$, we set for uniformity $y^n \overset{\triangle}{=} \epsilon$ (empty tuple) and $p(\epsilon|x) \overset{\triangle}{=} 1$. With this choice, $P^0_{succ} = \max_x p(x)$.

[2] More generally, we define the upper-rate (resp. lower-rate) $\overline{\text{rate}}(\{a_n\})$ (resp. $\underline{\text{rate}}(\{a_n\})$) by replacing the lim in (5) by lim sup (resp. lim inf).

$\rho = \text{rate}(\{a_n\})$. Note that we do allow $\text{rate}(\{a_n\}) = +\infty$, a case that arises for example when $\{a_n\}_{n\geq 0}$ is a constant sequence.

The rate of growth of P^n_{succ} is given by *Chernoff Information*. Given two probability distributions p, q on $\mathcal{Y}$, we let their Chernoff Information be

$$C(p,q) \stackrel{\triangle}{=} -\min_{0\leq\lambda\leq 1} \log\Big(\sum_{y\in\text{supp}(p)\cap\text{supp}(q)} p^{\lambda}(y)q^{1-\lambda}(y)\Big) \tag{6}$$

where we stipulate that $C(p,q) = +\infty$ if $\text{supp}(p) \cap \text{supp}(q) = \emptyset$. Here $C(p,q)$ can be thought of as a sort of distance[3] between p and q: the more p and q are far apart, the the less observations are needed to discriminate between them. More precisely, assume we are in the binary case $\mathcal{X} = \{x_1, x_2\}$ (binary hypothesis testing) and let $p_i = p(\cdot|x_i)$ for $i = 1, 2$. Then a well-known result gives us the rate of convergence for the probabilities of success and error, with the proviso that $p(x_1) > 0$ and $p(x_2) > 0$ (cf. [12]): $P^n_{succ} \doteq 1-2^{-nC(p_1,p_2)}$ and $P^n_e \doteq 2^{-nC(p_1,p_2)}$(here we stipulate $2^{-\infty} = 0$). Note that this rate does not depend on the prior distribution $p(x)$ on $\{x_1, x_2\}$, but only on the probability distributions p_1 and p_2. This result extends to the general case $|\mathcal{X}| \geq 2$. Provided $\mathcal{R}$ is non-degenerate, it is enough to replace $C(p_1, p_2)$ by $\min_{x\neq x'} C(p_x, p_{x'})$, thus (see [5,20]):

$$P^n_{succ} \doteq 1 - 2^{-n\min_{x\neq x'} C(p_x,p_{x'})} \tag{7}$$

$$P^n_e \doteq 2^{-n\min_{x\neq x'} C(p_x,p_{x'})} \tag{8}$$

(with the understanding that, in the min, $p(x) > 0$ and $p(x') > 0$).

3 Semantic Security of Randomization Mechanisms

We shall consider two scenarios. In the worst-case scenario, one is interested in the seriousness of a breach, independently of how much the breach is likely; this is also the scenario underlying differential privacy, which we will examine in Section 6. In the average-case scenario, one considers, so to speak, the seriousness of the breach averaged on the probability of the observed Y. In each scenario, our aim is to characterize when a randomization mechanism can be considered secure both in a semantic and in an operational fashion. We fix a generic randomization mechanism $\mathcal{R}$ for the rest of the section.

3.1 The Worst-Case Scenario

In the worst-case definition, we compare the probability of predicates $Q \subseteq \mathcal{X}$ of the inputs, prior and posterior to one observation $y \in \mathcal{Y}$: a large variation in the posterior probability relative to any y implies a breach. Note that even the situation when the posterior probability is small compared to the prior is considered as dangerous, as it tells the adversary that $X \in Q^c$ is likely.

[3] Note that $C(p,q) = 0$ iff $p = q$ and that $C(p,q) = C(q,p)$. However $C(\cdot,\cdot)$ fails to satisfy the triangle inequality.

Definition 2 (worst-case breach). *Let $\epsilon \geq 0$. A ϵ-*breach *(*privacy breach of level *ϵ) for $\mathcal{R}$ is a subset $Q \subseteq \mathcal{X}$ such that for some a priori probability distribution $p(x)$ on $\mathcal{X}$, we have $p(Q) > 0$ and*

$$\max_{p(y)>0} |\log \frac{p(Q|y)}{p(Q)}| > \epsilon\,.$$

*$\mathcal{R}$ is ϵ-*secure *if it has no breach of level ϵ. The* security level *of $\mathcal{R}$ is defined as $\epsilon_{\mathcal{R}} \overset{\triangle}{=} \inf\{\epsilon \geq 0 : \mathcal{R}$ is ϵ-secure$\}$.*

If $|\log \frac{p(Q|y)}{p(Q)}| > \epsilon$, we say *$y$ causes a Q-breach of level ϵ.*

Remark 1. Note that the condition $\max_y |\log \frac{p(Q|y)}{p(Q)}| > \epsilon$ can be equivalently reformulated as $\max_y \max\{\frac{p(Q|y)}{p(Q)}, \frac{p(Q)}{p(Q|y)}\} > 2^{\epsilon}$.

For each $y \in \mathcal{Y}$, let $\pi_{M,y}$ and $\pi_{m,y}$ be the maximum and the minimum in the column y of the matrix $p(\cdot|\cdot)$, respectively. We give the following operational characterization of ϵ-security. A similar property (*amplification*) was considered as a sufficient condition for the absence of breaches in [15]. In the theorem below, we stipulate that $\frac{\pi_{M,y}}{\pi_{m,y}} = +\infty$ if $\pi_{M,y} > 0$ and $\pi_{m,y} = 0$.

Theorem 1. *$\mathcal{R}$ is ϵ-secure iff* $\log \max_y \frac{\pi_{M,y}}{\pi_{m,y}} \leq \epsilon$.

Example 1. The following example is inspired by [15]. The private information is represented by the set of integers $\mathcal{X} = \{0, ..., 5\}$, and $\mathcal{Y} = \mathcal{X}$. We consider a mechanism that replaces any $x \in \mathcal{X}$ by a number y that retains some information about the original x. More precisely, we let $Y = \lfloor X + \xi \rfloor \mod 6$, where with probability 0.5 ξ is a chosen uniformly at random in $\{-\frac{1}{2}, \frac{1}{2}\}$, and with probability 0.5 it is chosen uniformly at random in $\mathcal{X}$. We can easily compute the resulting conditional probability matrix.

$$\begin{pmatrix} 0.2500 & 0.2500 & 0.0833 & 0.0833 & 0.0833 & 0.2500 \\ 0.2500 & 0.2500 & 0.2500 & 0.0833 & 0.0833 & 0.0833 \\ 0.0833 & 0.2500 & 0.2500 & 0.2500 & 0.0833 & 0.0833 \\ 0.0833 & 0.0833 & 0.2500 & 0.2500 & 0.2500 & 0.0833 \\ 0.0833 & 0.0833 & 0.0833 & 0.2500 & 0.2500 & 0.2500 \\ 0.2500 & 0.0833 & 0.0833 & 0.0833 & 0.2500 & 0.2500 \end{pmatrix}.$$

The security level of this mechanism is $\epsilon_R = \log \frac{0.25}{0.083} = 3.0012$.

3.2 The Average-Case Scenario

We want to asses the security of $\mathcal{R}$ by comparing the prior and posterior success probability for an adversary wanting to infer whether the secret is in Q or not after observing Y. This will give us an *average* measure of the seriousness of the breach induced by Q.

Fix a prior probability distribution $p(x)$ on $\mathcal{X}$. For every nonempty $Q \subseteq \mathcal{X}$, we shall denote by $\hat{Q}$ the binary random variable $I_Q(X)$, where $I_Q : \mathcal{X} \to \{0, 1\}$ is the indicator function of Q – in this notation, the dependence from $p(x)$ is left implicit, as $p(x)$ will always be clear from the context. An adversary, after observing Y, wants to determine whether it holds $\hat{Q}$ or $\hat{Q}^c$. This is a binary Bayesian hypothesis testing problem, which, as seen in Section 2, can be formulated in terms of min-entropy.

Definition 3 (Average-case breach). *Let* $\epsilon \geq 0$. *A* ϵ-A-breach (average case breach of level ϵ) *of* $\mathcal{R}$ *is a* $Q \subseteq \mathcal{X}$ *s.t. for some a priori distribution* $p(x)$ *on* $\mathcal{X}$*, we have that* $p(Q) > 0$ *and* $\mathcal{L}(\hat{Q}; Y) = H_\infty(\hat{Q}) - H_\infty(\hat{Q}|Y) > \epsilon$. $\mathcal{R}$ *is* ϵ-A-secure *if it has no average case breach of level* ϵ. *The* A-security level *of* $\mathcal{R}$ *is defined as* $\epsilon^A_{\mathcal{R}} \triangleq \inf\{\epsilon \geq 0 :$ $\mathcal{R}$ *is* ϵ*-A-secure*$\}$.

Of course, Y leaks at most one bit about the truth of Q: $0 \leq \mathcal{L}(\hat{Q}; Y) \leq 1$. In the next theorem, recall that for each $x \in \mathcal{X}$, $p_x(\cdot)$ denotes the distribution $p(\cdot|x)$.

Theorem 2. *Let* $l \triangleq \max_{x,x'} ||p_x - p_{x'}||_1$ *and* $\epsilon \geq 0$. *Then* $\mathcal{R}$ *is* ϵ*-A-secure iff* $\log(\frac{l}{2} + 1) \leq \epsilon$.

4 Worst-Case Security vs. Differential Privacy

We first introduce DP, then discuss its relation to worst-case security. The definition of differential privacy relies on a notion of "neighborhood" between inputs of an underlying randomization mechanism. In the original formulation, two neighbors x and x' are two database instances that only differ by one entry. More generally, one can rely upon a notion of *adjacency*. An undirected graph is a pair (V, E) where V is a set of nodes and E is a set of unordered pairs $\{u, v\}$ with $u, v \in V$ and $u \neq v$. We also say that E is an adjacency relation on V and if $v \sim v'$ say v and v' are adjacent.

Definition 4 (differential privacy). *A* differentially private mechanism $\mathcal{D}$ *is a pair* $(\mathcal{R}, \sim)$ *where* $\mathcal{R} = (\mathcal{X}, \mathcal{Y}, p(\cdot|\cdot))$ *is a randomization mechanism and* $\sim$ *is an adjacency relation on* $\mathcal{X}$*, that is,* $(\mathcal{X}, \sim)$ *forms an undirected graph.*

Let $\epsilon > 0$. *We say* $\mathcal{D}$ *provides* ϵ-differential privacy *if for each* $x, x' \in \mathcal{X}$ *s.t.* $x \sim x'$*, it holds that for each* $y \in \mathcal{Y}$*:*

$$\max_y |\log \frac{p(y|x)}{p(y|x')}| \leq \epsilon\,. \tag{9}$$

Note that condition (9) is exactly that given in Theorem 1 to characterize worst-case privacy breaches, but limited to pairs of adjacent rows x and x'. This prompts the question of the exact relationship between the two notions. In the rest of the section, we will consider the standard domain $\mathcal{X} = \{0, 1\}^n$ of *databases*, corresponding to subsets of a given set of individuals $\{1, ..., n\}$. We deem two databases x, x' adjacent if they differ for the value of exactly one individual, that is if their Hamming distance is 1 [14,3,1]. Throughout the section, we let $\mathcal{D} = (\mathcal{R}, \sim)$ be a generic mechanism equipped with this $\mathcal{X}$ and this adjacency relation. Moreover, we will denote by Q_i ($i \in \{1, ..., n\}$) the set of databases $\{x \in \mathcal{X} \,|\, x_i = 1\}$, that is databases containing individual i.

The following theorem provides a precise characterization of (worst-case) ϵ-security in terms of privacy of individuals: interaction with the mechanism does not significantly change the belief about the participation of any individual to the database.

Theorem 3. $\mathcal{R}$ *satisfies* ϵ*-security iff for each* $i \in \{1, ..., n\}$ *and prior* $p(\cdot)$*,* Q_i *is not an* ϵ*-breach.*

Remark 2. The above theorem is of course still valid if one strengthens the "only if" part by requiring that *both* Q_i *and* Q_i^c are not ϵ-breach.

We proceed now by linking (worst-case) ϵ-security to ϵ-DP. The next result sets limits to the "arbitrariness" of background information against which DP offers guarantees: for example, it fails in some cases where an adversary has sufficient background information to rule out all possible databases but two, which are substantially different from each other.

Theorem 4. *If $\mathcal{R}$ satisfies ϵ-security then $\mathcal{D} = (\mathcal{R}, \sim)$ provides ϵ-DP. On the contrary, for each n there exist mechanisms providing ϵ-DP but not ϵ-security; in particular, these mechanisms exhibit Q_i-breaches ($i \in \{1, ..., n\}$) of level arbitrarily close to $n\epsilon > \epsilon$.*

Example 2. Let us consider the mechanism with input domain $\mathcal{X} = \{0, 1\}^2$, corresponding to the following matrix:

$$\begin{pmatrix} \frac{2}{3} & \frac{1}{6} & \frac{1}{12} & \frac{1}{64} & \frac{1}{48} & \frac{1}{48} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & \frac{1}{12} & \frac{1}{24} & \frac{1}{24} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & \frac{1}{6} & \frac{1}{12} & \frac{1}{12} \\ \frac{1}{12} & \frac{1}{12} & \frac{1}{6} & \frac{1}{3} & \frac{1}{6} & \frac{1}{6} \end{pmatrix}.$$

This mechanism provides ϵ-DP with $\epsilon = 1$. However, it is not ϵ-secure, as e.g. $\frac{2/3}{1/12} = 8 > 2^{\epsilon}$.

We recover coincidence between ϵ-security and ϵ-DP if we confine ourselves to background knowledge that can be factorised as the product of independent measures over individuals. This provides another characterization of ϵ-DP. In what follows, for any $x \in \mathcal{X}$, we denote by $x_{\setminus i}$ the element of $\{0, 1\}^{n-1}$ obtained by removing the i-th component from x.

Theorem 5. *The following statements are equivalent:*

1. *$\mathcal{D}$ satisfies ϵ-DP;*
2. *for each $i \in \{1, ..., n\}$ and $p(x)$ of the form $p_i(x_i)q(x_{\setminus i})$, Q_i is not an ϵ-breach;*
3. *for each $p(x)$ of the form $\prod_{j=1}^{n} p_j(x_j)$ and for each $i \in \{1, ..., n\}$, Q_i is not an ϵ-breach.*

5 Asymptotic Security

We assume that the attacker collects a tuple $y^n = (y_1, y_2, \ldots, y_n) \in \mathcal{Y}^n$ of observations generated i.i.d from the mechanism $\mathcal{R}$. We expect that, given Q, as n grows, the breach level approaches a threshold value. In order to the characterize synthetically the security of the randomization mechanism, though, it is important to characterize *how fast* this threshold is approached. Again, we distinguish a worst- from an average-case scenario and, for the rest of the section, fix a generic randomization mechanism $\mathcal{R}$.

5.1 Worst-Case Scenario

We begin with an obvious generalization of the notion of breach.

Definition 5 **(n-breach of level ϵ).** *A* (n, ϵ)-privacy breach *is a subset $Q \subseteq \mathcal{X}$ s.t. for some prior distribution $p(x)$ on $\mathcal{X}$, we have that $p(Q) > 0$ and*

$$\max_{p(y^n)>0} |\log \frac{p(Q|y^n)}{p(Q)}| > \epsilon\,.$$

The next proposition says that a notion of security based on bounding the level of n-breaches is not achievable. For the sake of simplicity, we shall discuss some of the following results in the case $\mathcal{R}$ is non-degenerate (all rows of the matrix are distinct).

Proposition 1. *Assume $\mathcal{R}$ is non-degenerate. For n large enough, $\mathcal{R}$ has n-breaches of arbitrary level. More explicitly, for any nonempty $Q \subseteq \mathcal{X}$ and any $\epsilon \geq 0$ there is a prior distribution $p(x)$ s.t. for any n large enough there is y^n ($p(y^n) > 0$) such that $\log \frac{p(Q|y^n)}{p(Q)} > \epsilon$.*

The above proposition suggests that, in the case of a large number of observations, worst-case analysis should focus on how fast $p(Q|y^n)$ can grow, rather than looking at the maximum level of a breach.

Definition 6 **(rate of a breach).** *Let $\rho \geq 0$. A* breach of rate ρ *is a subset $Q \subseteq \mathcal{X}$ such that there exist a prior distribution $p(x)$ on $\mathcal{X}$ with $p(Q) > 0$ and a sequence of n-tuples, $\{y^n\}_{n\geq 0}$, with $p(y^n) > 0$, such that $p(Q|y^n) \doteq 1 - 2^{-n\rho'}$ with $\rho' > \rho$. A randomization mechanism is* ρ-rate secure *if it has no privacy breach of rate ρ. The* rate security level *is defined as $\rho_{\mathcal{R}} \stackrel{\triangle}{=} \inf\{\rho \geq 0 : \mathcal{R}$ is ρ-rate secure$\}$.*

Theorem 6. *$\mathcal{R}$ is ρ-rate secure iff $\rho \geq \log \max_y \frac{\pi_{M,y}}{\pi_{m,y}}$.*

The above theorem says that, for large n, the seriousness of the breach, for certain y^n, can be as bad as $\approx \log \frac{1-(\frac{\pi_m}{\pi_M})^n}{p(Q)}$. The result, however, does not tell us *how likely* a serious breach is depending on n. The next result shows that the probability that *some* observable y^n causes a Q-breach grows exponentially fast. We premise some notation.

Fix a prior $p(x)$ over $\mathcal{X}$. Recall that we let $X \sim p(x)$ denote a random variable representing the secret information, and $Y^n = (Y_1, ..., Y_n)$ be the corresponding random vector of n observations, which are i.i.d. given X. Let us fix $Q \subseteq \mathcal{X}$ s.t. $p(Q) > 0$. Then $p(Q|Y^n)$ is a random variable. For any fixed $\epsilon > 0$, let us consider the two events

$$Breach_n^\epsilon \stackrel{\triangle}{=} \{\frac{p(Q|Y^n)}{p(Q)} > 2^\epsilon\} \quad \text{and} \quad \overline{Breach_n}^\epsilon \stackrel{\triangle}{=} \{\frac{p(Q)}{p(Q|Y^n)} > 2^\epsilon\}\,.$$

Clearly, the event $Breach_n^\epsilon \cup \overline{Breach_n}^\epsilon$ is the event that Y^n causes a Q-breach of level ϵ. As n grows, we expect that the probability of this event approaches 1 quite fast. The next theorem tells us exactly how fast.

Theorem 7. *Assume $\mathcal{R}$ is non-degenerate and strictly positive. Then, with the notation introduced above*

$$\Pr(Breach_n^\epsilon|X \in Q) \doteq 1 - 2^{-nC} \quad \textit{and} \quad \Pr(\overline{Breach_n}^\epsilon|X \in Q^c) \doteq 1 - 2^{-nC}$$

where $C = \min_{x\in Q, x'\in Q^c} C(p_x, p_{x'})$, with the understanding that x and x' in the min *are taken of positive probability. As a consequence, the probability that Y^n causes a Q-breach reaches 1 at rate at least C.*

5.2 Average-Case Scenario

It is straightforward to extend the definition of average-case breach to the case with multiple observations. For any nonempty subset $Q \subseteq \mathcal{X}$, and random variable $X \sim p(x)$, s.t. $p(Q) > 0$, we consider $\hat{Q} = I_Q(X)$ and define the leakage imputable to Q after n observations as

$$\mathcal{L}^n(\hat{Q}; Y^n) \triangleq H_\infty(\hat{Q}) - H_\infty(\hat{Q}|Y^n).$$

An n-breach of level $\epsilon \geq 0$ is a Q such that $\mathcal{L}^n(\hat{Q}; Y^n) > \epsilon$. Recall from (4) that $P^n_{succ} = 2^{-H_\infty(\hat{Q}|Y^n)}$ is the success probability of guessing between $p(\cdot|Q)$ and $p(\cdot|Q^c)$ after observing Y^n. Provided $p(\cdot|Q) \neq p(\cdot|Q^c)$, (7) implies that, as $n \to +\infty$ we have $P^n_{succ} \to 1$, hence $\mathcal{L}^n(\hat{Q}; Y^n) \to -\log \max\{p(Q), 1 - p(Q)\}$. If $p(\cdot|Q) = p(\cdot|Q^c)$ then P^n_{succ} is constantly $\max\{p(Q), 1 - p(Q)\}$, so that the observations give no advantage to the attacker. These remarks suggest that, in the case of repeated observations, it is again important to characterize how fast $P^n_{succ} \to 1$.

Definition 7 (rate of a breach - average case). *Let $\rho \geq 0$. An* A-breach of rate ρ *is a subset $Q \subseteq \mathcal{X}$ such that for some prior distribution $p(x)$ on $\mathcal{X}$ with $p(Q) > 0$ one has that $P^n_{succ} \doteq 1 - 2^{-n\rho'}$, for some $\rho' > \rho$. A randomization mechanism is* ρ-rate A-secure *if it has no privacy breach of rate ρ. The* rate A-security level *is defined as $\rho^A_{\mathcal{R}} \triangleq \inf\{\rho \geq 0 : \mathcal{R}$ is ρ-rate A-secure$\}$.*

Now we can proceed with the following theorem.

Theorem 8. *$\mathcal{R}$ is ρ-rate A-secure iff* $\max_{x,x'} C(p_x, p_{x'}) \leq \rho$.

6 Utility

We next turn to the study of utility. In the rest of the section, we fix a mechanism $\mathcal{R}$ and a prior distribution $p(\cdot)$. Without any significant loss of generality, we shall assume that $\mathcal{R}$ is strictly positive and that $\mathrm{supp}(p) = \mathcal{X}$. Moreover, in this section, we shall work under the more general assumption that $\mathcal{Y}$ is *finite or denumerable*.

For any $n \geq 1$, we are now going to define the expected utility of $\mathcal{R}$, depending on user-specific belief, modeled as a prior $p(\cdot)$ on $\mathcal{X}$, and on function $loss : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$. Here, $loss(x, x')$ represents the loss of a user who interprets the result of an observation of $\mathcal{R}$ as x', given that the real answer is x. For the sake of simplicity, we shall assume that $loss$ achieves a proper minimum when $x = x'$: for each $x \neq x'$, $loss(x, x) < loss(x, x')$. We also presuppose a *guessing function* $g : \mathcal{Y}^n \to \mathcal{X}$. The *expected utility* of $\mathcal{D}$ – relative to g – after n observations is in fact defined as an expected loss (the lower the better), thus

$$U_n \triangleq \sum_x p(x) \sum_{y^n} p(y^n|x) loss(x, g(y^n)) . \tag{10}$$

Note that this definition coincides with that of Ghosh et al. [18] when one interprets our guessing function g as the *remap* considered in [18]. This is also the utility model of Alvim et al. [3], modulo the fact they only consider the 0/1-loss, or better, the complementary gain.

Example 3. When $\mathcal{Y}$ is a subset of the reals, legal *loss* functions include the absolute value error $loss(x,x') = |x' - x|$ and the squared error $loss(x,x') = (x'-x)^2$. The binary loss function defined as 0 if $x = x'$ and 1 otherwise is another example: the resulting expected loss is just error probability, $U_n = P_e^n$.

It is quite easy to argue that, since a proper minimum in $loss(x,x')$ is reached when $x = x'$, the utility is maximized asymptotically when g respects the MAP criterion: $p(g(y^n)|y^n) \geq p(x|y^n)$ for each $x \in \mathcal{X}$. In what follows, we just assume that g is a fixed MAP function. Below, we study the behavior of utility in relation to differential privacy. The crucial quantity is

$$\rho_{\mathcal{R}} \stackrel{\triangle}{=} \min_{x,x' \in \mathcal{X}, p_x \neq p_{x'}} C(p_x, p_{x'})\,. \tag{11}$$

We will show that the the asymptotic rate of utility is determined solely by $\rho_{\mathcal{R}}$. Note that this quantity does not depend on the user-defined *loss* function, nor on the prior $p(\cdot)$. For the sake of simplicity, below we discuss the result only in the case when $\mathcal{R}$ is non-degenerate[4].

Remark 3. We note that the formula (6) for Chernoff Information extends to the case when $p(\cdot)$ and $q(\cdot)$ have the same *denumerable* support.

Theorem 9. *Assume $\mathcal{R}$ is non-degenerate. Then $U_n \doteq U_{\mathcal{R}} + 2^{-n\rho_{\mathcal{R}}}$, where $U_{\mathcal{R}} \stackrel{\triangle}{=} \sum_x p(x)loss(x,x)$.*

Having established the centrality of $\rho_{\mathcal{R}}$ in the asymptotic behavior of utility, we now discuss the relationship of this quantity with the worst-case security level ϵ provided by the mechanism. The first result provides us with a simple, general bound relating $\rho_{\mathcal{R}}$ and ϵ.

Theorem 10. *Assume $\mathcal{R}$ is worst-case ϵ-secure. Then $\rho_{\mathcal{R}} \leq \epsilon$. The same conclusion holds if $\mathcal{D} = (\mathcal{R}, \sim)$ provides ϵ-DP.*

In what follows, we will obtain more precise results relating ϵ to the utility rate $\rho_{\mathcal{R}}$ in the case of a class of mechanisms providing ϵ-DP. Specifically, we will consider mechanisms with a finite input domain $\mathcal{X} = \{0,1,\ldots,N\}$, a denumerable $\mathcal{Y} = \mathbb{Z}$ and a conditional probability matrix of the form $p_i(j) = Mc^{|i-j|}$, for some positive $c < 1$. This class of mechanisms includes the geometric mechanism (a discrete variant of the Laplacian mechanism, see [18]) and also a version extended to $\mathbb{Z}$ of the optimal mechanism considered by Alvim et al. [3].

Theorem 11. *Let $\mathcal{R}$ be a mechanism as described above. Then $\rho_{\mathcal{R}} = \log(1+c) - \frac{1}{2}\log c - 1$.*

Remark 4. The geometric mechanism is obtained by equipping the above described mechanism with the the line topology over $\mathcal{X} = \{0,...,N\}$: $i \sim j$ iff $d_{ij} \stackrel{\triangle}{=} |i-j| = 1$. This is the topology for counting queries in "oblivious" mechanisms, for example. If we set $c = 2^{-\epsilon}$, then this mechanism provides ϵ-DP. The above theorem tells us that in this case $\rho_{\mathcal{R}} = \frac{\epsilon}{2} + \log \frac{1+2^{-\epsilon}}{2}$. By setting e.g. $\epsilon = 1$, one gets $\rho_{\mathcal{R}} \approx 0.085$.

[4] The result carries over to the general case, at the cost of some notational burden: one has to replace $U_{\mathcal{R}}$ with a more complicated expression.

For any mechanism $\mathcal{R}$ with input $\mathcal{X} = \{0, ..., N\}$ and output $\mathcal{Y} = \mathbb{Z}$, we can consider the corresponding *truncated* mechanism $\mathcal{R}'$: it has $\mathcal{X} = \mathcal{Y} = \{0, 1, \ldots, N\}$ and its matrix is obtained from $\mathcal{R}$'s by summing all the columns $y < 0$ to column $y = 0$, and all the columns $y > N$ to column $y = N$.

Corollary 1. *Assume $\mathcal{R}'$ is the truncated version of a mechanism $\mathcal{R}$. Then $\rho_{\mathcal{R}'} < \rho_{\mathcal{R}}$.*

In the case of a single observation case, treated by Ghosh et al. [18], there is no substantial difference between the geometric mechanism and the truncated geometric one. Corollary 1 shows that the situation is different in the case with repeated observations.

7 Conclusion and Further Work

We have analyzed security of randomization mechanisms against privacy breaches with respect to various dimensions (worst vs. average case, single vs. repeated observations, utility). Whenever appropriate, we have characterized the resulting security measures in terms of simple row-distance properties of the underlying channel matrix. We have clarified the relation our worst-case measure with DP.

A problem left open by our study is the exact relationship between our average-case security notion and the maximum leakage considered in QIF – see e.g. [19]. We would also like to apply and possibly extend the results of the present paper to the setting of de-anonymization attacks on dataset containing micro-data. [23] has shown that the effectiveness of these attacks depends on certain features of sparsity and similarity of the dataset, which roughly quantify how difficult it is to find two rows of the dataset that are similar. The problem can be formalized in terms of randomization mechanisms with repeated observations – see [6] for some preliminary results on this aspect. Then the row-distance measures considered in the present paper appear to be strongly related to the notion of similarity, and might play a crucial in the formulation of a robust definition of dataset security.

References

1. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential Privacy: On the Trade-Off between Utility and Information Leakage. In: Barthe, G., Datta, A., Etalle, S. (eds.) FAST 2011. LNCS, vol. 7140, pp. 39–54. Springer, Heidelberg (2012)
2. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Palamidessi, C.: Quantitative Information Flow and Applications to Differential Privacy. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2011. LNCS, vol. 6858, pp. 211–230. Springer, Heidelberg (2011)
3. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Palamidessi, C.: On the Relation between Differential Privacy and Quantitative Information Flow. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 60–76. Springer, Heidelberg (2011)
4. Barthe, G., Köpf, B.: Information-theoretic Bounds for Differentially Private Mechanisms. In: 24rd IEEE Computer Security Foundations Symposium, CSF 2011, pp. 191–204. IEEE Computer Society (2011)
5. Boreale, M., Pampaloni, F., Paolini, M.: Asymptotic Information Leakage under One-Try Attacks. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 396–410. Springer, Heidelberg (2011), Full version to appear on *MSCS* available at `http://rap.dsi.unifi.it/~boreale/Asympt.pdf`

6. Boreale, M., Pampaloni, F., Paolini, M.: Quantitative Information Flow, with a View. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 588–606. Springer, Heidelberg (2011)
7. Boreale, M., Paolini, M.: Worst- and average-case privacy breaches in randomization mechanisms. Full version of the present paper, http://rap.dsi.unifi.it/~boreale/FullBreach.pdf
8. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative Notions of Leakage for One-try Attacks. In: Proc. of MFPS 2009. Electr. Notes Theor. Comput. Sci, vol. 249, pp. 75–91 (2009)
9. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Information and Computation 206(2-4), 378–401 (2008)
10. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the Bayes risk in information-hiding protocols. Journal of Computer Security 16(5), 531–571 (2008)
11. Chaum, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. Journal of Cryptology 1(1), 65–75 (1988)
12. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. John Wiley Sons (2006)
13. Dwork, C.: Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
14. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating Noise to Sensitivity in Private Data Analysis. In: Proc. of the 3rd IACR Theory of Cryptography Conference (2006)
15. Evfimievski, A., Gehrke, J., Srikant, R.: Limiting Privacy Breaches in Privacy Preserving Data Mining. In: Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (2003)
16. Friedman, A., Shuster, A.: Data Mining with Differential Privacy. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD (2010)
17. Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition Attacks and Auxiliary Information in Data Privacy. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD (2008)
18. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: STOC 2009, pp. 351–360 (2009)
19. Köpf, B., Smith, G.: Vulnerability Bounds and Leakage Resilience of Blinded Cryptography under Timing Attacks. In: CSF 2010, pp. 44–56 (2010)
20. Leang, C.C., Johnson, D.H.: On the asymptotics of M-hypothesis Bayesian detection. IEEE Transactions on Information Theory 43, 280–282 (1997)
21. McSherry, F.: Privacy Integrated Queries. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD (2009)
22. McSherry, F., Talwar, K.: Mechanism Design via Differential Privacy. In: Proceedings Annual IEEE Symposium on Foundations of Computer Science, FOCS (2007)
23. Narayanan, A., Shmatikov, V.: Robust De-anonymization of Large Sparse Datasets. In: Proc. of IEEE Symposium on Security and Privacy (2008)
24. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for Web Transactions. ACM Trans. Inf. Syst. Secur. 1(1), 66–92 (1998)
25. Rényi, A.: On Measures of Entropy and Information. In: Proc. of the 4th Berkeley Symposium on Mathematics, Statistics, and Probability, pp. 547–561 (1961)
26. Smith, G.: On the Foundations of Quantitative Information Flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)

Weak Bisimulations for Coalgebras over Ordered Functors

Tomasz Brengos[⋆]

Faculty of Mathematics and Information Sciences
Warsaw University of Technology
Koszykowa 75
00-662 Warszawa, Poland
t.brengos@mini.pw.edu.pl

Abstract. The aim of this paper is to introduce a coalgebraic setting in which it is possible to generalize and compare the two known approaches to defining weak bisimulation for labelled transition systems. We introduce two definitions of weak bisimulation for coalgebras over ordered functors, show their properties and give sufficient conditions for them to coincide. We formulate a weak coinduction principle.

Keywords: coalgebra, bisimulation, saturator, weak bisimulation, weak coinduction.

1 Introduction

The notion of a strong bisimulation for different transition systems plays an important role in theoretical computer science. A weak bisimulation is a relaxation of this notion by allowing silent, unobservable transitions. It is a well established notion for many deterministic and probabilistic transition systems (see [1], [4], [7], [8]). For many state-based systems one can equivalently introduce weak bisimulation in two different ways one of which has computational advantages over the other. To be more precise we will demonstrate this phenomenon on labelled transition systems. By a *labelled transition system* (or *LTS* in short) we mean a tuple $\langle A, \Sigma, \to \rangle$, where A is a set of *states*, Σ is a non-empty set called an *alphabet* and $\to$ is a subset of $A \times \Sigma \times A$ and is called a *transition*. For an LTS $\langle A, \Sigma, \to \rangle$ and $s \in \Sigma$ we define a relation on A by

$$\xrightarrow{s} := \{(a, a') \in A^2 \mid (a, s, a') \in \to\}.$$

For a fixed alphabet letter $\tau \in \Sigma$, representing a silent, unobservable transition label, and an LTS $\langle A, \Sigma, \to \rangle$ let $\xrightarrow{\tau^*}$ be the reflexive and transitive closure of the relation $\xrightarrow{\tau}$. The following definition of a weak bisimulation for LTS can be found in [4].

⋆ This work has been supported by the European Union in the framework of European Social Fund through the Warsaw University of Technology Development Programme and the grant of Warsaw University of Technology no. 504M for young researchers.

Definition 1. *A relation* $R \subseteq A \times A$ *is a* weak bisimulation *if it satisfies the following conditions. For* $(a,b) \in R$ *and* $\sigma \neq \tau$ *if* $a \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau^*} a'$ *then there is* $b' \in A$ *such that* $b \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau^*} b'$ *with* $(a',b') \in R$ *and conversely, for* $b \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau^*} b''$ *there is* $a'' \in A$ *such that* $a \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau^*} a''$ *and* $(a'',b'') \in R$. *Moreover, for* $\sigma = \tau$ *if* $a \xrightarrow{\tau}^* a'$ *then* $b \xrightarrow{\tau}^* b'$ *for some* $b' \in B$ *with* $(a',b') \in R$ *and conversely, if* $b \xrightarrow{\tau}^* b''$ *then* $a \xrightarrow{\tau}^* a''$ *for some* $a'' \in A$ *and* $(a'',b'') \in R$.

It is easily shown we can equivalently restate the definition of a weak bisimulation as follows.

Definition 2. *A relation* $R \subseteq A \times A$ *is a weak bisimulation if it satisfies the following condition. If* $(a,b) \in R$ *then for* $\sigma \neq \tau$ *if* $a \xrightarrow{\sigma} a'$ *then* $b \xrightarrow{\tau}^* \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau}^* b'$ *for some* $b' \in A$ *and* $(a',b') \in R$, *for* $\sigma = \tau$ *if* $a \xrightarrow{\tau} a'$ *then* $b \xrightarrow{\tau}^* b'$ *for some* $b' \in A$ *and* $(a',b') \in R$, *moreover for* $\sigma \neq \tau$ *if* $b \xrightarrow{\sigma} b'$ *then* $a \xrightarrow{\tau}^* \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau}^* a'$ *for some* $a' \in A$ *and* $(a',b') \in R$, *for* $\sigma = \tau$ *if* $b \xrightarrow{\tau} b'$ *then* $a \xrightarrow{\tau}^* a'$ *for some* $a' \in A$ *and* $(a',b') \in R$.

From the point of view of computation and automatic reasoning the latter approach to defining weak bisimulation is better since, unlike the former, it does not require the knowledge of the full saturated transition. Indeed, in order to show that two states $a, b \in A$ of a labelled transition system $\langle A, \Sigma, \rightarrow \rangle$ are weakly bisimilar in the sense of Definition 1 one needs to consider all states $a' \in A$ reachable from a via the saturated transitions $\xrightarrow{\tau}^* \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau}^*$ or $\xrightarrow{\tau}^*$ and compare them with similar states reachable from b. Whereas, to prove that two states $a, b \in A$ are weakly bisimilar in the sense of Definition 2 one needs to consider all states reachable from a via single step transitions $\xrightarrow{\sigma}$ and compare them with some states reachable from b via the saturated transitions.

The notion of a strong bisimulation, unlike the weak bisimulation, has been well captured coalgebraically (see e.g. [2],[11]). Different approaches to defining weak bisimulations for coalgebras have been presented. The earliest paper is [10], where the author studies weak bisimulations for while programs. In [5] the author introduces a definition of weak bisimulation for coalgebras by translating a coalgebraic structure into an LTS. This construction works for coalgebras over a large class of functors but does not cover the distribution functor, hence it is not applicable to different types of probabilistic systems. In [6], weak bisimulations are introduced via weak homomorphisms. As noted in [9] this construction does not lead to intuitive results for probabilistic systems. Finally, in [9] the authors present a definition of weak bisimulation for classes of coalgebras over functors obtained from bifunctors. Here, weak bisimulation of a system is defined as a strong bisimulation of a transformed system. First of all, it is worth noting that, although very interesting, neither of the approaches cited above expresses coalgebraically the computational advantages of Definition 2 over Definition 1. Secondly, all of them require to explicitly work with observable and unobservable part of the coalgebraic structure. The method of defining weak bisimulation presented in this paper only requires that a saturator is given and no explicit knowledge of silent and visible part of computation is neccessary.

The aim of this paper is to introduce a coalgebraic setting in which we can define weak bisimulation in two ways generalizing Definition 1 and Definition 2 and compare them. Additionally, we formulate a weak coinduction principle. The paper is organized as follows. In Section 2 we present basic definitions and properties from known universal coalgebra. In Section 3 we present a definition of a saturator and present some natural and well-known examples of saturators. In Section 4 we give two approaches to defining weak bisimulation via saturators and show their properties. Finally, Section 5 is devoted to formulation of a weak coinduction rule.

2 Basic Notions and Properties

Let Set be the category of all sets and mappings between them. Let $F\colon \mathsf{Set} \to \mathsf{Set}$ be a functor. An *F-coalgebra* is a tuple $\langle A, \alpha \rangle$, where A is a set and α is a mapping $\alpha\colon A \to FA$. The set A is called a *carrier* and the mapping α is called a *structure* of the coalgebra $\langle A, \alpha \rangle$.

A *homomorphism* from an F-coalgebra $\langle A, \alpha \rangle$ to a F-coalgebra $\langle B, \beta \rangle$ is a mapping $f\colon A \to B$ such that $T(f) \circ \alpha = \beta \circ f$.

An F-coalgebra $\langle S, \sigma \rangle$ is said to be a *subcoalgebra* of an F-coalgebra $\langle A, \alpha \rangle$ whenever there is an injective homomorphism from $\langle S, \sigma \rangle$ into $\langle A, \alpha \rangle$. This fact is denoted by $\langle S, \sigma \rangle \leq \langle A, \alpha \rangle$.

We denote the disjoint union of a family $\{X_j\}_{j \in J}$ of sets by $\Sigma_{j \in J} X_j$. Let $\{\langle A_i, \alpha_i \rangle\}_{i \in I}$ be a family of F-coalgebras. The *disjoint sum* $\Sigma_{i \in I} \langle A_i, \alpha_i \rangle$ of the family $\{\langle A_i, \alpha_i \rangle\}_{i \in I}$ of F-coalgebras is an F-coalgebra defined as follows. The carrier set of the disjoint sum $\Sigma_{i \in I} \langle A_i, \alpha_i \rangle$ is the disjoint union of the carriers of $\langle A_i, \alpha_i \rangle$, i.e. $A := \Sigma_{i \in I} A_i$. The structure α of the disjoint sum $\Sigma_{i \in I} \langle A_i, \alpha_i \rangle$ is defined as

$$\alpha : A \to FA; A_i \ni a \mapsto F(e_i) \circ \alpha_i(a),$$

where $e_i : A_i \to A; a \mapsto (a, i)$ for any $i \in I$.

A functor $F\colon \mathsf{Set} \to \mathsf{Set}$ *preserves pullbacks* if for any mappings $f\colon A \to B$ and $g : C \to B$ and their pullback $P(f, g) = \{(a, c) \in A \times C \mid f(a) = g(c)\}$ with π_1 and π_2 the following diagram is a pullback diagram.

$$\begin{tikzcd}
FA \arrow[r, "Ff"] & FB \\
F[P(f,g)] \arrow[u, "F\pi_1"] \arrow[r, "F\pi_2"] & FC \arrow[u, "Fg"']
\end{tikzcd}$$

We say that F *weakly preserves pullbacks* if the diagram above is a weak pullback. A functor F *(weakly) preserves kernel pairs* if it (weakly) preserves pullbacks $P(f, f), \pi_1, \pi_2$ for any mapping $f : A \to B$. For a detailed analysis of pullback and kernel pair preservations the reader is referred to [2].

Let Pos be the category of all posets and monotonic mappings. Note that there is a forgetful functor $U : \mathsf{Pos} \to \mathsf{Set}$ assigning to each poset $(X, \leq)$ the underlying set X and to each monotonic map $f : (X, \leq) \to (Y, \leq)$ the map $f : X \to Y$.

From now on we assume that a functor F we work with is $F : \mathsf{Set} \to \mathsf{Pos}$. We may naturally assign to F its composition $\bar{F} = U \circ F$ with the forgetful functor $U : \mathsf{Pos} \to \mathsf{Set}$. For the sake of simplicity of notation most of the times we will identify the functor $F : \mathsf{Set} \to \mathsf{Pos}$ with the Set-endofunctor $\bar{F} = U \circ F$ and write F to denote both F and $\bar{F}$. Considering set-based coalgebras over functors whose codomain category is a concrete category different from Set is not a new approach. A similar one has been adopted by e.g. J. Hughes and B. Jacobs in [3] when definining simulations for coalgebras.

Example 1. The powerset endofunctor $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ can be considered a functor $\mathcal{P} : \mathsf{Set} \to \mathsf{Pos}$ which assigns to any set X the poset $(\mathcal{P}(X), \subseteq)$ and to any map $f : X \to Y$ the order preserving map $\mathcal{P}(f)$.

Example 2. For any functor $H : \mathsf{Set} \to \mathsf{Set}$ the composition $\mathcal{P}H$ may be regarded as a functor $\mathcal{P}H : \mathsf{Set} \to \mathsf{Pos}$ with a natural ordering given by inclusion. In this paper we will focus our attention on coalgebras over the following functors:

- $\mathcal{P}(\Sigma \times \mathcal{I}d)$,
- $\mathcal{P}(\Sigma + \mathcal{I}d)$,
- $\mathcal{P}(\Sigma \times \mathcal{D})$,

where $\mathcal{D}$ is the distribution functor, i.e. a functor which assigns to any set X the set $\mathcal{D}X := \{\mu : X \to [0,1] \mid \sum_{x \in X} \mu(x) = 1\}$ of discrete measures and to any mapping $f : X \to Y$ a mapping $\mathcal{D}f : \mathcal{D}X \to \mathcal{D}Y$, which assigns to any measure $\mu \in \mathcal{D}X$ the measure $\mathcal{D}f(\mu) : Y \to [0,1]$ such that

$$\mathcal{D}f(\mu)(y) = \sum_{f(x)=y} \mu(x) \text{ for any } y \in Y.$$

The coalgebras over the first functor are exactly labelled transition systems. The coalgebras for $\mathcal{P}(\Sigma + \mathcal{I}d)$ expand a class of coalgebras studied by J. Rutten in [10]. Finally, the $\mathcal{P}(\Sigma \times \mathcal{D})$-coalgebras generalize the class of simple Segala systems introduced and thoroughly studied in [7],[8].

For a functor $F : \mathsf{Set} \to \mathsf{Pos}$ and for any sets X, Y we introduce an order on the set $Hom(X, FY)$ as follows. For $f, g \in Hom(X, FY)$ put

$$f \leq g \overset{def}{\iff} f(x) \leq_{FY} g(x) \text{ for any } x \in X.$$

Given $f : X \to Y$, $\alpha : X \to FZ$, $g : Z \to U$ and $\beta : Y \to FU$ an inequality $Fg \circ \alpha \leq \beta \circ f$ will be denoted by a diagram on the left and an equality $Fg \circ \alpha = \beta \circ f$ will be denoted by a diagram on the right:

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "\leq"] & Y \arrow[d, "\beta"] \\ FZ \arrow[r, "Fg"'] & FU \end{tikzcd} \qquad \begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "="] & Y \arrow[d, "\beta"] \\ FZ \arrow[r, "Fg"'] & FU \end{tikzcd}$$

Lemma 1. *Let $\alpha, \beta \in Hom(X, FY)$ and let $f : Z \to X$ be an epimorphism in* Set*. If $\alpha \circ f \leq \beta \circ f$ in $Hom(Z, FY)$ then $\alpha \leq \beta$ in $Hom(X, FY)$.*

Proof. Since $\alpha \circ f \leq \beta \circ f$ then for any $z \in Z$ we have $\alpha(f(z)) \leq_{FY} \beta(f(z))$. Because $f(Z) = X$ we have $\alpha(x) \leq_{FY} \beta(x)$ for any $x \in X$. Hence, $\alpha \leq \beta$ in $Hom(X, FY)$. □

3 Coalgebraic Operators and Saturators

Definition 3. *Let $U : \mathsf{Set}_F \to \mathsf{Set}$ be the forgetful functor and let C be full subcategory of the category of F-coalgebras and homomorphisms between them which is closed under taking inverse images of homomorphisms, i.e. if $\langle B, \beta\rangle \in \mathsf{C}$ and there is a homomorphism $f : \langle A, \alpha\rangle \to \langle B, \beta\rangle$ for $\langle A, \alpha\rangle \in \mathsf{Set}_F$ then $\langle A, \alpha\rangle \in \mathsf{C}$. A* coalgebraic operator *$\mathfrak{o}$ with respect to a class C is a functor $\mathfrak{o} : \mathsf{C} \to \mathsf{Set}_F$ such that the following diagram commutes:*

$$\begin{tikzcd}
\mathsf{C} \arrow[r, "\mathfrak{o}"] \arrow[dr, "U"'] & \mathsf{Set}_F \arrow[d, "U"] \\
& \mathsf{Set}
\end{tikzcd}$$

In other words, if $f : A \to B$ is a homomorphism between two F-coalgebras $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ belonging to C then f is a homomorphism between $\langle A, \mathfrak{o}\alpha\rangle$ and $\langle B, \mathfrak{o}\beta\rangle$, i.e.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "="] & B \arrow[d, "\beta"] & \Longrightarrow & A \arrow[r, "f"] \arrow[d, "\mathfrak{o}\alpha"'] \arrow[dr, phantom, "="] & B \arrow[d, "\mathfrak{o}\beta"] \\
FB \arrow[r, "Ff"'] & FB & & FA \arrow[r, "Ff"'] & FB
\end{tikzcd}$$

We say that a coalgebraic operator $\mathfrak{s}$ with respect to a class C is a saturator *if for any two F-coalgebras $\langle A, \alpha\rangle$, $\langle B, \beta\rangle$ from C and any mapping $f : A \to B$ the inequality $Ff \circ \alpha \leq \mathfrak{s}\beta \circ f$ is equivalent to $Ff \circ \mathfrak{s}\alpha \leq \mathfrak{s}\beta \circ f$. We may express the property in diagrams as follows:*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\mathfrak{s}\beta"] & \Longleftrightarrow & A \arrow[r, "f"] \arrow[d, "\mathfrak{s}\alpha"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\mathfrak{s}\beta"] \\
FA \arrow[r, "Ff"'] & FB & & FA \arrow[r, "Ff"'] & FB
\end{tikzcd}$$

Lemma 2. *Let $\mathfrak{s} : \mathsf{C} \to \mathsf{Set}_F$ be an operator w.r.t. a full subcategory C of Set_F and additionally let $\mathfrak{s}(\mathsf{C}) \subseteq \mathsf{C}$. Then $\mathfrak{s}$ is a saturator if and only if it satisfies the following three properties:*

- *$\alpha \leq \mathfrak{s}\alpha$ for any coalgebra $\langle A, \alpha\rangle \in \mathsf{C}$ (extensivity),*
- *$\mathfrak{s} \circ \mathfrak{s} = \mathfrak{s}$ (idempotency),*
- *if $Ff \circ \alpha \leq \beta \circ f$ then $Ff \circ \mathfrak{s}\alpha \leq \mathfrak{s}\beta \circ f$ for any $f : X \to Y$ (monotonicity):*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\beta"] \\ FA \arrow[r, "Ff"'] & FB \end{tikzcd} \implies \begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\mathfrak{s}\alpha"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\mathfrak{s}\beta"] \\ FA \arrow[r, "Ff"'] & FB \end{tikzcd}$$

The intuition behind the notion of a saturator is the following. Given a coalgebraic structure $\alpha : A \to FA$ it contains some information about observable and unobservable single step transitions. The process of saturating a structure intuitively boils down to adding additional information to α about multiple compositions of unobservable steps and a single composition of observable transitions (see examples below). Since for any set A the set FA is intuitively considered as the set of all possible outcomes of a computation, the partial order $\leq$ on FA compares those outcomes. In particular, the property of extensivity of a saturator means the saturated structure $\mathfrak{s}\alpha$ contains at least the same information about single-step transitions as α (in the sense of the partial order $\leq$). Idempotency means that the process of adding new information to α by saturating it ends after one iteration. Finally, monotonicity is self-explanatory.

Example 3. Let Σ be a non-empty set. Any LTS $\langle A, \Sigma, \to\rangle$ may be represented as a $\mathcal{P}(\Sigma \times \mathcal{I}d)$-coalgebra $\langle A, \alpha\rangle$ as follows. We define $\alpha : A \to \mathcal{P}(\Sigma \times A)$ by:

$$(\sigma, a') \in \alpha(a) \iff a \xrightarrow{\sigma} a'.$$

Let $\tau \in \Sigma$ be a silent transition label. For a coalgebra structure $\alpha : A \to \mathcal{P}(\Sigma \times A)$ we define its saturation $\mathfrak{s}\alpha : A \to \mathcal{P}(\Sigma \times A)$ as follows. For any element $a \in A$ put

$$\mathfrak{s}\alpha(a) := \alpha(a) \cup \{(\tau, a') \mid a \xrightarrow{\tau^*} a'\} \cup \{(\sigma, a') \mid a \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} \circ \xrightarrow{\tau^*} a' \text{ for } \sigma \neq \tau\}.$$

Verifying that $\mathfrak{s} : \mathsf{Set}_{\mathcal{P}(\Sigma\times\mathcal{I}d)} \to \mathsf{Set}_{\mathcal{P}(\Sigma\times\mathcal{I}d)}; \langle A, \alpha\rangle \mapsto \langle A, \mathfrak{s}\alpha\rangle$ is a coalgebraic saturator with respect to the class of all $\mathcal{P}(\Sigma \times \mathcal{I}d)$-coalgebras is left to the reader.

Example 4. Consider the functor $F = \mathcal{P}(\Sigma + \mathcal{I}d)$. Let $\alpha : A \to \mathcal{P}(\Sigma + A)$ be a structure of an F-coalgebra $\langle A, \alpha\rangle$. For the sake of simplicity of notation for any $a \in A$ let $\eta(a) := \alpha(a) \cap A$ and $\theta(a) := \alpha(a) \cap \Sigma$. Put $\eta^*(a) := \{a\} \cup \bigcup_{n\in\mathbb{N}} \eta^n(a)$, where $\eta^n(a) := \eta(\eta^{n-1}(a))$ for $n > 1$ and $\theta^*(a) := \theta(\eta^*(a))$. Define the saturation $\mathfrak{s}\alpha : A \to FA$ as follows:

$$\mathfrak{s}\alpha(a) := \eta^*(a) \cup \theta^*(a) \text{ for any } a \in A.$$

The assignment $\mathfrak{s}$ is a coalgebraic saturator with respect to the class of all F-coalgebras.

Example 5. For the functor $F = \mathcal{P}(\Sigma \times \mathcal{D})$, an F-coalgebra $\langle A, \alpha\rangle$, a state $a \in A$ and $\sigma \in \Sigma$ we write $a \xrightarrow{\sigma} \mu$ if $(\sigma, \mu) \in \alpha(a)$. For a state $a \in A$ and a measure $\nu \in \mathcal{D}(\Sigma \times A)$ a pair (a, ν) is called a *step* in $\langle A, \alpha\rangle$ only if there is $\sigma \in A$ and $\mu \in \mathcal{D}A$ such that $a \xrightarrow{\sigma} \mu$ and $\nu(\sigma, a') = \mu(a')$ for any $a' \in A$. A *combined* *step* in $\langle A, \alpha\rangle$ is a pair (a, ν), where $a \in A$ and $\nu \in \mathcal{D}(\Sigma \times A)$ for which there is a countable family of non-negative numbers $\{p_i\}_{i\in I}$ such that $\sum_{i\in I} p_i = 1$ and a countable family of steps $\{(a, \nu_i)\}_{i\in I}$ in $\langle A, \alpha\rangle$ such that $\nu = \sum_{i\in I} p_i \cdot \nu_i$. The definition of a combined step is a slight modification of a similar definition presented in [7]. The notion of weak arrows $\overset{\sigma}{\Longrightarrow}_P$ remains the same regardless of the small difference between the two definitions. Let $\tau \in \Sigma$ be the invisible transition. As in [7] for any $\sigma \in \Sigma$ we write $a \overset{\sigma}{\Longrightarrow}_P \mu$ whenever $\sigma = \tau$ and $\mu \in \mathcal{D}A$ for which $\mu(a) = 1$ or there is a combined step (a, ν) in $\langle A, \alpha\rangle$ such that if $(\sigma', a') \notin \{\sigma, \tau\} \times A$ then $\nu(\sigma', a') = 0$ and $\mu = \sum_{(\sigma',a')\in\{\sigma,\tau\}\times A} \nu(\sigma', a') \cdot \mu_{(\sigma',a')}$ and if $\sigma' = \sigma$ then $a' \overset{\tau}{\Longrightarrow}_P \mu_{(\sigma',a')}$ otherwise $\sigma' = \tau$ and $a' \overset{\sigma}{\Longrightarrow}_P \mu_{(\sigma',a')}$. Now, define $\mathfrak{s}\alpha : A \to FA$ by putting $\mathfrak{s}\alpha(a) := \{(\sigma, \mu) \mid a \overset{\sigma}{\Longrightarrow}_P \mu\}$ for any $a \in A$. A proof that $\mathfrak{s}$ is a coalgebraic saturator is left to the reader.

4 Two Approaches to Defining Weak Bisimulation

In this section we assume that $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ are members of the class C.

Definition 4. *A relation $R \subseteq A \times B$ is called a* weak bisimulation *provided that there is a structure $\gamma_1 : R \to FR$ and a structure $\gamma_2 : R \to FR$ for which:*

- $\alpha \circ \pi_1 = F\pi_1 \circ \gamma_1$ *and* $F\pi_2 \circ \gamma_1 \leq \mathfrak{s}\beta \circ \pi_2$,
- $\beta \circ \pi_2 = F\pi_2 \circ \gamma_2$ *and* $F\pi_1 \circ \gamma_2 \leq \mathfrak{s}\alpha \circ \pi_1$.

$$\begin{tikzcd}
A \arrow[d, "\alpha"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma_1"] \arrow[l, phantom, "=", shift right=6] \arrow[r, phantom, "\leq", shift right=6] & B \arrow[d, "\mathfrak{s}\beta"] \\
FA & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FB
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, "\mathfrak{s}\alpha"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma_2"] \arrow[l, phantom, "\geq", shift right=6] \arrow[r, phantom, "=", shift right=6] & B \arrow[d, "\beta"] \\
FA & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FB
\end{tikzcd}$$

Example 6. Consider the LTS functor $F = \mathcal{P}(\Sigma \times \mathcal{I}d)$ and the saturator $\mathfrak{s}$ introduced in Example 3. Let $\langle A, \alpha\rangle$ be an F-coalgebra. Consider a relation $R \subseteq A \times A$ which satisfies the assumptions of Definition 4. This means that if $(a, b) \in R$ then there is $\gamma_1 : R \to \mathcal{P}(\Sigma \times R)$ such that $\alpha(a) = F(\pi_1)(\gamma_1(a, b))$ and $F(\pi_2)(\gamma_1(a, b)) \subseteq \mathfrak{s}\alpha(b)$. In other words, there is a subset $S \subseteq \Sigma \times R$ such that $\gamma_1(a, b) = S$ and $F(\pi_1)(S) = \alpha(a)$ and $F(\pi_2)(S) \subseteq \mathfrak{s}\alpha(b)$. This means that for any $(\sigma, a') \in \alpha(a)$ there is $b' \in A$ such that $(\sigma, b') \in \mathfrak{s}\alpha(b)$ and $(a', b') \in R$. Hence, if $\sigma = \tau$ then $a \overset{\tau}{\to} a'$ implies $b \overset{\tau^*}{\to} b'$ and $(a', b') \in R$ otherwise $a \overset{\sigma}{\to} a'$ implies $b \overset{\tau^*\sigma\tau^*}{\to} b'$ and $(a', b') \in R$. The second condition from Definition 4 gives us the following assertion. If $(a, b) \in R$ and $b \overset{\tau}{\to} b'$ then there is $a' \in A$ such that $a \overset{\tau^*}{\to} a'$ and $(a', b') \in R$. Moreover, if for $\sigma \neq \tau$ we have $b \overset{\sigma}{\to} b'$ then there is $a' \in A$ such that $a \overset{\tau^*\sigma\tau^*}{\to} a'$ and $(a', b') \in R$. We see that this is exactly the condition presented in Definition 2.

Example 7. Consider the functor $F = \mathcal{P}(\Sigma + \mathcal{I}d)$ and the saturator $\mathfrak{s}$ from Example 4. Since the functor $\Sigma + \mathcal{I}d$ is a subfunctor of F take an F-coalgebra

$\langle A, \alpha \rangle$ which is a $\Sigma + \mathcal{I}d$-coalgebra, i.e. the structure α is a mapping $\alpha : A \to \Sigma + A$. Now take a relation $R \in A \times A$ which satisfies the assumptions presented in Definition 4 and let $(a, b) \in R$. This means that there is a structure $\gamma_1 : R \to \mathcal{P}(\Sigma + R)$ such that $F(\pi_1)(\gamma_1(a, b)) = \alpha(a)$ and $F(\pi_2)(\gamma_1(a, b)) \subseteq \mathfrak{s}\alpha(a)$. In other words there is a pair X, S of subsets, where $X \subseteq R$ and $S \subseteq \Sigma$, and $F(\pi_1)(X \cup S) = (\pi_1(X) \cup S) = \alpha(a)$ and $F(\pi_2)(X \cup S) = (\pi_2(X) \cup S) \subseteq \mathfrak{s}\alpha(a)$. If $\alpha(a) = a' \in A$ then this means that $\pi_1(X) = \{a'\}$ and $S = \varnothing$. Hence, there is $b' \in A$ such that $(a', b') \subseteq X \subseteq R$ and $b' \in \eta^*(b) = \{b'' \mid b \to^* b''\}$. If $\alpha(a) = \sigma \in \Sigma$ then $\pi_1(X) = \varnothing$, and hence $X = \varnothing$, $S = \{\sigma\}$. Therefore, $\sigma \in \theta^*(b)$. It follows that there is $b' \in B$ such that $b \to^* b'$ and $b' \downarrow \sigma$. By the second condition of Definition 4 we infer that if $b \to b'$ then there is $a' \in A$ such that $a \to^* a'$ and $(a', b') \in R$. Otherwise if $b \downarrow \sigma \in \Sigma$ then there is $a' \in A$ such that $a \to^* a'$ and $a' \downarrow \sigma$. This definition coincides with a definition of weak bisimulation between $\Sigma + \mathcal{I}d$-coalgebras presented in [10].

Example 8. Let $F = \mathcal{P}(\Sigma \times \mathcal{D})$ and consider the saturator $\mathfrak{s} : \mathsf{Set}_F \to \mathsf{Set}_F$ defined in Example 5. It is easy to see that for two simple Segala systems $\langle A, \alpha \rangle$, $\langle B, \beta \rangle$ a relation $R \subseteq A \times B$ is a weak bisimulation provided that the following condition holds. If $(a, b) \in R$ and $a \xrightarrow{\sigma} \mu$ then $b \overset{\sigma}{\Longrightarrow}_P \mu'$ and $(\mu, \mu') \in (F\pi_1, F\pi_2)(\mathcal{D}R)$. Moreover, if $(a, b) \in R$ and $b \xrightarrow{\sigma} \mu'$ then $a \overset{\sigma}{\Longrightarrow}_P \mu$ and $(\mu, \mu') \in (F\pi_1, F\pi_2)(\mathcal{D}R)$. This definition coincides with the one presented in [7],[8].

Proposition 1. *Let $R \subseteq A \times B$ be a standard bisimulation between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$. Then R is also a weak bisimulation between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$.*

Theorem 1. *If a relation $R \subseteq A \times B$ is a weak bisimulation between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$ then $R^{-1} = \{(b, a) \mid (a, b) \in R\}$ is a weak bisimulation between $\langle B, \beta \rangle$ and $\langle A, \alpha \rangle$.*

Theorem 2. *If all members of a family $\{R_i\}_{i \in I}$ of relations $R_i \subseteq A \times B$ are weak bisimulations between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$ then $\bigcup_{i \in I} R_i$ is also a weak bisimulation between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$.*

Proof. Let $\{R_i\}_{i \in I}$ together with $\gamma_1^i : R_i \to FR_i$ and $\gamma_2^i : R_i \to FR_i$ be a family of weak bisimulations between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$.

$$\begin{tikzcd}
A \arrow[d, "\alpha"'] & R_i \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma_1^i"'] \arrow[l, phantom, "=", shift right=6ex] \arrow[r, phantom, "\leq", shift right=6ex] & B \arrow[d, "\mathfrak{s}\beta"] \\
FA & FR_i \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FB
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, "\mathfrak{s}\alpha"'] & R_i \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma_2^i"'] \arrow[l, phantom, "\geq", shift right=6ex] \arrow[r, phantom, "=", shift right=6ex] & B \arrow[d, "\beta"] \\
FA & FR_i \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FB
\end{tikzcd}$$

First consider the disjoint sum $\sum_{i \in I} \langle R_i, \gamma_1^i \rangle = \langle \sum_{i \in I} R_i, \gamma_1 \rangle$. We will prove that given $\tau : \sum_{i \in I} R_i \to A \times B; (r, i) \to r$ the mappings $p_1 = \pi_1 \circ \tau$ and $p_2 = \pi_2 \circ \tau$ satisfy:

$$\alpha \circ p_1 = F(p_1) \circ \gamma_1,$$
$$\mathfrak{s}\beta \circ p_2 \geq F(p_2) \circ \gamma_1.$$

Note that for any $i \in I$ we have

$$\alpha \circ p_1 \circ e_i = \alpha \circ \pi_1 = F(\pi_1) \circ \gamma_1^i = F(p_1) \circ F(e_i) \circ \gamma_1^i = F(p_1) \circ \gamma_1 \circ e_i.$$

Hence, $\alpha \circ p_1 = F(p_1) \circ \gamma_1$. Moreover, for any $i \in I$ we have

$$\mathfrak{s}\beta \circ p_2 \circ e_i = \mathfrak{s}\beta \circ \pi_2 \geq F(\pi_2) \circ \gamma_1^i = F(p_2) \circ F(e_i) \circ \gamma_1^i = F(p_2) \circ \gamma_1 \circ e_i.$$

By Lemma 1 it follows that $\mathfrak{s}\beta \circ p_2 \geq F(p_2) \circ \gamma_1$.

$$\begin{tikzcd}
\Sigma_{i\in I} R_i \arrow[d, "\gamma_1"'] \arrow[rr, bend left, "p_1"] \arrow[dr, phantom, "="] & R_i \arrow[l, "e_i"] \arrow[r, "\pi_1"'] \arrow[d, "\gamma_1^i"'] \arrow[dr, phantom, "="] & A \arrow[d, "\alpha"] \\
F\Sigma_{i\in I} R_i \arrow[rr, bend right, "Fp_1"'] & FR_i \arrow[l, "Fe_i"'] \arrow[r, "F\pi_1"] & FA
\end{tikzcd}
\qquad
\begin{tikzcd}
\Sigma_{i\in I} R_i \arrow[d, "\gamma_1"'] \arrow[rr, bend left, "p_2"] \arrow[dr, phantom, "="] & R_i \arrow[l, "e_i"] \arrow[r, "\pi_2"'] \arrow[d, "\gamma_1^i"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\mathfrak{s}\beta"] \\
F\Sigma_{i\in I} R_i \arrow[rr, bend right, "Fp_2"'] & FR_i \arrow[l, "Fe_i"'] \arrow[r, "F\pi_2"] & FB
\end{tikzcd}$$

Note that the image of $\sum_{i\in I} R_i$ under the map $\tau : \sum_{i\in I} R_i \to A \times B$ is equal to $\bigcup_{i\in I} R_i \subseteq A \times B$. Put $\gamma_1' : \bigcup_{i\in I} R_i \to F(\bigcup_{i\in I} R_i)$ so that, for any $r \in \bigcup_{i\in I} R_i$ there is $(r,i) \in \sum_{i\in I} R_i$ such that $\gamma_1'(r) = F(\tau)(\gamma_1(r,i))$. Observe that

$$\begin{aligned}
&\alpha(\pi_1(r)) = \alpha(\pi_1(\tau(r,i))) = \alpha(p_1(r,i)) = F(p_1)(\gamma_1(r,i)) = \\
&= F(\pi_1)(F(\tau)(\gamma_1(r,i))) = F(\pi_1) \circ \gamma_1'(r), \\
&\mathfrak{s}\beta(\pi_2(r)) = \mathfrak{s}\beta(\pi_2(\tau(r,i))) = \mathfrak{s}\beta \circ p_2(r,i) \geq F(p_2) \circ \gamma_1(r,i) = \\
&= F(\pi_2) \circ F(\tau) \circ \gamma_1(r,i) = F(\pi_2) \circ \gamma_1'(r).
\end{aligned}$$

Hence, $\alpha \circ \pi_1 = F(\pi_1) \circ \gamma_1'$ and $\mathfrak{s}\beta \circ \pi_2 \geq F(\pi_2) \circ \gamma_1'$. Similarily we prove existence of $\gamma_2' : \bigcup_{i\in I} R_i \to F(\bigcup_{i\in I} R_i)$ possessing the desired properties and making $\bigcup_{i\in I} R_i$ a weak bisimulation. □

The following lemma is an analogue of a similar result for standard bisimulations presented in e.g. [11] (Lemma 5.3). Moreover, the proof of Lemma 3 is a direct translation of the proof of the analogous result. Hence, we leave the following result without a proof.

Lemma 3. *Let X be a set and let $\xi_1 : X \to FX$ and $\xi_2 : X \to FX$ be two coalgebraic structures. Finally, let $f : X \to A$, $g : X \to B$ be mappings such that f is a homomorphism from $\langle X, \xi_1 \rangle$ to $\langle A, \alpha \rangle$, g is a homomorphism from $\langle X, \xi_2 \rangle$ to $\langle B, \beta \rangle$ and the mappings f, g satisfy:*

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, "\xi_1"'] \arrow[dr, phantom, "\leq"] & B \arrow[d, "\mathfrak{s}\beta"] \\
FX \arrow[r, "Fg"'] & FB
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\xi_2"'] \arrow[dr, phantom, "\leq"] & A \arrow[d, "\mathfrak{s}\alpha"] \\
FX \arrow[r, "Ff"'] & FA
\end{tikzcd}$$

then the set $\langle f, g \rangle (X) = \{(f(x), g(x)) \in A \times B \mid x \in X\}$ is a weak bisimulation between $\langle A, \alpha \rangle$ and $\langle B, \beta \rangle$.

Theorem 3. *Let $F : \mathsf{Set} \to \mathsf{Set}$ weakly preserve pullbacks and let $\langle A, \alpha\rangle$, $\langle B, \beta\rangle$ and $\langle C, \delta\rangle$ be F-coalgebras from the class C. Let R_1 be a weak bisimulation between $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ and R_2 be a weak bisimulation between $\langle B, \beta\rangle$ and $\langle C, \delta\rangle$. Then*

$$R_1 \circ R_2 = \{(a, c) \mid \exists b \in B \text{ s.t. } (a, b) \in R_1 \text{ and } (b, c) \in R_2\}$$

is a weak bisimulation between $\langle A, \alpha\rangle$ and $\langle C, \delta\rangle$.

Corollary 1. *If $F : \mathsf{Set} \to \mathsf{Set}$ weakly preserves pullbacks then the greatest weak bisimulation on a coalgebra $\langle A, \alpha\rangle$ is an equivalence relation.*

Definition 5. *A relation $R \subseteq A \times B$ is said to be a* saturated weak bisimulation *between $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ provided that there is a structure $\gamma : R \to FR$ for which the following diagram commutes:*

$$\begin{tikzcd}
A \arrow[d, "\mathfrak{s}\alpha"'] \arrow[dr, phantom, "="] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"'] \arrow[dr, phantom, "="] & B \arrow[d, "\mathfrak{s}\beta"] \\
FA & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FB
\end{tikzcd}$$

Remark 1. We see that a saturated weak bisimulation between $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ is defined as a standard bisimulation between saturated models $\langle A, \mathfrak{s}\alpha\rangle$ and $\langle B, \mathfrak{s}\beta\rangle$. Hence, any property true for standard bisimulation is also true for a saturated weak bisimulation.

Proposition 2. *Let $R \subseteq A \times B$ be a standard bisimulation between $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$. Then R is also a saturated weak bisimulation between $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$.*

Theorem 4. *Let $F : \mathsf{Set} \to \mathsf{Set}$ weakly preserve kernel pairs and let $R \subseteq A \times A$ be an equivalence relation which is a weak bisimulation on $\langle A, \alpha\rangle$. Then R is a saturated weak bisimulation on $\langle A, \alpha\rangle$.*

Proof. Let $\gamma_1 : R \to FR$ be a structure for which $\alpha \circ \pi_1 = F\pi_1 \circ \gamma_1$ and $F\pi_2 \circ \gamma_1 \leq \mathfrak{s}\alpha \circ \pi_2$. By properties of the saturator $\mathfrak{s}$ it follows that $\mathfrak{s}\alpha \circ \pi_1 = F\pi_1 \circ \mathfrak{s}\gamma_1$ and $F\pi_2 \circ \mathfrak{s}\gamma_1 \leq \mathfrak{s}\alpha \circ \pi_2$. In other words,

$$\begin{tikzcd}
A \arrow[d, "\mathfrak{s}\alpha"'] \arrow[dr, phantom, "="] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\mathfrak{s}\gamma_1"'] \arrow[dr, phantom, "\leq"] & A \arrow[d, "\mathfrak{s}\alpha"] \\
FA & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FA
\end{tikzcd}$$

Let $p : A \to A/R; a \mapsto a/R$. Since $F : \mathsf{Set} \to \mathsf{Set}$ preserves kernel pairs the following diagram is a weak pullback diagram:

$$\begin{tikzcd}
& FR \arrow[dl, "F\pi_1"'] \arrow[dr, "F\pi_2"] & \\
FA \arrow[dr, "Fp"'] & & FA \arrow[dl, "Fp"] \\
& F(A/R) &
\end{tikzcd}$$

Since $Fp \circ F\pi_1 = Fp \circ F\pi_2$ we have $Fp \circ \mathfrak{s}\alpha \circ \pi_1 = Fp \circ F\pi_1 \circ \mathfrak{s}\gamma_1 = Fp \circ F\pi_2 \circ \mathfrak{s}\gamma_1 \leq Fp \circ \mathfrak{s}\alpha \circ \pi_2$. Let $k : R \to R; (a, b) \mapsto (b, a)$. We see that $Fp \circ \mathfrak{s}\alpha \circ \pi_1 \circ k \leq Fp \circ \mathfrak{s}\alpha \circ \pi_2 \circ k$. Since $\pi_1 \circ k = \pi_2$ and $\pi_2 \circ k = \pi_1$ it follows that

$$Fp \circ \mathfrak{s}\alpha \circ \pi_2 \leq Fp \circ \mathfrak{s}\alpha \circ \pi_1.$$

Hence, $Fp \circ \mathfrak{s}\alpha \circ \pi_1 = Fp \circ \mathfrak{s}\alpha \circ \pi_2$. In other words, the set R together with $\mathfrak{s}\alpha \circ \pi_1$ and $\mathfrak{s}\alpha \circ \pi_2$ is a cone over the diagram $FA \overset{Fp}{\to} F(A/R) \overset{Fp}{\leftarrow} FA$. Recall that FR with $F\pi_1$ and $F\pi_2$ is a weak pullback of $FA \overset{Fp}{\to} F(A/R) \overset{Fp}{\leftarrow} FA$. The fact that F preserves weak pullbacks provides us with a mediating morphism $\gamma : R \to FR$ satisfying

$$\mathfrak{s}\alpha \circ \pi_1 = F\pi_1 \circ \gamma \text{ and } \mathfrak{s}\alpha \circ \pi_2 = F\pi_2 \circ \gamma.$$

□

We say that two elements $a, b \in A$ are weakly bisimilar, and write $a \approx_w b$ if there is a weak bisimulation $R \subseteq A \times A$ on $\langle A, \alpha \rangle$ for which $(a, b) \in R$. We say that a and b are saturated weakly bisimilar, and write $a \approx_{sw} b$, if there is a saturated weak bisimulation R on $\langle A, \alpha \rangle$ containing (a, b).

Corollary 2. *Let $F : \mathsf{Set} \to \mathsf{Set}$ be a functor weakly preserving pullbacks. Then the relations $\approx_w$ and $\approx_{sw}$ are equivalence relations and*

$$\approx_w \subseteq \approx_{sw} .$$

Definition 6. *We say that a functor $F : \mathsf{Set} \to \mathsf{Pos}$* preserves downsets *provided that for any $f : X \to Y$ and any $\boldsymbol{x} \in FX$ the following equality holds:*

$$Ff(\boldsymbol{x}\downarrow) = Ff(\{\boldsymbol{x}' \in FX \mid \boldsymbol{x}' \leq \boldsymbol{x}\}) = Ff(\boldsymbol{x})\downarrow = \{\boldsymbol{y} \in FY \mid \boldsymbol{y} \leq Ff(\boldsymbol{x})\}.$$

It is easy to see that all functors from Example 2 preserve downsets. In the following example we will present a functor weakly preserving pullbacks and not preserving downsets for which the greatest weak bisimulation and the greatest saturated weak bisimulation do not always coincide.

Example 9. Define a functor $F : \mathsf{Set} \to \mathsf{Set}$ by $F = \mathcal{I}d^2 + \mathcal{I}d$. Clearly, the functor F weakly preserves pullbacks. For any set X let us introduce a partial order $\leq$ on FX as the smallest partial order satisfying

$$x \leq (x, x) \text{ for any } x \in X.$$

The order $\leq$ is well defined and turns the functor F into $F : \mathsf{Set} \to \mathsf{Pos}$. Now consider sets $X = \{x_1, x_2\}$, $Y = \{y\}$ and the unique mapping $f : X \to Y$. Take $(y, y) \in FY$ and note that $y \leq (y, y) = (f(x_1), f(x_2)) = Ff((x_1, x_2))$. In other words, $y \in Ff(x_1, x_2)\downarrow$ and $(x_1, x_2)\downarrow = \{(x_1, x_2)\}$. Hence, the functor F does not preserve downsets. For any F-coalgebra $\langle A, \alpha \rangle$ define an operator $\mathfrak{s}\alpha : A \to FA$ by

$$\mathfrak{s}\alpha(a) := \textbf{ if } \alpha(a) = b \textbf{ then } (b, b) \textbf{ else } \alpha(a).$$

The operator $\mathfrak{s} : \mathsf{Set}_F \to \mathsf{Set}_F$ is a coalgebraic saturator with respect to the class of all F-coalgebras. Now consider a set $A = \{x, y\}$ and define a structure $\alpha : A \to FA$ by $\alpha(x) = x$ and $\alpha(y) = (x, y)$. Clearly, $x \approx_{sw} y$ since $\mathfrak{s}\alpha(x) = (x, x)$ and $\mathfrak{s}\alpha(y) = (x, y)$ and if we put $R = \{(x, y), (x, x)\}$ then for $\gamma : R \to FR$ defined by $\gamma(x, y) = ((x, x), (x, y))$, $\gamma(x, x) = ((x, x), (x, x))$ we have $\mathfrak{s}\alpha \circ \pi_1 = F\pi_1 \circ \gamma$ and $\mathfrak{s}\alpha \circ \pi_2 = F\pi_2 \circ \gamma$. At the same time the states x and y are not weakly bisimilar. Indeed, if there was a weak bisimulation R containing (x, y) then it would imply existence of $\gamma_1 : R \to FR$ satisfying $\alpha \circ \pi_1 = F\pi_1 \circ \gamma_1$ and $F\pi_2 \circ \gamma_1 \leq \mathfrak{s}\alpha \circ \pi_2$. Since $(x, y) \downarrow = \{(x, y)\}$ we would then have

$$x = \alpha(x) = \alpha(\pi_1(x, y)) = F\pi_1(\gamma(x, y)),$$
$$(x, y) = \mathfrak{s}\alpha(y) = \mathfrak{s}\alpha(\pi_2(x, y)) = F\pi_2(\gamma(x, y))$$

which is impossible.

Theorem 5. *Let $F : \mathsf{Set} \to \mathsf{Set}$ weakly preserve kernel pairs and preserve downsets. Let $R \subseteq A \times A$ be an equivalence relation which is a saturated weak bisimulation on $\langle A, \alpha \rangle$. Then R is a weak bisimulation on $\langle A, \alpha \rangle$.*

Proof. Let $\gamma : R \to FR$ be the structure for which $\mathfrak{s}\alpha \circ \pi_1 = F\pi_1 \circ \gamma$ and $F\pi_2 \circ \gamma = \mathfrak{s}\alpha \circ \pi_2$. Let $p : A \to A/R; a \mapsto a/R$. Since $F : \mathsf{Set} \to \mathsf{Set}$ preserves kernel pairs the following diagram is a weak pullback diagram:

$$\begin{tikzcd}
 & FR \arrow[dl, "F\pi_1"'] \arrow[dr, "F\pi_2"] & \\
FA \arrow[dr, "Fp"'] & & FA \arrow[dl, "Fp"] \\
 & F(A/R) &
\end{tikzcd}$$

Consider the mappings $Fp \circ \alpha \circ \pi_1$ and $Fp \circ \mathfrak{s}\alpha \circ \pi_2$ and observe that

$$Fp \circ \alpha \circ \pi_1 \leq Fp \circ \mathfrak{s}\alpha \circ \pi_1 =$$
$$Fp \circ F\pi_1 \circ \gamma = Fp \circ F\pi_2 \circ \gamma = Fp \circ \mathfrak{s}\alpha \circ \pi_2.$$

This means that for any pair $(a, b) \in R$ we have $Fp(\alpha(a)) \leq Fp(\mathfrak{s}\alpha(b))$. In other words,

$$Fp(\alpha(a)) \in Fp(\mathfrak{s}\alpha(b)) \downarrow$$

Since F preserves downsets, there exists an element $\boldsymbol{x} \in FA$ such that $\boldsymbol{x} \leq \mathfrak{s}\alpha(b)$ for which

$$Fp(\alpha(a)) = Fp(\boldsymbol{x}).$$

Because FR together with $F\pi_1$ and $F\pi_2$ is a weak pullback of the diagram $FA \overset{Fp}{\to} F(A/R) \overset{Fp}{\leftarrow} FA$, there is an element $\boldsymbol{r}_{(a,b)} \in FR$ such that $F\pi_1(\boldsymbol{r}_{(a,b)}) = \alpha(a)$ and $F\pi_2(\boldsymbol{r}_{(a,b)}) = \boldsymbol{x}$. Define $\gamma_1 : R \to FR; (a, b) \mapsto \boldsymbol{r}_{(a,b)}$. The structure γ_1 satisfies $\alpha \circ \pi_1 = F\pi_1 \circ \gamma_1$ and $F\pi_2 \circ \gamma_1 \leq \mathfrak{s}\alpha \circ \pi_2$. Similarily, we prove existence of $\gamma_2 : R \to FR$ satisfying $\mathfrak{s}\alpha \circ \pi_1 \geq F\pi_1 \circ \gamma_2$ and $F\pi_2 \circ \gamma_2 = \alpha \circ \pi_2$. □

Corollary 3. *Let $S : \mathsf{Set} \to \mathsf{Set}$ weakly preserve pullbacks and preserve downsets. Then for any S-coalgebra $\langle A, \alpha \rangle$ the relations $\approx_w$ and $\approx_{sw}$ are equivalence relations and*

$$\approx_w = \approx_{sw} .$$

5 Weak Coinduction Principle

In this section we assume that the saturator $\mathfrak{s}$ we work with is an operator defined on the whole category of F-coalgebras. In the category Set_F for any mapping $f : A \to B$ we have $f : \langle A, \alpha\rangle \to \langle B, \beta\rangle$ is a homomorphism if and only if the relation $gr(f) = \{(a, f(a)) \in A \times B \mid a \in A\}$ is a standard bisimulation (see [2],[11]). This motivates considering the following category (F. Bonchi, personal communication). Let Set_F^w denote the category in which objects are standard F-coalgebras and in which a map $f : A \to B$ is a morphism between two objects $\langle A, \alpha\rangle$ and $\langle B, \beta\rangle$ provided that the relation $gr(f) = \{(a, f(a)) \mid a \in A\}$ is a weak bisimulation.

Lemma 4. *Let $\langle A, \alpha\rangle$ be an F-coalgebra and let a family $\langle S_i, \sigma_i\rangle$, where $S_i \subseteq A$, be a family of subcoalgebras of $\langle A, \alpha\rangle$ in Set_F such that $\mathfrak{s}\sigma_i = \sigma_i$. Then there is a structure $\sigma : \bigcup S_i \to F(\bigcup S_i)$ making $\langle \bigcup S_i, \sigma\rangle$ a subcoalgebra of $\langle A, \alpha\rangle$ and $\mathfrak{s}\sigma = \sigma$.*

Corollary 4. *Let $\langle A, \alpha\rangle$ be an F-coalgebra. There is the greatest subcoalgebra $\langle S, \sigma\rangle$ of $\langle A, \alpha\rangle$ such that $\mathfrak{s}\sigma = \sigma$.*

Lemma 5. *Let $\langle T, t\rangle$ be a terminal coalgebra in the category Set_F. Then $\langle T, \mathfrak{s}t\rangle$ is a weakly terminal object in Set_F^w.*

Let $\langle T_s, t'\rangle$ be the greatest subcoalgebra of the terminal coalgebra $\langle T, t\rangle$ in Set_F such that $\mathfrak{s}t' = t'$.

Lemma 6. *If F admits a terminal coalgebra $\langle T, t\rangle$ in Set_F and weakly preserves pullbacks then the greatest weak bisimulation on $\langle T_s, t'\rangle$ is the equality relation.*

Proof. Assume that for $x, y \in T_s$ we have $x \approx_w y$. This implies $x \approx_{sw} y$. Since, $\mathfrak{s}t' = t'$ this means that the saturated weak bisimulation $\approx_{sw}$ is a standard bisimulation on $\langle T_s, t'\rangle$. Since, the coalgebra $\langle T_s, t'\rangle$ is a subcoalgebra of the terminal coalgebra $\langle T, t\rangle$ the bisimulation $\approx_{sw}$ is an equality relation (see [2,11] for details). Hence $x = y$. □

The above lemma allows us to formulate a weak coinduction principle for $\langle T_s, t'\rangle$. For two states $x, y \in T_s$ we have $x \approx_w y \iff x = y$.

Theorem 6. *If F admits a terminal coalgebra $\langle T, t\rangle$ in Set_F and weakly preserves pullbacks then the coalgebra $\langle T_s, t'\rangle$ is a terminal object in Set_F^w.*

Proof. To prove that $\langle T_s, t'\rangle$ is weakly terminal it is enough to construct a homomorphism in Set_F^w from $\langle T, \mathfrak{s}t\rangle$ to $\langle T_s, t'\rangle$ and apply Lemma 5. Since $\langle T, t\rangle$ is terminal in Set_F there exists a unique homomorphism $[\![-]\!]_{\mathfrak{s}t} : \langle T, \mathfrak{s}t\rangle \to \langle T, t\rangle$. Any homomorphic image is a subcoalgebra of the codomain (see [2], [11] for details). Therefore, we can consider $[\![-]\!]_{\mathfrak{s}t}$ as an onto homomorphism between $\langle T, \mathfrak{s}t\rangle$ and $\left\langle [\![T]\!]_{\mathfrak{s}t}, t_{|[\![T]\!]_{\mathfrak{s}t}}\right\rangle$, where the object $\left\langle [\![T]\!]_{\mathfrak{s}t}, t_{|[\![T]\!]_{\mathfrak{s}t}}\right\rangle$ is a subcoalgebra of $\langle T, t\rangle$ in Set_F. In other words, we have

$$\begin{tikzcd} T \arrow[r, "\llbracket - \rrbracket_{\mathfrak{s}t}"] \arrow[d, "\mathfrak{s}t"'] \arrow[dr, phantom, "="] & \llbracket T \rrbracket_{\mathfrak{s}t} \arrow[d, "t_{|\llbracket T \rrbracket_{\mathfrak{s}t}}"] \\ FT \arrow[r, "F\llbracket - \rrbracket_{\mathfrak{s}t}"'] & F\llbracket T \rrbracket_{\mathfrak{s}t} \end{tikzcd}$$

Hence,

$$\begin{tikzcd} T \arrow[r, "\llbracket - \rrbracket_{\mathfrak{s}t}"] \arrow[d, "\mathfrak{s}t"'] \arrow[dr, phantom, "="] & \llbracket T \rrbracket_{\mathfrak{s}t} \arrow[d, "\mathfrak{s}t_{|\llbracket T \rrbracket_{\mathfrak{s}t}}"] \\ FT \arrow[r, "F\llbracket - \rrbracket_{\mathfrak{s}t}"'] & F\llbracket T \rrbracket_{\mathfrak{s}t} \end{tikzcd}$$

Therefore, $\mathfrak{s}t_{|\llbracket T \rrbracket_{\mathfrak{s}t}} = t_{|\llbracket T \rrbracket_{\mathfrak{s}t}}$ and $\left\langle \llbracket T \rrbracket_{\mathfrak{s}t}, t_{|\llbracket T \rrbracket_{\mathfrak{s}t}} \right\rangle$ is a subcoalgebra of $\langle T_s, t' \rangle$. For uniqueness consider two homomorphisms f_1, f_2 from an F-coalgebra $\langle A, \alpha \rangle$ to $\langle T_s, t' \rangle$ in Set_F^w. This means that the relations $gr(f_1)$ and $gr(f_2)$ are weak bisimulations between $\langle A, \alpha \rangle$ and $\langle T_s, t' \rangle$. By the properties of weak bisimulations the relation $gr(f_1)^{-1} \circ gr(f_2) = \{(f_1(a), f_2(a)) \mid a \in A\}$ is a weak bisimulation on $\langle T_s, t' \rangle$. By Lemma 6 we get that $f_1(a) = f_2(a)$ for any $a \in A$. □

For an F-coalgebra $\langle A, \alpha \rangle$ let $\llbracket - \rrbracket_\alpha^w$ denote the unique homomorphism from $\langle A, \alpha \rangle$ to $\langle T_s, t' \rangle$ in Set_F^w. We see in the proof of Theorem 6 that $\llbracket - \rrbracket_\alpha^w = \llbracket - \rrbracket_{\mathfrak{s}t} \circ \llbracket - \rrbracket_\alpha$.

Theorem 7. *Let F weakly preserve pullbacks. For two elements $a, b \in A$ we have*

$$a \approx_w b \iff \llbracket a \rrbracket_\alpha^w = \llbracket b \rrbracket_\alpha^w$$

Proof. Assume $a \approx_w b$. Since $\llbracket - \rrbracket_\alpha^w$ is a homomorphism in Set_F^w the relation $gr(\llbracket - \rrbracket_\alpha^w)$ is a weak bisimulation between $\langle A, \alpha \rangle$ and $\langle T_s, t' \rangle$. Since F weakly preserves pullbacks the relation $gr(\llbracket - \rrbracket_\alpha^w)^{-1} \circ \approx_w \circ gr(\llbracket - \rrbracket_\alpha^w)$ is a weak bisimulation on $\langle T_s, t' \rangle$ such that

$$(\llbracket a \rrbracket_\alpha^w, \llbracket b \rrbracket_\alpha^w) \in gr(\llbracket - \rrbracket_\alpha^w)^{-1} \circ \approx_w \circ gr(\llbracket - \rrbracket_\alpha^w).$$

By Lemma 6 we get that $\llbracket a \rrbracket_\alpha^w = \llbracket b \rrbracket_\alpha^w$. Conversely, let $\llbracket a \rrbracket_\alpha^w = \llbracket b \rrbracket_\alpha^w$. This means that the weak bisimulation $gr(\llbracket - \rrbracket_\alpha^w) \circ gr(\llbracket - \rrbracket_\alpha^w)^{-1}$ on $\langle A, \alpha \rangle$ contains a pair (a, b). Hence, $a \approx_w b$. □

6 Summary and Future Work

In this paper we introduced a coalgebraic setting in which we can define weak bisimulation in two ways generalizing Definition 1 and Definition 2 and compared them. We showed that the definitions coincide with the standard definitions of weak bisimulation for labelled transition systems and simple Segala systems. The approach towards defining weak bisimulation presented in this paper has two main advantages. First of all, it is a very general and simple approach. In particular it does not require an explicit specification of the observable and unobservable part of the functor. Second of all, it easily captures the computational aspects of weak bisimilarity. It is worth noting that it has some limitations. Part of the author's ongoing research is to establish the reason why it does not work for e.g. fully probabilistic processes introduced in [1] and studied from the perspective of coalgebra in [9]. Moreover, it may seem that the setting presented in the paper is too general. To justify the statement note that for instance for an LTS coalgebra $\langle A, \alpha \rangle$ we may define a saturator as follows:

$$\mathfrak{s}\alpha(a) := \{(\tau, a)\} \cup \{(\sigma, a') \mid a \xrightarrow{\tau^*} \circ \xrightarrow{\sigma} a'\}.$$

The above definition of a saturator would lead to a different definition of weak bisimulation for LTS. Therefore, it is necessary to establish more concrete ways for defining standard saturators of coalgebras that lead to standard definitions of weak bisimulations.

Acknowledgements. I would like to thank Jan Rutten for many interesting comments and hospitality in CWI, Amsterdam, where part of the work on this paper was completed. I am very grateful to F. Bonchi, M. Bonsangue, H. Hansen, B. Klin and A. Silva for various suggestions for improvement of my results and for future work. I would also like to express my gratitude to anonymous referees of the paper for valuable comments and remarks.

References

1. Baier, C., Hermanns, H.: Weak Bisimulation for Fully Probabilistic Processes. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 119–130. Springer, Heidelberg (1997)
2. Gumm, H.P.: Elements of the general theory of coalgebras. In: LUATCS 1999, Rand Afrikaans University, Johannesburg (1999)
3. Hughes, J., Jacobs, B.: Simulations in coalgebra. TCS 327(1-2), 71–108 (2004)
4. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
5. Rothe, J.: A syntactical approach to weak (bi)-simulation for coalgebras. In: Moss, L. (ed.) Proc. CMCS 2002. ENTCS, vol. 65, pp. 270–285 (2002)
6. Rothe, J., Masulović, D.: Towards weak bisimulation for coalgebras. In: Proc. Categorical Methods for Concurrency, Interaction and Mobility. ENTCS, vol. 68(1), pp. 32–46 (2002)
7. Segala, R., Lynch, N.: Probabilistic Simulations for Probabilistic Processes. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 481–496. Springer, Heidelberg (1994)
8. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis. MIT (1995)
9. Sokolova, A., de Vink, E., Woracek, H.: Coalgebraic Weak Bisimulation for Action-Type Systems. Sci. Ann. Comp. Sci. 19, 93–144 (2009)
10. Rutten, J.J.M.M.: A note on coinduction and weak bisimilarity for while programs. Theoretical Informatics and Applications (RAIRO) 33, 393–400 (1999)
11. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. TCS 249(1), 3–80 (2000)

Appendix

Proof (Lemma 2). Let $\mathfrak{s}$ be a saturator and let $\langle A,\alpha\rangle \in \mathsf{C}$. We see that for the identity mapping $id_A : A \to A$ we have

$$F(id_A)\circ \mathfrak{s}\alpha = \mathfrak{s}\alpha \leq \mathfrak{s}\alpha = \mathfrak{s}\alpha \circ id_A.$$

This implies that $F(id_A)\circ\alpha \leq \mathfrak{s}\alpha\circ id_A$. Hence, $\alpha \leq \mathfrak{s}\alpha$. We see that by extensivity rule and the assumption $\mathfrak{s}(\mathsf{C}) \subseteq \mathsf{C}$ we get $\mathfrak{s}\alpha \leq \mathfrak{s}(\mathfrak{s}\alpha)$. Therefore, to prove idempotency it is enough to show that $\mathfrak{s}(\mathfrak{s}\alpha) \leq \mathfrak{s}\alpha$. To do this consider the following inequality $F(id_A)\circ\mathfrak{s}\alpha = \mathfrak{s}\alpha \leq \mathfrak{s}\alpha = \mathfrak{s}\alpha\circ id_A$. By order preservation it implies that $F(id_A)\circ\mathfrak{s}(\mathfrak{s}\alpha) \leq \mathfrak{s}\alpha\circ id_A$. Thus, $\mathfrak{s}(\mathfrak{s}\alpha) \leq \mathfrak{s}\alpha$. Finally, consider two coalgebras $\langle A,\alpha\rangle$ and $\langle B,\beta\rangle$ from C and a mapping $f : A \to B$ such that $Ff\circ\alpha \leq \beta\circ f$. Note that $Ff\circ\alpha \leq \beta\circ f \leq \mathfrak{s}\beta\circ f$. Since $Ff\circ\alpha \leq \mathfrak{s}\beta\circ f$ then by the fact that $\mathfrak{s}$ is a saturator it follows that $Ff\circ\mathfrak{s}\alpha \leq \mathfrak{s}\beta\circ f$.

Now in order to prove the converse consider a coalgebraic operator $\mathfrak{s} : \mathsf{C} \to \mathsf{C}$ which is a closure operator (i.e. is extensive, idempotent and monotonic). Let $\langle A,\alpha\rangle$ and $\langle B,\beta\rangle$ be coalgebras from C and let $f : A \to B$ be a mapping. Assume that $Ff\circ\alpha \leq \mathfrak{s}\beta\circ f$. By idempotency and monotonicity we get $Ff\circ\mathfrak{s}\alpha \leq \mathfrak{s}\beta\circ f$. By extensivity we conclude that $Ff\circ\alpha \leq \mathfrak{s}\beta\circ f$. □

Proof (Lemma 4). Let $\sigma : \bigcup S_i \to F(\bigcup S_i)$ be the unique structure making $\langle\bigcup S_i,\sigma\rangle$ a subcoalgebra of $\langle A,\alpha\rangle$ (such a structure always exists [2],[10]). Let $e_i : S_i \to \bigcup S_i$ denote the inclusions. Then $\sigma\circ e_i = F(e_i)\circ\sigma_i$. Since $\mathfrak{s}$ is a saturator this means that $\mathfrak{s}\sigma\circ e_i = F(e_i)\circ\mathfrak{s}\sigma_i = F(e_i)\circ\sigma_i$. Therefore, for any i we have $\mathfrak{s}\sigma\circ e_i = \sigma\circ e_i$. Hence, $\mathfrak{s}\sigma = \sigma$. □

Proof (Lemma 5). Let $\langle A,\alpha\rangle$ be any F-coalgebra. Let $[\![-]\!]_\alpha : \langle A,\alpha\rangle \to \langle T,t\rangle$ be the unique homomorphism in Set_F with $\langle T,t\rangle$ as codomain. Then $[\![-]\!]_\alpha$ is also a homomorphism between $\langle A,\mathfrak{s}\alpha\rangle$ and $\langle T,\mathfrak{s}t\rangle$. Since $[\![-]\!]_\alpha$ is a standard homomorphim the relation $gr([\![-]\!]_\alpha) = \{(a,[\![a]\!]_\alpha) \mid a\in A\}$ is a standard bisimulation between $\langle A,\alpha\rangle$ and $\langle T,t\rangle$. I.e. there is $\gamma : gr([\![-]\!]_\alpha) \to F(gr([\![-]\!]_\alpha))$ such that

$$\begin{tikzcd}
A \arrow[d, "\alpha"'] & gr([\![-]\!]_\alpha) \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & T \arrow[d, "t"] \\
FA & Fgr([\![-]\!]_\alpha) \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FT
\arrow[from=1-1, to=2-2, phantom, "="] \arrow[from=1-2, to=2-3, phantom, "="]
\end{tikzcd}$$

Since $t \leq \mathfrak{s}t$ and $\mathfrak{s}\mathfrak{s}t = \mathfrak{s}t$ this implies that

$$\begin{tikzcd}
A \arrow[d, "\alpha"'] & gr([\![-]\!]_\alpha) \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & T \arrow[d, "\mathfrak{s}\mathfrak{s}t=\mathfrak{s}t"] \\
FA & Fgr([\![-]\!]_\alpha) \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FT
\arrow[from=1-1, to=2-2, phantom, "="] \arrow[from=1-2, to=2-3, phantom, "\leq"]
\end{tikzcd}$$

Moreover, by saturating the same diagram we get

$$\begin{tikzcd}
A \arrow[d, "\mathfrak{s}\alpha"'] & gr(\llbracket - \rrbracket_\alpha) \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\mathfrak{s}\gamma"] \arrow[l, phantom, "=", shift right=20] \arrow[r, phantom, "=", shift right=20] & T \arrow[d, "\mathfrak{s}t"] \\
FA & Fgr(\llbracket - \rrbracket_\alpha) \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FT
\end{tikzcd}$$

This means that $gr(\llbracket - \rrbracket_\alpha)$ together with γ and $\mathfrak{s}\gamma$ is a weak bisimulation between $\langle A, \alpha\rangle$ and $\langle T, \mathfrak{s}t\rangle$ which concludes the proof. □

A Context-Free Linear Ordering with an Undecidable First-Order Theory*

Arnaud Carayol[1] and Zoltán Ésik[2]

[1] Laboratoire d'Informatique Gaspard-Monge, Université Paris-Est, France
[2] Institute of Informatics, University of Szeged, Hungary

Abstract. The words of a context-free language, ordered by the lexicographic ordering, form a context-free linear ordering. It is well-known that the linear orderings associated with deterministic context-free languages have a decidable monadic second-order theory. In stark contrast, we give an example of a context-free language whose lexicographic ordering has an undecidable first-order theory.

1 Introduction

When the alphabet of a language L is linearly ordered, we may equip L with the lexicographic ordering. It is known that every countable linear ordering is isomorphic to the lexicographic ordering of a (prefix) language.

The lexicographic orderings of regular languages (i.e., the regular linear orderings) were studied in [1–4, 12, 15, 20, 24, 28]. These linear orderings agree with the leaf orderings of the regular trees, and are all automatic linear orderings as defined in [22]. It follows from results in [20] that all scattered regular linear orderings have finite Hausdorff rank, or finite condensation rank (FC-rank), as defined in [27]. In fact all automatic linear orderings have finite FC-rank [22]. Moreover, an ordinal is the order type of a regular well-ordering if and only if it is strictly less than ω^ω.

The study of the lexicographic orderings of context-free languages (context-free linear orderings) was initiated in [4] and further developed in [5, 6, 8, 16–18] and was extended to languages generated by *deterministic* higher order grammars in [7].

It follows from early results in [13] that the lexicographic orderings of deterministic context-free languages are (up to isomorphism) identical to the leaf orderings of the algebraic trees, cf. [5]. In [4], it was shown that every ordinal

* Arnaud Carayol has been supported by the project AMIS (ANR 2010 JCJC 0203 01 AMIS). Both authors received partial support from the project TÁMOP-4.2.1/B-09/1/KONV-2010-0005 "Creating the Center of Excellence at the University of Szeged", supported by the European Union and co-financed by the European Regional Fund. Zoltán Ésik was also partly supported by the National Foundation of Hungary for Scientific Research, grant no. K 75249, and by a chair Labex Bézout as part of the program "Investissements d'Avenir" (ANR-10-LABX-58).

less than ω^{ω^ω} is the order type of a deterministic context-free linear ordering and it was conjectured that a well-ordering is isomorphic to a context-free linear ordering if and only if its order type is less than ω^{ω^ω}. This conjecture was confirmed in [5] for deterministic context-free linear orderings, and in [18] for context-free linear orderings. Moreover, it was shown in [6] and [18] that the FC-rank of every scattered deterministic context-free linear ordering and in fact every scattered context-free linear ordering is less than ω^ω. Since the FC-rank of a well-ordering is less than ω^ω exactly when its order type is less than ω^{ω^ω}, it follows in conjunction with results proved in [4] that a well-ordering is isomorphic to the lexicographic ordering of a context-free language or deterministic context-free language if and only if its order type is less than ω^{ω^ω}. Exactly the same ordinals are the order types of the tree automatic well-orderings, see [14]. Eventually, it was proved in [8] that the FC-rank of every context-free linear ordering is less than ω^ω. However, the question whether there exists a context-free linear ordering that is not a deterministic context-free linear ordering remained open.

Since deterministic context-free linear orderings belong to the pushdown hierarchy [9–11, 25], they all have decidable monadic second-order theories. In fact, there exists an algorithm that takes two inputs, an $LR(1)$ grammar (or equivalently a deterministic pushdown automaton) and a sentence of the monadic second-order logic of linear orders and tells whether the sentence holds in the lexicographic ordering of the language generated by the grammar. Such a decision procedure does not exist for all context-free grammars and monadic second-order or even first-order sentences, since as shown in [16], it is undecidable to tell whether a context-free linear ordering (given by a context-free grammar) is dense. In contrast, it is decidable whether a context-free linear ordering is a well-ordering or a scattered ordering.

In this paper we prove that there is a context-free linear ordering whose first-order theory is undecidable. Thus there exists a context-free linear ordering which is not the lexicographic ordering of a deterministic context-free language. The context-free language defining this linear ordering is a finite disjoint union of deterministic context-free languages. Hence our undecidability result holds for the class of unambiguous context-free linear orderings.

As a corollary, we also obtain the existence of a (unambiguous) context-free language whose associated tree has an undecidable monadic second-order theory. The tree of a language is composed of the set of all prefixes of the words of the language as set of vertices, and its ancestor relation is simply the prefix relation. This result in turn proves the existence a context-free language that cannot be accepted by any deterministic collapsible pushdown automaton (an extension of the classical notion of pushdown automaton with nested stacks and links [19]), as shown previously by Paweł Parys using a pumping argument [26].

The paper is organised as follows. In Section 2, we recall basic definitions on linear orderings. Definitions concerning first-order logic and the structures associated with languages are given in Section 3. Section 1 presents our main result and its corollaries are given in Section 5. Section 6 concludes the paper.

2 Linear Orderings

A piece of notation: for a nonnegative integer n, we will denote the set $\{1, \dots, n\}$ by $[n]$.

When A is an alphabet, we let A^* denote the set of all finite words over A, including the empty word ϵ. The set A^+ is $A^* - \{\epsilon\}$. We let u^R denote the mirror image of a word $u \in A^*$.

A *linear ordering* [27] $(I, <)$ is a set I equipped with a strict linear order relation $<$. As usual, we will write $x \leq y$ for $x, y \in I$ if $x < y$ or $x = y$. A linear ordering $(I, <)$ is finite or countable if I is. A *morphism* of linear orderings is an order preserving map. Note that every morphism is necessarily injective. When $(I, <)$ and $(J, <')$ are linear orderings such that $I \subseteq J$ and the embedding $I \hookrightarrow J$ is a morphism, we call $(I, <)$ a *subordering* of $(J, <')$. In this case the relation $<$ is the restriction of the relation $<'$ onto I and we usually write just I for $(I, <)$.

An *isomorphism* is a bijective morphism. Isomorphic linear orderings are said to have the same *order type*. The order types of the positive integers $\mathbb{N}$, negative integers $\mathbb{N}_-$, all integers $\mathbb{Z}$, and the rationals $\mathbb{Q}$, ordered as usual, are denoted ω, ω^*, ζ and η, respectively. As usual, the finite order types may be identified with the nonnegative integers.

Recall that a linear ordering $(I, <)$ is *dense* if it has at least two elements and for every $x, y \in I$ with $x < y$ there is some $z \in I$ with $x < z < y$. A *quasi-dense* linear ordering is a linear ordering that has a dense subordering, and a *scattered* linear ordering is a linear ordering that is not quasi-dense. For example, $\mathbb{N}$ and $\mathbb{Z}$ are scattered, $\mathbb{Q}$ is dense, and the ordering obtained by replacing each or some point in $\mathbb{Q}$ with a 2-element linear ordering is quasi-dense but not dense. Clearly, every subordering of a scattered linear ordering is scattered. It is well-known that a linear ordering is quasi-dense if and only if it has a subordering of order type η. Moreover, up to isomorphism, there are 4 countable dense linear orderings, the ordering $\mathbb{Q}$ of the rationals possibly equipped with a least or greatest element, or both.

When $(I, <)$ is a linear ordering and for each $i \in I$, $(J_i, <_i)$ is a linear ordering, the *ordered sum*

$$\sum_{i \in I} (J_i, <_i)$$

is the disjoint union $\bigcup_{i \in I} (J_i \times \{i\})$ equipped with the order relation $(x, i) < (y, j)$ if and only if either $i < j$, or $i = j$ and $x <_i y$. When each $(J_i, <_i)$ is the linear ordering $(J, <')$, we call the ordered sum the *product* of $(I, <)$ and $(J, <')$, denoted $(I, <\ ) \times (J, <')$. Finite ordered sums are also denoted as $(I_1, <_1\ ) + \cdots + (I_n, <_n)$. Since the operation of ordered sum preserves isomorphism, we may also define ordered sums of order types. For example, $1 + \eta + 1$ is the order type of the rationals equipped with both a least and a greatest element. It is known that every scattered sum of scattered linear orderings is scattered. This means that if $(I, <)$ is scattered as is each $(J_i, <_i)$, then $\sum_{i \in I} (J_i, <_i)$ is also scattered. A sum over a dense linear ordering $(I, <)$ is referred to as a *dense sum*.

3 First-Order Logic

A *signature* is a ranked set σ of symbols. We let $|R|$ denote the arity (≥ 1) of the symbol R. A *relational structure* S over σ is given by a tuple $(D, (R^S)_{R\in\sigma})$, where D is the domain of S, and where for all $R \in \sigma$, the interpretation of R in S denoted R^S is a subset of $D^{|R|}$. When S is clear from the context, we just write R for its interpretation R^S.

Let $S = (D, (R^S)_{R\in\sigma})$ and $S' = (D', (R^{S'})_{R\in\sigma})$ be two structures over σ. An *isomorphism* h from D to D' is a bijection from D to D' such that for all $R \in \sigma$ and for all $u_1, \dots, u_{|R|} \in D$, $(u_1, \dots, u_{|R|}) \in R^S$ if and only if $(h(u_1), \dots, h(u_{|R|})) \in R^{S'}$. We let $S \cong S'$ denote the existence of an isomorphism between S and S'.

A linear ordering $(I, <_I)$ is naturally represented as a structure over the signature σ_{ord} with one symbol $<$ of arity 2. Its domain is the set I and the symbol $<$ is interpreted as $<_I$.

First-order formulas use first-order variables, which are interpreted by elements of the structure and are denoted by lower case letters $x, y \dots$. Atomic first-order formulas are of the form $R(x_1, \dots, x_{|R|})$, where R is a relation symbol from the signature and $x_1, \dots, x_{|R|}$ are first-order variables, or $x = y$ for first-order variables x, y with the obvious semantics. Complex formulas are built as usual from atomic ones by the use of Boolean connectives and quantifiers. Free and bound occurrences of variables in a formula are defined as usual. We write $\varphi(x_1, \dots, x_n)$ to denote that the formula φ has free variables in $\{x_1, \dots, x_n\}$. A *closed formula* has no free variables.

For a formula $\varphi(x_1, \dots, x_n)$ and elements of the domain $u_1, \dots, u_n$, we write $S \models \varphi[u_1, \dots, u_n]$ to denote that the structure S *satisfies* the formula φ when the free variable x_i, $i \in [n]$, is interpreted as u_i. For a closed formula φ, we simply write $S \models \varphi$.

For example, the following formula over the signature σ_{ord} expresses that the structure is a linear ordering:

$$\begin{aligned} &\forall x\, \forall y \;\; x < y \rightarrow \neg(y < x) \\ \wedge\; &\forall x\, \forall y \;\; x < y \vee y < x \vee x = y \\ \wedge\; &\forall x\, \forall y\, \forall z \;\; (x < y \wedge y < z) \rightarrow x < z \end{aligned}$$

3.1 First-Order Interpretations

First-order interpretation is a transformation defining a structure in another structure using first-order logic.

Definition 1. *A first-order interpretation from a signature σ to a signature σ' is given by a tuple $(\delta, (\varphi_R)_{R\in\sigma'})$, where δ is a formula over σ with one free variable x_1, and for each symbol $R \in \sigma'$, φ_R is a formula over σ with free variables $x_1, \dots, x_{|R|}$.*

Applying a first-order interpretation $\mathcal{I}$ to a structure S over the signature σ gives rise to a structure over the signature σ', denoted $\mathcal{I}(S)$. Its domain is the set $D' = \{u \in D \mid S \models \delta[u]\}$. A symbol $R \in \sigma'$ is interpreted in $\mathcal{I}(S)$ as the set of all tuples satisfying φ_R :

$$\{(u_1, \ldots, u_{|R|}) \in (D')^{|R|} \mid S \models \varphi_R[u_1, \ldots, u_{|R|}]\}.$$

An example of first-order interpretation is given in Section 3.2.

3.2 Structures Associated with Words and Languages

Let A be a finite alphabet. A word w over A can be represented by a structure over the signature $\sigma_A = \{P_a \mid a \in A\} \cup \sigma_{\text{ord}}$ with $|P_a| = 1$ for all $a \in A$. This structure, denoted S_w, has the set $[|w|]$ of positions in the word as its domain. The symbol $<$ is interpreted (in S_w) as the natural order and for all $a \in A$, P_a is interpreted as the predicate marking all occurrences of the letter a.

For $A = \mathbf{2} = \{0, 1\}$, the formula φ over the signature σ_A given below expresses that a word starts with the letter 0 (i.e., for all $w \in A^*$, $S_w \models \varphi$ if and only if w stars with 0) :

$$\exists x \ \ (\forall y \neg (y < x) \wedge P_0(x)) \ .$$

Similarly, when $L \subseteq A^*$, we define the structure S_L over the signature σ_A associated with a language L. This structure is obtained by taking the disjoint union of all the structures S_w for $w \in L$. Note that as soon as L contains two nonempty words, the relation $<$ is no longer a linear ordering.

The following formula is satisfied by the languages in which all nonempty words start with the letter 0.

$$\forall x \ \ (\forall y \neg (y < x) \rightarrow P_0(x))$$

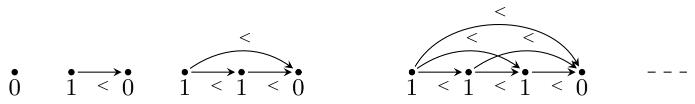


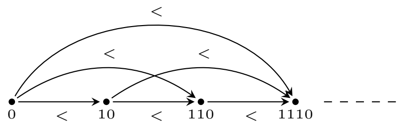
Fig. 1. The structure S_L associated with the language $L = 1^*0$

3.3 Lexicographic Ordering and Countable Words Associated with a Language

We will consider countable linear orderings that arise as lexicographic orderings of languages. Suppose that A is an alphabet which is linearly ordered by the relation $<$. Then we define a strict partial order $<_s$ on A^* by $u <_s v$ if and only if $u = xay$ and $v = xbz$ for some $x, y, z \in A^*$ and $a, b \in A$ with $a < b$. We also define $u <_p v$ if and only if u is a *proper* prefix of v, and $u <_\ell v$ if and only if $u <_s v$ or $u <_p v$. The *lexicographic order* relation $<_\ell$ turns A^* into a linear ordering. In particular, any language $L \subseteq A^*$ gives rise to a structure over the signature σ_{ord} denoted O_L and called the *lexicographic ordering of L*. The domain of O_L is the language L and the symbol $<$ is interpreted as the lexicographic ordering $<_\ell$.

We say that a language $L \subseteq A^*$ is scattered, dense, etc. if its lexicographic ordering has the corresponding property. Moreover, we say that a lexicographic ordering is a *regular* or a *context-free linear ordering* if it is isomorphic to the lexicographic ordering of a regular or context-free language. *Deterministic context-free linear orderings* are defined in the same way.

Example 1. Consider the alphabet $\mathbf{2} = \{0, 1\}$ ordered by $0 < 1$. The lexicographic ordering of the regular language 1^*0 is of order type ω and is depicted below.



Similarly the lexicographic orderings of the regular languages 0^*1, $0^+1 + 1^+0$ are of order type ω^* and ζ, respectively. The lexicographic ordering of $(00 + 11)^*01$ is η. The context-free linear ordering $(\bigcup_{n\geq 0} 1^n0(1^*0)^n, <_\ell)$ is of order type $1 + \omega + \omega^2 + \cdots = \omega^\omega$. The context-free linear orderings $(\bigcup_{n\geq 1} 1^n0(0(0^+1 + 1^+0) + 10^{<n}), <_\ell)$ and $(\bigcup_{n\geq 1} 1^n0(0(00 + 11)^*01 + 1(1^*0)^n), <_\ell)$ have respective order types $\zeta + 1 + \zeta + 2 + \cdots$ and $\eta + \omega + \eta + \omega^2 + \cdots$.

A *countable word* (called *arrangement* in [12]) over an alphabet B is a countable linear ordering whose elements are labelled by letters of B. Each language over an ordered alphabet A not containing the empty word gives rise to a *countable word* W_L over A, which is represented by a structure over the signature σ_A. Its domain is the language L. The symbol $<$ is interpreted as the lexicographic order $<_\ell$, and for all $a \in A$, P_a is interpreted as the set of words of L ending with the letter a. We say that a countable word is *context-free* if it is isomorphic to the countable word of some context-free language.

Lemma 1. *Every context-free linear ordering (resp. word) can be represented by a prefix context-free language not containing the empty word.*

Proof. We establish the result for context-free words. Let $A = \{a_1, \ldots, a_n\}$ with $a_1 < \cdots < a_n$, and let $L \subseteq A^+$ be a context-free language which does not contain the empty word.

Let $A' = \{a'_1, \ldots, a'_n\}$ be an alphabet disjoint from A and let $\pi : A^* \mapsto (A')^*$ be the morphism mapping a_i to a'_i for all $i \in [n]$.

Consider the context-free language L' over $A \cup A'$ ordered by $a_1 < \cdots < a_n < a'_1 < \cdots < a'_n$ defined by

$$L' = \{\pi(wa)a \mid wa \in L \text{ and } a \in A\}.$$

The language L' is prefix (as L' is included in $(A')^+A$ and A and A' are assumed to be disjoint). To conclude the proof, we observe that the mapping $\theta : L \mapsto L'$ mapping $wa \in L$ to $\pi(wa)a \in L'$ is an isomorphism from W_L to $W_{L'}$. □

Context-free words are clearly closed under substitution. Thus we have:

Lemma 2. *Let L be context-free language over an ordered alphabet A which does not contain the empty word, and suppose that for each $a \in A$, P_a is a context-free linear ordering. Then the ordered sum*

$$\sum_{u \in L} P_{\lambda(u)} \qquad \textit{where } \lambda(u) \textit{ designates the last letter of } u,$$

obtained by replacing each $u \in L$ ending with $a \in A$ by a copy of P_a, is a context-free linear ordering.

Proof. By Lemma 1, we can assume w.l.o.g. that L is prefix. For all $a \in A$, let L_a be a context-free language (which does not contain the empty word) defining the context-free linear ordering P_a. The ordered sum $\sum_{u \in L} P_{u(|u|)}$ is isomorphic to the context-free linear ordering defined by

$$\{waL_a \mid wa \in L \text{ and } a \in A\}.$$

□

This property in turn implies that context-free words can be defined in context-free linear orderings.

Lemma 3. *Let A be an ordered alphabet and let L be a context-free language not containing the empty word. There exists a context-free language L' and a first-order interpretation $\mathcal{I}$ such that W_L is isomorphic to $\mathcal{I}(O_{L'})$.*

Proof. Let L be a context-free language not containing the empty word over the alphabet $A = \{a_1, \ldots, a_n\}$. Consider the linear ordering O obtained by replacing in W_L each vertex labelled a_i by a copy of a linear ordering of order type $\zeta + i + \zeta$. As for all $i \geq 0$, $\zeta + i + \zeta$ is a context-free linear ordering, we obtain by Lemma 2 that O is a context-free linear ordering. Let L' be a context-free language such that $O_{L'}$ is isomorphic to O.

We now define a first-order interpretation transforming $O_{L'}$ into W_L. The first-order interpretation $\mathcal{I}$ only keeps vertices that have no predecessor. These vertices correspond to the first vertex in between two consecutive copies of ζ. Therefore these vertices are in a one to one correspondence with the elements of L. The order relation $<$ is inherited by $\mathcal{I}$. The predicate P_{a_1} is defined for those vertices with no successors, hence guaranteeing that the vertex (which must have no predecessor) lies in a copy of $\zeta + 1 + \zeta$. Similarly P_{a_2} is defined for vertices having a successor with no successor, etc. Formally the interpretation is defined by

$$\begin{aligned}
&\delta(x) = \forall y \, \neg \mathrm{Succ}(y, x) \\
&\varphi_{<}(x, y) = x < y \\
&\varphi_{P_{a_i}}(x_1) = \exists x_2 \ldots \exists x_i \quad \textstyle\bigwedge_{j \in [i-1]} \mathrm{Succ}(x_j, x_{j+1}) \wedge \forall y \, \neg \mathrm{Succ}(x_i, y)
\end{aligned}$$

where $\mathrm{Succ}(x, y)$ is the formula expressing that y is the successor of x. □

3.4 Tree of Language

The tree of a language L over A is a structure, denoted T_L, over the signature $\sigma_{\text{anc}} = \sigma_A \cup \{\prec\}$, where $\prec$ is of arity 2. The domain is the the set of prefixes of the language L. For all $a \in A$, P_a is interpreted as the set of words of L ending with the letter a, and $\prec$ is interpreted as the prefix relation $<_p$.

It is possible to define the tree of a language over a more restricted signature $\sigma_{\text{suc}} = \sigma_A \cup \{\text{Succ}\}$, where Succ is interpreted as the direct successor relation.

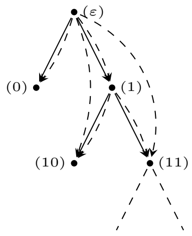


Fig. 2. The tree T_L of the language $L = 1^*0$ where full edges represent the Succ relation and dashed edges represent the relation $\prec$. The root is not labelled, all leaves are labelled by 0, and all other nodes by 1.

Lemma 4. *For any language L over an ordered alphabet A, the linear ordering O_L can be first-order interpreted in T_L over the signature σ_{anc}.*

4 Main Undecidability Result

This section is devoted to establishing the following theorem.

Theorem 1. *There exists a context-free word with an undecidable first-order theory. Furthermore, such a context-free word can be defined by a finite disjoint union of deterministic context-free languages.*

We now proceed with the proof of Theorem 1. The key ingredient of the proof are the languages obtained by a special form of product, denoted $\otimes$, of deterministic context-free languages.

Definition 2. *Let $L_1, \ldots, L_n$ be languages over the alphabet A. We let $L_1 \otimes \cdots \otimes L_n$ denote the language over the alphabet $A \times \mathbf{2}^n$ containing all nonempty words*

$$(a_1, \bar{b}_1) \cdots (a_m, \bar{b}_m)$$

such that for all $i \in [m]$, a_i belongs to A and $\bar{b}_i$ belongs to $\mathbf{2}^n$ and furthermore for all $\ell \in [n]$, the ℓ-th component of the "flag" $\bar{b}_i$ is equal to 1 if and only if the word $a_1 \cdots a_i$ belongs to L_ℓ.

Intuitively the ℓth bit of the ith letter of the attached flag signals if the prefix of length i projected on A belongs to the language L_ℓ.

Example 2. Let A be the alphabet $\{a, b, c, d\}$. Consider the two deterministic context-free languages $L_1 = \{w \in A^* \mid |w|_a = |w|_b\}$ and $L_2 = \{w \in A^* \mid |w|_c = |w|_d\}$. The language $L_1 \otimes L_2$ contains the word

$$\begin{pmatrix} a \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} c \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} b \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} d \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} a \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} d \\ 0 \\ 0 \end{pmatrix}$$

Note that the language $L_1 \otimes L_2$ is not a context-free language[1].

The key observation is that the structure associated with the product $L_1 \otimes L_2$ of two deterministic context-free languages L_1 and L_2 can be defined in first-order logic in some context-free word.

Proposition 1. *Let L_1 and L_2 be two deterministic context-free languages. There exists a language L over an ordered alphabet which is the disjoint union of deterministic context-free languages not containing the empty word such that the structure $S_{L_1 \otimes L_2}$ can be first-order interpreted in W_L.*

Proof. Let L_1 and L_2 be two deterministic context-free languages. Using a standard binary encoding, we can assume that L_1 and L_2 are on the binary alphabet $A = \{a, b\}$ ordered by $a < b$.

Consider the alphabet $B = \{\triangleright, \bar{a}, \bar{b}, 0_1, 1_1, 0_2, 1_2, \triangleleft, \#, a, b\}$ with $\triangleright < \bar{a} < \bar{b} < 0_1 < 1_1 < 0_2 < 1_2 < \triangleleft < \# < a < b$ and the language L which is the (disjoint) union of the following languages:

$$\begin{array}{ll}
\{u\#\triangleright & \mid u \in A^*\} \\
\{u\#u^R\triangleleft & \mid u \in A^*\} \\
\{u\#vx\bar{x} & \mid u \in A^*, v \in A^*, x \in A \text{ and } vx \leq_p u^R\} \\
\{u\#v0_i & \mid u \in A^*, v \in A^+ \text{ and } v \notin L_i\}, \quad i = 1, 2 \\
\{u\#v1_i & \mid u \in A^*, v \in A^+ \text{ and } v \in L_i\}, \quad i = 1, 2.
\end{array}$$

We now define a first-order interpretation transforming W_L into $S_{L_1 \otimes L_2}$. The interpretation only keeps the vertices labelled by the predicate $\bar{a}$ or $\bar{b}$:

$$\delta(x) = P_{\bar{a}}(x) \vee P_{\bar{b}}(x).$$

The order relation $<$ coincides with the order relation of W_L but restricted to vertices lying between a vertex labelled by $\triangleright$ and a vertex labelled by $\triangleleft$ with no vertex labelled by $\triangleleft$ in between:

$$\begin{aligned}
\varphi_<(x, y) = \exists z_1 \exists z_2 \ & (z_1 < x < y < z_2 \wedge P_\triangleright(z_1) \wedge P_\triangleleft(z_2) \\
& \wedge \forall z' \ (z_1 < z' < z_2 \to \neg P_\triangleleft(z'))).
\end{aligned}$$

[1] Indeed, by taking the intersection of $L_1 \otimes L_2$ with the regular language $(A \times \mathbf{2}^2)^*(A \times \{1\} \times \{1\})$ and then projecting on the first component, we can obtain the language $\{w \in A^* \mid w \neq \epsilon, \ |w|_a = |w_b| \text{ and } |w|_c = |w|_d\}$, which is known not to be context-free.

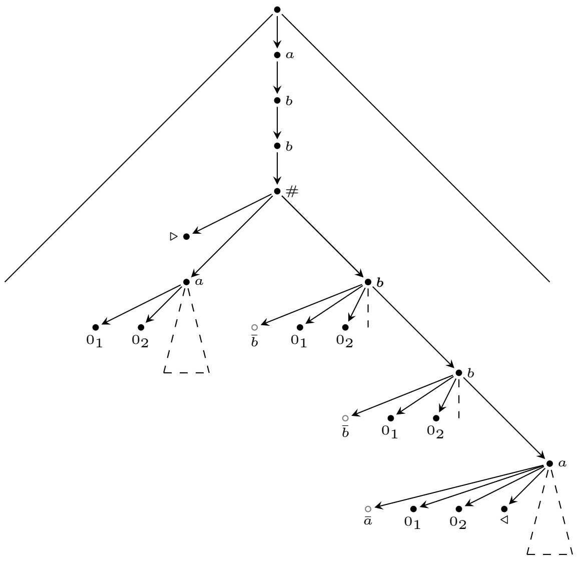


Fig. 3. Assuming that $L_1 = A^*b$ and $L_2 = A^*ba$, we depict the part of the tree T_L corresponding to the subset of the language producing after interpretation the word $(b, 1, 0)(b, 1, 0)(a, 0, 1)$. The white nodes correspond to words that are kept by the interpretation.

For $(a, b, c) \in A \times \mathbf{2} \times \mathbf{2}$, the predicate $P_{(a,b,c)}$ is defined by

$$\varphi_{P_{(a,b,c)}}(x) = P_{\bar{a}}(x) \wedge \exists y \exists z \ (\mathrm{Succ}(x, y) \ \wedge \ \mathrm{Succ}(y, z) \wedge P_{b_1}(y) \wedge P_{c_2}(z))$$

where b_1 is 0_1 if $b = 0$ and 1_1 otherwise, and similarly, c_2 is 0_2 if $c = 0$ and 1_2 otherwise. □

To conclude the proof, it remains to show that there exists a product of two deterministic context-free languages whose structure has an undecidable first-order theory.

Proposition 2. *There exist two deterministic context-free languages L_1 and L_2 such that $S_{L_1 \otimes L_2}$ has an undecidable first-order theory.*

Proof. Let A be the alphabet containing the symbols $+_\mathbf{1}, +_\mathbf{2}, -_\mathbf{1}, -_\mathbf{2}, =$ and \$. Consider the following deterministic context-free languages:

$$L_1 = \{w \in A^* \mid |w|_{+_\mathbf{1}} = |w|_{-_\mathbf{1}}\}$$
$$L_2 = \{w \in A^* \mid |w|_{+_\mathbf{2}} = |w|_{-_\mathbf{2}}\}.$$

In our argument, we will use reduction from the halting problem of 2-counter machines. A program for a 2-counter machine is a nonempty sequence $I_1; \ldots; I_n$ of instructions, where I_n is a halt instruction and all other instructions I_i are of the form Inc_j, Dec_j or $\mathrm{Test}_j(k)$, $j = 1, 2$, $k \in [n]$. Here, Inc_j increments the value of the jth counter by 1, Dec_j decrements it by 1 – provided that the current value is not 0. If the current value is 0, then Dec_j corresponds to a skip instruction. A conditional branch instruction $\mathrm{Test}_j(k)$, where $k \in [n]$, tests the current value of the jth counter and transfers the control to the kth instruction if this value is 0. Initially, the values of the counters are 0. The instructions are executed sequentially, except for the effect of the conditional branch instructions. The machine halts when I_n is executed. Without loss of generality we will consider machines that do not try to decrease the value of a counter whose value is 0. This condition can be syntactically enforced by prefacing each decrease operation with a test.

Formally, a computation sequence for the program is a sequence $i_1, \ldots, i_m$ of instruction numbers in $[n]$ such that one can define a valuation mapping $v : [m] \mapsto \mathbb{N} \times \mathbb{N}$ associating to every index $\ell \in [m]$, the value of the two counters before executing the ℓth instruction. The computation sequence and the valuation must satisfy the following properties:

- $i_1 = 1$ and $v(1) = (0, 0)$
- for all $\ell \in [m-1]$, if $I_{i_\ell} = \mathrm{Inc}_1$ (resp. $I_{i_\ell} = \mathrm{Inc}_2$), then $i_{\ell+1} = i_\ell + 1$ and $v(\ell+1) = v(\ell) + (1, 0)$ (resp. $v(\ell+1) = v(\ell) + (0, 1)$),
- for all $\ell \in [m-1]$, if $I_{i_\ell} = \mathrm{Dec}_1$ (resp. $I_{i_\ell} = \mathrm{Dec}_2$), then $i_{\ell+1} = i_\ell + 1$ and $v(\ell+1) = v(\ell) + (-1, 0)$ (resp. $v(\ell+1) = v(\ell) + (0, -1)$),
- for all $\ell \in [m-1]$, if $I_{i_\ell} = \mathrm{Test}_1(k)$ (resp. $I_{i_\ell} = \mathrm{Test}_2(k)$), then $v(\ell+1) = v(\ell)$, and $i_{\ell+1} = k$ if the first (resp. second) component of $v(\ell)$ is equal to zero and $i_{\ell+1} = i_\ell + 1$ otherwise.

Furthermore, a computation sequence is halting if $i_m = n$.

A word w over A is said to represent a computation sequence if it satisfies the following conditions:

1. it is of the form $\$^{i_1} x_1 \cdots x_{m-1} \$^{i_m}$ with $x_1, \ldots, x_{m-1} \in \{+_\mathbf{1}, +_\mathbf{2}, -_\mathbf{1}, -_\mathbf{2}, =\}$ and $i_\ell \in [n]$ for $\ell \in [m]$;
2. for all $\ell \in [m-1]$, x_ℓ is $+_\mathbf{1}$ if $I_{i_\ell} = \mathrm{Inc}_1$, $+_\mathbf{2}$ if $I_{i_\ell} = \mathrm{Inc}_2$, $-_\mathbf{1}$ if $I_{i_\ell} = \mathrm{Dec}_1$, $-_\mathbf{2}$ if $I_{i_\ell} = \mathrm{Dec}_2$, and $=$ otherwise;
3. $i_1, \ldots, i_m$ is a computation sequence.

Claim. The exists a first-order formula φ_{Halt} such that for every word $w \in L_1 \otimes L_2$, $S_w \models \varphi_{\mathrm{Halt}}$ if and only if w projected on A represents a halting computation sequence.

Proof. It is straightforward to write a first-order formula ensuring that w projected on A satisfies the conditions 1 and 2 above. To be able to express condition 3 in first-order, the only difficulty consists in testing if before the ℓth instruction the value of a given counter is 0, where $\ell \in [n]$. For this it is enough to notice that if the value of the two counters before the ℓth instruction is given by $(|w_\ell|_{+_\mathbf{1}} - |w_\ell|_{-_\mathbf{1}}, |w_\ell|_{+_\mathbf{2}} - |w_\ell|_{-_\mathbf{2}})$, where $w_\ell = \$^{i_1} x_1 \cdots x_{\ell-1} \$^{i_\ell}$ – recall that we do not consider machines that can decrease the value of a counter when its value is zero –, then by the definition of $L_1 \otimes L_2$, the fact that the jth counter has value 0 can be tested by reading the jth bit of the attached flag.

To conclude the proof, we construct a formula ϕ_P such that $S_{L_1 \otimes L_2} \models \phi_P$ if and only if $L_1 \otimes L_2$ contains at least one word satisfying φ_{Halt} (and hence if and only if P is halting). The formula ϕ_P is equal to $\exists x \; \varphi'_{\text{Halt}}(x)$, where $\varphi'_{\text{Halt}}(x)$ is the formula obtained from φ_{Halt} by relativizing all quantifications to elements that are comparable to x with respect to $<$ or $=$. $\square$

We can now prove Theorem 1.

Proof (Proof of Theorem 1). By Proposition 2, there exist two deterministic context-free languages L_1 and L_2 such that $S_{L_1 \otimes L_2}$ has an undecidable first-order theory. By Proposition 1, there exists a language L over an ordered alphabet which is the disjoint union of deterministic context-free languages (not containing the empty word), such that $S_{L_1 \otimes L_2}$ can be first-order interpreted in W_L. Thus W_L has an undecidable first-order theory. $\square$

5 Corollaries of Theorem 1

Using Lemma 3, Theorem 2 can be transferred to context-free linear orderings.

Corollary 1. *There exists a context-free linear ordering with an undecidable first-order theory. Furthermore such a context-free linear ordering can be defined by a finite disjoint union of deterministic context-free languages.*

Proof. By Theorem 1, there exists a finite union L of deterministic context-free languages such that W_L has an undecidable first-order theory. By Lemma 3, there exists a context-free language L' and a first-order interpretation $\mathcal{I}$ of W_L in $O_{L'}$. Since the first-order theory of W_L is undecidable, it follows that the first-order theory of $O_{L'}$ is also undecidable.

Observe that as L is a finite disjoint union of deterministic context-free languages, the language L' constructed in Lemma 3 can also be chosen to be a finite disjoint union of deterministic context-free languages. $\square$

As all deterministic context-free linear orderings have a decidable monadic second-order theory, this result provides an example of a context-free linear ordering that is not deterministic context-free.

Corollary 2. *There exists a context-free linear ordering that is not a deterministic context-free linear ordering.*

Moving our focus to trees, we obtain a simple proof of a result first proved in [26].

Corollary 3. *There exists a finite disjoint union of deterministic context-free languages such that*

1. *the associated tree has an undecidable first-order theory over the signature σ_{anc} (which includes the ancestor relation),*
2. *it cannot be accepted by any deterministic collapsible automaton.*

Proof. The first claim is a direct consequence of Corollary [1] and Lemma [4]. The second claim then follows form the fact that any language accepted by a deterministic collapsible automaton has a tree with a decidable MSO-theory [25], and hence a decidable first-order theory over the signature σ_{anc}. □

In a draft of this article, we asked if there exists a context-free language whose associated tree has an undecidable first-order theory over the signature σ_{suc}. This question was positively answered by Markus Lohrey [23].

Proposition 3 (M. Lohrey). *There exists a context-free language whose associated tree has an undecidable first-order theory over the signature σ_{suc}.*

Proof. The proof starts by establishing that there exists a context-free language L_0 over an alphabet A such that the following problem is undecidable : "Given a word $w \in A^+$, decide if L_0 contains all words ending with w".

To construct L_0, we consider a universal Turing machine M with a set Q of states and a set Γ of tape symbols. It is well-known that the set L_M of words representing ill-formed or non-terminating computations of a Turing machine is a context-free language [21]. More precisely, a configuration is represented by a word in $\Gamma^* Q \Gamma^*$ and a computation $c_0 \vdash \cdots \vdash c_n$ is represented by the word $w = c_0 \sharp c_1^R \sharp c_2 \cdots$. For the language L_0, we take $L_M^R = \{w^R \mid w \in L_M\}$. For a word w representing an initial configuration c_0 of M, it is clear that L_M^R contains all words ending with w^R if and only if M does not have a halting computation starting from c_0. Hence the aforementioned problem is undecidable for L_0.

Let \$ be a fresh symbol. We show that the first-order theory over σ_{suc} of $T_{L_0\$}$ is undecidable.

Let $w = a_1 \cdots a_n$ with $n > 0$ be a word over A. The formula φ_w over σ_{suc},

$$\begin{aligned}\varphi_w = \forall x\ \exists x_1 \cdots \exists x_{n+1} \quad \text{Succ}(x, x_1) \wedge \textstyle\bigwedge_{i \in [n]} \text{Succ}(x_i, x_{i+1}) \\ \wedge \textstyle\bigwedge_{i \in [n]} P_{a_i}(x_i) \wedge P_{\$}(x_{n+1}),\end{aligned}$$

expresses that for every word u, the word $uw\$$ is in the domain of $T_{L_0\$}$ and hence that uw belongs to L_0. Hence for all $w \in A^+$, $T_{L_0\$} \models \varphi_w$ if and only if L_0 contains all words ending in w. We conclude that the undecidability of the first-order theory of $T_{L_0\$}$ follows from the undecidability of the above problem. □

As observed by M. Lohrey, the above proposition can be stated for the more restrictive signature $\{\text{Succ}\}$ in which labels are omitted. Indeed, this follows from the following lemma that shows that a context-free tree over σ_{suc} can be first-order interpreted in a context-free trees over $\{\text{Succ}\}$.

Lemma 5. *Let L be a context-free language. There exists a context-free language L' such that T_L over the signature σ_{suc} can be first-order interpreted in $T_{L'}$ over the signature* $\{\text{Succ}\}$.

Proof. Let L be a context-free language over the alphabet $A = \{a_1, \ldots, a_n\}$. Without loss of generality we may assume that L contains a nonempty word. Consider the context-free language

$$L' = \{u a_j i \mid i \leq j \text{ and } \exists w \in L, u a_j \leq_p w\}$$

over the alphabet $A \cup [n]$. The first-order interpretation of T_L over σ_{suc} in $T_{L'}$ over $\{\mathrm{Succ}\}$ only keeps non-leaf nodes. The Succ relation is inherited. For $i \in [n]$, the predicate P_{a_i} holds at those non-leaf nodes which have exactly i sons which are leaves. Formally the interpretation is defined by

$$\begin{aligned}
&\delta(x) = \neg\mathrm{Leaf}(x)\\
&\varphi_{\mathrm{Succ}}(x,y) = \mathrm{Succ}(x,y)\\
&\varphi_{P_{a_i}}(x) = \exists^{=i} y\ \mathrm{Succ}(x,y) \wedge \mathrm{Leaf}(y)
\end{aligned}$$

where $\mathrm{Leaf}(x)$ is a formula expressing that x is a leaf. □

It would be interesting to know if Property 3 remains true when only considering context-free languages which are finite unions of deterministic context-free languages.

6 Discussion

In this article, we have established that the linear orderings and trees associated with context-free languages are more complex than those associated to deterministic context-free languages. This result even holds for finite disjoint unions of deterministic context-free languages and hence for unambiguous context-free languages. It would be interesting to investigate whether such a separation result exists for context-free scattered linear orderings.

Acknowledgements. The authors would like to thank Markus Lohrey for his remarks on a previous draft of this article as well as for the proof of Proposition 3. The authors are also grateful for the remarks of the anonymous referees.

References

1. Bloom, S.L., Choffrut, C.: Long words: the theory of concatenation and omega-power. Theoretical Computer Science 259, 533–548 (2001)
2. Bloom, S.L., Ésik, Z.: Deciding whether the frontier of a regular tree is scattered. Fundamenta Informaticae 55, 1–21 (2003)
3. Bloom, S.L., Ésik, Z.: The equational theory of regular words. Information and Computation 197, 55–89 (2005)
4. Bloom, S.L., Ésik, Z.: Regular and Algebraic Words and Ordinals. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 1–15. Springer, Heidelberg (2007)
5. Bloom, S.L., Ésik, Z.: Algebraic ordinals. Fundamenta Informaticae 99, 383–407 (2010)
6. Bloom, S.L., Ésik, Z.: Algebraic linear orderings. Int. J. Foundations of Computer Science 22, 491–515 (2011)
7. Braud, L., Carayol, A.: Linear Orders in the Pushdown Hierarchy. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 88–99. Springer, Heidelberg (2010)
8. Carayol, A., Ésik, Z.: The FC-rank of a context-free language (to appear)

9. Carayol, A., Wöhrle, S.: The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)
10. Caucal, D.: On Infinite Terms Having a Decidable Monadic Theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
11. Caucal, D.: On infinite transition graphs having a decidable monadic theory. Theoretical Computer Science 290, 79–115 (2003)
12. Courcelle, B.: Frontiers of infinite trees. Theoretical Informatics and Applications 12, 319–337 (1978)
13. Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science 25, 95–169 (1983)
14. Delhommé, C.: Automaticité des ordinaux et des graphes homogènes. C. R. Acad. Sci. Paris, Ser. I 339, 5–10 (2004)
15. Ésik, Z.: Representing small ordinals by finite automata. In: 12th Workshop Descriptional Complexity of Formal Systems, Saskatoon, Canada. EPTCS, vol. 31, pp. 78–87 (2010)
16. Ésik, Z.: An undecidable property of context-free linear orders. Information Processing Letters 111, 107–109 (2010)
17. Ésik, Z.: Scattered Context-Free Linear Orderings. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 216–227. Springer, Heidelberg (2011)
18. Ésik, Z., Iván, S.: Hausdorff Rank of Scattered Context-Free Linear Orders. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 291–302. Springer, Heidelberg (2012)
19. Hague, M., Murawski, A., Ong, C.-H.L., Serre, O.: Collapsible Pushdown Automata and Recursion Schemes. In: LICS 2008, pp. 452–461. IEEE (2008)
20. Heilbrunner, S.: An algorithm for the solution of fixed-point equations for infinite words. Theoretical Informatics and Applications 14, 131–141 (1980)
21. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
22. Khoussainov, B., Rubin, S., Stephan, F.: Automatic linear orders and trees. ACM Trans. Comput. Log. 6, 625–700 (2005)
23. Lohrey, M.: Private communication (2012)
24. Lohrey, M., Mathissen, C.: Isomorphism of Regular Trees and Words. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 210–221. Springer, Heidelberg (2011)
25. Luke Ong, C.-H.: On model-checking trees generated by higher-order recursion schemes. In: LICS 2006, pp. 81–90. IEEE Press (2006)
26. Parys, P.: Higher-order stacks can not replace nondeterminism. Note published on the authors webpage (3 p.) (February 2010)
27. Rosenstein, J.G.: Linear Orderings. Pure and Applied Mathematics, vol. 98. Academic Press (1982)
28. Thomas, W.: On frontiers of regular trees. Theoretical Informatics and Applications 20, 371–381 (1986)

Open Bisimulation for Quantum Processes

Yuxin Deng[1,⋆] and Yuan Feng[2,⋆⋆]

[1] Shanghai Jiao Tong University
and Chinese Academy of Sciences, China
[2] University of Technology, Sydney, Australia
and Tsinghua University, China

Abstract. Quantum processes describe concurrent communicating systems that may involve quantum information. We propose a notion of open bisimulation for quantum processes and show that it provides both a sound and complete proof methodology for a natural extensional behavioural equivalence between quantum processes. We also give a modal characterisation of the behavioural equivalence, by extending the Hennessy-Milner logic to a quantum setting.

1 Introduction

The theory of quantum computing has attracted considerable research efforts in the past twenty years. Benefiting from the superposition of quantum states and linearity of quantum operations, quantum computing may provide considerable speedup over its classical analogue [30,13,14].

As is well known, it is very difficult to guarantee the correctness of classical communication protocols at the design stage, and some simple protocols were eventually found to have fundamental flaws. One expects that the design of complex quantum protocols is at least as error-prone, if not more, than in the classical case. In view of the success that classical process algebras [23,18,1] achieved in analyzing and verifying classical communication protocols, several research groups proposed various quantum process algebras with the purpose of modeling quantum protocols. Jorrand and Lalire [21,22] defined a language QPAlg (Quantum Process Algebra) by adding primitives expressing unitary transformations and quantum measurements, as well as communications of quantum states, to a CCS-like classical process algebra. An operational semantics of QPAlg is given, and further a probabilistic branching bisimulation between quantum processes is defined. Gay and Nagarajan [12,11] proposed a language CQP (Communicating Quantum Processes), which is obtained from the pi-calculus [24] by adding primitives for measurements and transformations of quantum states, and allowing transmission of qubits. They presented a type system for CQP, and in particular proved that the semantics preserves typing and that typing guarantees that each qubit is owned by a unique process within a system. A probabilistic branching

⋆ Supported by the Natural Science Foundation of China (61173033 and 61033002).
⋆⋆ Supported by Australian Research Council (FT100100218 and DP110103473).

bisimulation for CQP was proposed by Davidson [3] and shown to be a congruence. The second author of the current paper, together with his colleagues, proposed a language named qCCS [8,31,9] for quantum communicating systems by adding quantum input/output and quantum operation/measurement primitives to classical value-passing CCS [15,16]. One distinctive feature of qCCS, compared to QPAlg and CQP, is that it provides a framework to describe, as well as reason about, the communication of quantum systems which are entangled with other systems. Furthermore, a bisimulation for processes in qCCS has been introduced, and the associated bisimilarity is proven to be a congruence with respect to all process constructors of qCCS. Uniqueness of the solutions to recursive process equations is also established, which provides a powerful proof technique for verifying complex quantum protocols.

In the study of quantum systems, as well as classical communicating systems, an important problem is to tell if two given systems exhibit the same behaviour, as this may allow us to replace a complex system with a simplified but equivalent one. To approach the problem we first need to give criteria for reasonable behavioural equivalence. Two systems should only be distinguished on the basis of the chosen criteria. Therefore, these criteria induce an extensional equivalence between systems, $\approx_{\mathrm{behav}}$, namely the largest equivalence which satisfies them.

Having an independent notion of which systems should, and which should not, be distinguished, one can then justify a particular notion of equivalence, e.g. bisimulation, by showing that it captures precisely the touchstone equivalence. In other words, a particular definition of bisimulation is appropriate because the associated bisimulation equivalence, say $\approx_{bis}$, is *sound* with respect to the touchstone equivalence and provides for it a *complete* proof methodology, i.e. $s_1 \approx_{bis} s_2$ if and only if $s_1 \approx_{\mathrm{behav}} s_2$.

This approach originated in [19] but has now been widely used for different process description languages; for example, see [20,28] for its application to higher-order process languages, [26] for mobile ambients, [10] for asynchronous languages and [6] for probabilistic timed languages. Moreover, in each case the distinguishing criteria are almost the same. The touchstone equivalence should be *compositional* (preserved by some natural operators for constructing systems), *barb-preserving* (equivalent processes exhibit the same observables) and *reduction-closed* (nondeterministic choices are in some sense preserved).

We adapt this approach to quantum processes. Using natural versions of these criteria we obtain an appropriate touchstone equivalence, which we call *reduction barbed congruence*, $\approx_r$. We then develop a theory of bisimulations which is both sound and complete for $\approx_r$. Moreover, we provide a modal characterisation of $\approx_r$ in a quantum logic based on Hennessy-Milner logic [17] by establishing the coincidence of the largest bisimulation with logical equivalence.

Due to lack of space, we omit all proofs; they can be found in [5]. We also refer the readers to [25] for the basic notions of linear algebra and quantum information theory used in this paper.

2 A Probabilistic Model

We review the model of probabilistic labelled transition systems (pLTSs). Later on we will interpret the behaviour of quantum processes in terms of pLTSs because quantum measurements give rise to probability distributions naturally.

We begin with some notations. A (discrete) probability distribution over a set S is a function $\Delta : S \to [0,1]$ with $\sum_{s\in S} \Delta(s) = 1$; the support of such a Δ is the set $\lceil \Delta \rceil = \{\, s \in S \mid \Delta(s) > 0 \,\}$. The point distribution $\overline{s}$ assigns probability 1 to s and 0 to all other elements of S, so that $\lceil \overline{s} \rceil = \{s\}$. In this paper we only need to use distributions with finite support, and let $Dist(S)$ denote the set of finite support distributions over S, ranged over by Δ, Θ etc. If $\sum_{k\in K} p_k = 1$ for some collection of $p_k \geq 0$, and the Δ_k are distributions, then so is $\sum_{k\in K} p_k \cdot \Delta_k$ with $(\sum_{k\in K} p_k \cdot \Delta_k)(s) \;=\; \sum_{k\in K} p_k \cdot \Delta_k(s)$.

Definition 1. *A* probabilistic labelled transition system *is a triple* $\langle S, \mathsf{Act}_\tau, \to \rangle$*, where* S *is a set of states,* Act_τ *is a set of labels* Act *augmented with distinguished element* τ*, and* $\to$ *is a subset of* $S \times \mathsf{Act}_\tau \times Dist(S)$*.*

We often write $s \xrightarrow{\alpha} \Delta$ for $(s, \alpha, \Delta) \in \to$, and $s \xrightarrow{\alpha}$ for $\exists \Delta : s \xrightarrow{\alpha} \Delta$. In a pLTS actions are only performed by states, in that actions are given by relations from states to distributions. But in general we allow distributions over states to perform an action. For this purpose, we *lift* these relations so that they also apply to distributions [7].

Definition 2. *Let* $\mathcal{R} \subseteq S \times Dist(S)$ *be a relation from states to distributions in a pLTS. Then* $\mathcal{R}^\circ \subseteq Dist(S) \times Dist(S)$ *is the smallest relation that satisfies the two rules: (i)* $s \;\mathcal{R}\; \Theta$ *implies* $\overline{s} \;\mathcal{R}^\circ\; \Theta$*; (ii)* $\Delta_i \;\mathcal{R}^\circ\; \Theta_i$ *for all* $i \in I$ *implies* $(\sum_{i\in I} p_i \cdot \Delta_i) \;\mathcal{R}^\circ\; (\sum_{i\in I} p_i \cdot \Theta_i)$ *for any* $p_i \in [0,1]$ *with* $\sum_{i\in I} p_i = 1$*, where* I *is a countable index set.*

We apply this operation to the relations $\xrightarrow{\alpha}$ in the pLTS for $\alpha \in \mathsf{Act}_\tau$, where we also write $\xrightarrow{\alpha}$ for $(\xrightarrow{\alpha})^\circ$. Thus as source of a relation $\xrightarrow{\alpha}$ we now also allow distributions. But note that $\overline{s} \xrightarrow{\alpha} \Delta$ is more general than $s \xrightarrow{\alpha} \Delta$ because if $\overline{s} \xrightarrow{\alpha} \Delta$ then there is a collection of distributions Δ_i and probabilities p_i such that $s \xrightarrow{\alpha} \Delta_i$ for each $i \in I$ and $\Delta = \sum_{i\in I} p_i \cdot \Delta_i$ with $\sum_{i\in I} p_i = 1$.

We write $s \xrightarrow{\hat{\tau}} \Delta$ if either $s \xrightarrow{\tau} \Delta$ or $\Delta = \overline{s}$. We define weak transitions $\overset{\hat{a}}{\Longrightarrow}$ by letting $\overset{\hat{\tau}}{\Longrightarrow}$ be the reflexive and transitive closure of $\xrightarrow{\hat{\tau}}$ and writing $\Delta \overset{\hat{a}}{\Longrightarrow} \Theta$ for $a \in \mathsf{Act}$ whenever $\Delta \overset{\hat{\tau}}{\Longrightarrow} \xrightarrow{a} \overset{\hat{\tau}}{\Longrightarrow} \Theta$. If Δ is a point distribution, we often write $s \overset{\hat{a}}{\Longrightarrow} \Theta$ instead of $\overline{s} \overset{\hat{a}}{\Longrightarrow} \Theta$.

Let $\mathcal{R} \subseteq S \times S$ be a relation between states. It induces a special relation $\hat{\mathcal{R}} \subseteq S \times Dist(S)$ between states and distributions by letting $\hat{\mathcal{R}} \overset{def}{=} \{(s, \overline{t}) \mid s \;\mathcal{R}\; t\}$. Then we can use Definition 2 to lift $\hat{\mathcal{R}}$ to be a relation $(\hat{\mathcal{R}})^\circ$ between distributions. For simplicity, we combine the above two lifting operations and directly write $\mathcal{R}^\circ$ for $(\hat{\mathcal{R}})^\circ$ in the sequel, with the intention that a relation between states can be lifted to a relation between distributions via a special application of Definition 2. In this particular case, it holds that $\Delta \;\mathcal{R}^\circ\; \Theta$ implies $\Theta \;(\mathcal{R}^{-1})^\circ\; \Delta$, where $s \;\mathcal{R}\; t$ iff

$t\ \mathcal{R}^{-1}\ s$ for any $s, t \in S$. This way of lifting relations has elegant mathematical characterisations; see [4] for more details.

3 Quantum CCS

We introduce a quantum extension of classical CCS (qCCS) which was originally studied in [8,31,9]. Three types of data are considered in qCCS: as classical data we have `Bool` for booleans and `Real` for real numbers, and as quantum data we have `Qbt` for qubits. Consequently, two countably infinite sets of variables are assumed: *cVar* for classical variables, ranged over by $x, y, ...$, and *qVar* for quantum variables, ranged over by $q, r, ...$. We assume a set *Exp*, which includes *cVar* as a subset and is ranged over by $e, e', \ldots$, of classical data expressions over `Real`, and a set of boolean-valued expressions *BExp*, ranged over by $b, b', \ldots$, with the usual boolean constants `true`, `false`, and operators $\neg$, $\wedge$, $\vee$, and $\rightarrow$. In particular, we let $e \bowtie e'$ be a boolean expression for any $e, e' \in Exp$ and $\bowtie \in \{>, <, \geq, \leq, =\}$. We further assume that only classical variables can occur freely in both data expressions and boolean expressions. Two types of channels are used: *cChan* for classical channels, ranged over by $c, d, ...$, and *qChan* for quantum channels, ranged over by $\underline{c}, \underline{d}, ...$ A relabelling function f is a map on $cChan \cup qChan$ such that $f(cChan) \subseteq cChan$ and $f(qChan) \subseteq qChan$. Sometimes we abbreviate a sequence of distinct variables $q_1, ..., q_n$ into $\tilde{q}$.

The terms in qCCS are given by:

$$\begin{aligned} P, Q ::= \ & \textbf{nil} \ \mid \ \tau.P \ \mid \ c?x.P \ \mid \ c!e.P \ \mid \ \underline{c}?q.P \ \mid \ \underline{c}!q.P \ \mid \ \mathcal{E}[\tilde{q}].P \ \mid \ M[\tilde{q};x].P \ \mid \\ & P + Q \ \mid \ P \,||\, Q \ \mid \ P[f] \ \mid \ P\backslash L \ \mid \ \textbf{if}\ b\ \textbf{then}\ P \ \mid \ A(\tilde{q};\tilde{x}) \end{aligned}$$

where f is a relabelling function and $L \subseteq cChan \cup qChan$ is a set of channels. Most of the constructors are standard as in CCS [23]. We briefly explain a few new constructors. The process $\underline{c}?q.P$ receives a quantum datum along quantum channel $\underline{c}$ and evolves into P, while $\underline{c}!q.P$ sends out a quantum datum along quantum channel $\underline{c}$ before evolving into P. The symbol $\mathcal{E}$ represents a trace-preserving super-operator applied on the systems $\tilde{q}$. The process $M[\tilde{q};x].P$ measures the state of qubits $\tilde{q}$ according to the observable M and stores the measurement outcome into the classical variable x of P.

Free classical variables can be defined in the usual way, except for the fact that the variable x in the quantum measurement $M[\tilde{q};x]$ is bound. A process P is closed if it contains no free classical variable, i.e. $fv(P) = \emptyset$.

The set of free quantum variables for process P, denoted by $qv(P)$ can be inductively defined as in Figure 1. For a process to be legal, we require that

1. $q \notin qv(P)$ in the process $\underline{c}!q.P$;
2. $qv(P) \cap qv(Q) = \emptyset$ in the process $P \,||\, Q$;
3. Each constant $A(\tilde{q};\tilde{x})$ has a defining equation $A(\tilde{q};\tilde{x}) := P$, where P is a term with $qv(P) \subseteq \tilde{q}$ and $fv(P) \subseteq \tilde{x}$.

The first condition says that a quantum system will not be referenced after it has been sent out. This is a requirement of the quantum no-cloning theorem.

$$\begin{array}{rlrl}
qv(\textsf{\textbf{nil}}) &= \emptyset & qv(\tau.P) &= qv(P) \\
qv(c?x.P) &= qv(P) & qv(c!e.P) &= qv(P) \\
qv(\underline{c}?q.P) &= qv(P) - \{q\} & qv(\underline{c}!q.P) &= qv(P) \cup \{q\} \\
qv(\mathcal{E}[\tilde{q}].P) &= qv(P) \cup \tilde{q} & qv(M[\tilde{q};x].P) &= qv(P) \cup \tilde{q} \\
qv(P+Q) &= qv(P) \cup qv(Q) & qv(P \mid\mid Q) &= qv(P) \cup qv(Q) \\
qv(P[f]) &= qv(P) & qv(P \backslash L) &= qv(P) \\
qv(\textbf{if } b \textbf{ then } P) &= qv(P) & qv(A(\tilde{q};\tilde{x})) &= \tilde{q}.
\end{array}$$

Fig. 1. Free quantum variables

The second condition says that parallel composition $||$ models separate parties that never reference a quantum system simultaneously.

Throughout the paper we implicitly assume the convention that processes are identified up to α-conversion, bound variables differ from each other and they are different from free variables.

We now give the semantics of qCCS. For each quantum variable q we assume a 2-dimensional Hilbert space $\mathcal{H}_q$. For any nonempty subset $S \subseteq qVar$ we write $\mathcal{H}_S$ for the tensor product space $\bigotimes_{q\in S} \mathcal{H}_q$ and $\mathcal{H}_{\overline{S}}$ for $\bigotimes_{q\notin S} \mathcal{H}_q$. In particular, $\mathcal{H} = \mathcal{H}_{qVar}$ is the state space of the whole environment consisting of all the quantum variables, which is a countably infinite dimensional Hilbert space.

Let P be a closed quantum process and ρ a density operator on $\mathcal{H}$,[1] the pair $\langle P, \rho \rangle$ is called a *configuration*. We write *Con* for the set of all configurations, ranged over by $\mathcal{C}$ and $\mathcal{D}$. We interpret qCCS with a pLTS whose states are all the configurations definable in the language, and whose transitions are determined by the rules in Figure 2; we have omitted the obvious symmetric counterparts to the rules *(C-Com)*, *(Q-Com)*, *(Int)* and *(Sum)*. The set of actions Act takes the following form, consisting of classical/quantum input/output actions.

$$\{c?v, c!v \mid c \in cChan, v \in \texttt{Real}\} \cup \{\underline{c}?r, \underline{c}!r \mid \underline{c} \in qChan, r \in qVar\}$$

We use $cn(\alpha)$ for the set of channel names in action α. For example, we have $cn(\underline{c}?x) = \{\underline{c}\}$ and $cn(\tau) = \emptyset$.

In the first eight rules in Figure 2, the targets of arrows are point distributions, and we use the slightly abbreviated form $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ to mean $\mathcal{C} \xrightarrow{\alpha} \overline{\mathcal{C}'}$.

The rules use the obvious extension of the function $||$ on terms to configurations and distributions. To be precise, $\mathcal{C} \mid\mid P$ is the configuration $\langle Q \mid\mid P, \rho\rangle$ where $\mathcal{C} = \langle Q, \rho \rangle$, and $\Delta \mid\mid P$ is the distribution defined by:

$$(\Delta \mid\mid P)(\langle Q, \rho\rangle) \overset{def}{=} \begin{cases} \Delta(\langle Q', \rho \rangle) \text{ if } Q = Q' \mid\mid P \text{ for some } Q' \\ 0 \qquad\qquad\quad \text{otherwise.} \end{cases}$$

Similar extension applies to $\Delta[f]$ and $\Delta\backslash L$.

[1] As $\mathcal{H}$ is infinite dimensional, ρ should be understood as a density operator on some finite dimensional subspace of $\mathcal{H}$ which contains $\mathcal{H}_{qv(P)}$.

(Tau)
$$\langle \tau.P, \rho\rangle \xrightarrow{\tau} \langle P, \rho\rangle$$

(C-Inp)
$$\frac{v \in \texttt{Real}}{\langle c?x.P, \rho\rangle \xrightarrow{c?v} \langle P[v/x], \rho\rangle}$$

(C-Outp)
$$\frac{v = [[e]]}{\langle c!e.P, \rho\rangle \xrightarrow{c!v} \langle P, \rho\rangle}$$

(C-Com)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{c?v} \langle P_1', \rho\rangle \qquad \langle P_2, \rho\rangle \xrightarrow{c!v} \langle P_2', \rho\rangle}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\tau} \langle P_1' \mid\mid P_2', \rho\rangle}$$

(Q-inp)
$$\frac{r \notin qv(\underline{c}?q.P)}{\langle \underline{c}?q.P, \rho\rangle \xrightarrow{\underline{c}?r} \langle P[r/q], \rho\rangle}$$

(Q-Outp)
$$\langle \underline{c}!q.P, \rho\rangle \xrightarrow{\underline{c}!q} \langle P, \rho\rangle$$

(Q-Com)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\underline{c}?r} \langle P_1', \rho\rangle \qquad \langle P_2, \rho\rangle \xrightarrow{\underline{c}!r} \langle P_2', \rho\rangle}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\tau} \langle P_1' \mid\mid P_2', \rho\rangle}$$

(Oper)
$$\langle \mathcal{E}[\tilde{q}].P, \rho\rangle \xrightarrow{\tau} \langle P, \mathcal{E}_{\tilde{q}}(\rho)\rangle$$

(Meas)
$$\frac{M = \sum_{i\in I} \lambda_i E^i \qquad p_i = tr(E^i_{\tilde{q}}\rho)}{\langle M[\tilde{q}; x].P, \rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i \langle P[\lambda_i/x], E^i_{\tilde{q}}\rho E^i_{\tilde{q}}/p_i\rangle}$$

(Int)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\alpha} \Delta \qquad qbv(\alpha) \cap qv(P_2) = \emptyset}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\alpha} \Delta \mid\mid P_2}$$

(Sum)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\alpha} \Delta}{\langle P_1 + P_2, \rho\rangle \xrightarrow{\alpha} \Delta}$$

(Rel)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta}{\langle P[f], \rho\rangle \xrightarrow{f(\alpha)} \Delta[f]}$$

(Res)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta \qquad cn(\alpha) \cap L = \emptyset}{\langle P\backslash L, \rho\rangle \xrightarrow{\alpha} \Delta\backslash L}$$

(Cho)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta \qquad [[b]] = \texttt{true}}{\langle \textbf{if } b \textbf{ then } P, \rho\rangle \xrightarrow{\alpha} \Delta}$$

(Cons)
$$\frac{\langle P[\widetilde{v}/\widetilde{x}, \tilde{r}/\tilde{q}], \rho\rangle \xrightarrow{\alpha} \Delta \qquad A(\widetilde{x}, \tilde{q}) := P}{\langle A(\widetilde{v}, \widetilde{r}), \rho\rangle \xrightarrow{\alpha} \Delta}$$

Fig. 2. Operational semantics of qCCS. Here in rule *(C-Outp)*, $[[e]]$ is the evaluation of e, and in rule *(Meas)*, $E^i_{\tilde{q}}$ denotes the operator E^i acting on the quantum systems $\tilde{q}$.

4 An Extensional Equivalence

Let $\mathcal{C} = \langle P, \rho\rangle$. We use the notation $qv(\mathcal{C}) := qv(P)$ for free quantum variables and $\text{env}(\mathcal{C}) := \text{tr}_{qv(P)}(\rho)$ for partial traces. Let $\Delta = \sum_{i\in I} p_i \cdot \overline{\langle P_i, \rho_i\rangle}$. We write $\mathcal{E}(\Delta)$ for the distribution $\sum_{i\in I} p_i \cdot \overline{\langle P_i, \mathcal{E}(\rho_i)\rangle}$.

We formally define three criteria, namely barb-preservation, reduction-closedness and compositionality, in order to judge whether two processes are equivalent.

Definition 3. *A relation $\mathcal{R}$ is*

– barb-preserving *if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies that $\mathcal{C} \Downarrow_c^{\geq p}$ iff $\mathcal{D} \Downarrow_c^{\geq p}$ for any $p \in [0,1]$ and any classical channel c, where $\mathcal{C} \Downarrow_c^{\geq p}$ holds if $\mathcal{C} \overset{\hat{\tau}}{\Longrightarrow} \Delta$ for some Δ with*

$$\sum\{\Delta(\mathcal{C}') \mid \mathcal{C}' \xrightarrow{c!v} \textit{ for some } v\} \ \geq\ p;$$

– reduction-closed *if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies*

- *whenever $\mathcal{C} \overset{\hat{\tau}}{\Longrightarrow} \Delta$, there exists Θ such that $\mathcal{D} \overset{\hat{\tau}}{\Longrightarrow} \Theta$ and $\Delta\ \mathcal{R}^\circ\ \Theta$,*
- *whenever $\mathcal{D} \overset{\hat{\tau}}{\Longrightarrow} \Theta$, there exists Δ such that $\mathcal{C} \overset{\hat{\tau}}{\Longrightarrow} \Delta$ and $\Delta\ \mathcal{R}^\circ\ \Theta$;*

- compositional *if* $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ *implies* $(\mathcal{C}||R)\ \mathcal{R}\ (\mathcal{D}||R)$ *for any process* R *with* $qv(R)$ *disjoint from* $qv(\mathcal{C}) \cup qv(\mathcal{D})$*, and* $\mathcal{R}$ *is closed under super-operator application, namely* $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ *implies* $\mathcal{E}(\mathcal{C})\ \mathcal{R}\ \mathcal{E}(\mathcal{D})$ *for any* $\mathcal{E} \in \mathcal{TSO}(\mathcal{H}_{\overline{qv(\mathcal{C})}})$*, where* $\mathcal{TSO}(\mathcal{H}_{\overline{qv(\mathcal{C})}})$ *stands for the set of trace-preserving super-operators on finite dimensional subspaces of* $\mathcal{H}_{\overline{qv(\mathcal{C})}}$*.*

Here barb-preservation means that two related configurations have the same probability to send out values on classical channels. Reduction-closure ensures that non-deterministic choices are in some sense preserved. In the definition of compositionality, it is worth noting that we only allow the super-operator $\mathcal{E}$ to be applied on $\mathcal{H}_{\overline{qv(\mathcal{C})}}$. The intuition behind this restriction is that systems in $qv(\mathcal{C})$ are actually the *local* quantum variables of $\mathcal{C}$, and they cannot be manipulated by the outer environment.

Definition 4 (Reduction barbed congruence). *Let* reduction barbed congruence*, written* $\approx_r$*, be the largest relation over configurations which is barb-preserving, reduction-closed and compositional, and furthermore, if* $\mathcal{C} \approx_r \mathcal{D}$ *then* $qv(\mathcal{C}) = qv(\mathcal{D})$ *and* $\mathrm{env}(\mathcal{C}) = \mathrm{env}(\mathcal{D})$*.*

With the above definition, it is difficult to prove if two given configurations are related by reduction barbed congruence. Therefore, we need to discover some proof techniques which are easy to use.

4.1 Open Bisimulations

We now introduce a coinductively defined relation which will be used later on to characterise reduction barbed congruence.

Definition 5. *A relation* $\mathcal{R} \subseteq Con \times Con$ *is an* open simulation *if* $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ *implies that* $qv(\mathcal{C}) = qv(\mathcal{D})$*,* $\mathrm{env}(\mathcal{C}) = \mathrm{env}(\mathcal{D})$*, and for any* $\mathcal{E} \in \mathcal{TSO}(\mathcal{H}_{\overline{qv(\mathcal{C})}})$*,*

- *whenever* $\mathcal{E}(\mathcal{C}) \xrightarrow{\alpha} \Delta$*, there is some* Θ *with* $\mathcal{E}(\mathcal{D}) \overset{\hat{\alpha}}{\Longrightarrow} \Theta$ *and* $\Delta\ \mathcal{R}^\circ\ \Theta$*.*

A relation $\mathcal{R}$ *is an* open bisimulation *if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are open simulations. We let* $\approx_o$ *be the largest open bisimulation.*

Two quantum processes P *and* Q *are bisimilar, denoted by* $P \approx_o Q$*, if for any quantum state* ρ *and any indexed set* $\tilde{v}$ *of classical values, we have*

$$\langle P\{\tilde{v}/\tilde{x}\}, \rho\rangle \approx_o \langle Q\{\tilde{v}/\tilde{x}\}, \rho\rangle.$$

Here $\tilde{x}$ *is the set of free classical variables contained in* P *and* Q*.*

The above definition is inspired by the work of Sangiorgi [27], where a notion of bisimulation is defined for the π-calculus by treating name instantiation in an "open" style (name instantiation happens before any transition). Here we deal with super-operator application in an "open" style, but the instantiation of variables is in an "early" style (variables are instantiated when input actions are performed) because the operational semantics given in Figure 2 is essentially an early semantics. For more variants of semantics, see e.g. [29].

4.2 A Useful Proof Technique

In Definition 5 super-operator application and transitions are considered at the same time. In fact, we can separate the two issues and approach the concept of open bisimulation in an incremental way, which turns out to be very useful when proving that two configurations are bisimilar.

Definition 6. *A relation $\mathcal{R} \subseteq Con \times Con$ is a* ground simulation *if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies that $qv(\mathcal{C}) = qv(\mathcal{D})$,* env$(\mathcal{C})$ = env$(\mathcal{D})$*, and*

– *whenever $\mathcal{C} \xrightarrow{\alpha} \Delta$, there is some distribution Θ with $\mathcal{D} \overset{\hat{\alpha}}{\Longrightarrow} \Theta$ and $\Delta\ \mathcal{R}^\circ\ \Theta$.*

A relation $\mathcal{R}$ is a ground bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are ground simulations.*

Proposition 1. *Suppose that a relation $\mathcal{R}$*

1. *is a ground bisimulation, and*
2. *is closed under all super-operator application.*

Then $\mathcal{R}$ is an open bisimulation.

Proposition 1 provides us with a useful proof technique: in order to show that two configurations $\mathcal{C}$ and $\mathcal{D}$ are open bisimilar, it suffices to exhibit a binary relation including the pair $(\mathcal{C}, \mathcal{D})$, and then to check that the relation is a ground bisimulation and is closed under all super-operator application. This is analogous to a proof technique of open bisimulation for the π-calculus [27], where name instantiation is playing the same role as super-operator application here.

Proposition 2. *$\approx_o$ is the largest ground bisimulation that is closed under all super-operator application.*

For a sanity check, we can prove that $\approx_o$ is an equivalence relation. As a relation between configurations, $\approx_o$ is preserved by all static constructors.

Proposition 3. *If $\langle P, \rho\rangle \approx_o \langle Q, \sigma\rangle$ then*

1. *$\langle P \| R, \rho\rangle \approx_o \langle Q \| R, \sigma\rangle$;*
2. *$\langle P[f], \rho\rangle \approx_o \langle Q[f], \sigma\rangle$;*
3. *$\langle P \backslash L, \rho\rangle \approx_o \langle Q \backslash L, \sigma\rangle$;*
4. *$\langle \textbf{if } b \textbf{ then } P, \rho\rangle \approx_o \langle \textbf{if } b \textbf{ then } Q, \sigma\rangle$.*

We do not have a counterpart of the above proposition for dynamic constructors such as prefix. For example, consider the two configurations taken from [9]: $\langle P, \rho\rangle$ and $\langle Q, \rho\rangle$, where $P = M_{0,1}[q; x].\textbf{nil}$ with $M_{0,1} = \lambda_0|0\rangle\langle 0| + \lambda_1|1\rangle\langle 1|$ being the 1-qubit measurement according to the computational basis, $Q = I[q].\textbf{nil}$, and $\rho = |0\rangle\langle 0|_q \otimes \sigma$ with σ being a state on $\mathcal{H}_{\overline{q}}$. We have $\langle P, \rho\rangle \approx_o \langle Q, \rho\rangle$, but $\langle H[q].P, \rho\rangle \not\approx_o \langle H[q].Q, \rho\rangle$, where H is the Hadamard operator.

Nevertheless, as a relation between processes, $\approx_o$ is preserved by almost all constructors of qCCS.

Theorem 1. *The relation $\approx_o$ between processes is preserved by all the constructors of qCCS except for summation.*

It turns out that reduction barbed congruence can be captured by open bisimulation precisely. This gives a coinductive technique to judge if two configurations are behaviourally equivalent.

Theorem 2 (Soundness). *If $\mathcal{C} \approx_o \mathcal{D}$ then $\mathcal{C} \approx_r \mathcal{D}$.*

In order to obtain completeness, the converse of Theorem 2, we make use of a proof technique that involves examining the barbs of processes in certain contexts; the following technical lemma enhances this technique.

Lemma 1. *If $\Delta||c!0 \ (\approx_r)^\circ \ \Theta||c!0$ where c is a fresh channel, then $\Delta \ (\approx_r)^\circ \ \Theta$.*

We are now in a position to show that $\approx_r$ is complete with respect to $\approx_o$.

Theorem 3 (Completeness). *If $\mathcal{C} \approx_r \mathcal{D}$ then $\mathcal{C} \approx_o \mathcal{D}$.*

Proof. (*Schema*) Since $\approx_r$ is closed under any super-operator application, by Proposition 1 it suffices to show that $\approx_r$ is a ground bisimulation. The key idea is the following. For any transition $\mathcal{C} \xrightarrow{\alpha} \Delta$, we design a test process T, depending on the form of α, such that $\mathcal{C}||T \overset{\hat{\tau}}{\Longrightarrow} \Gamma_1$ for some distribution Γ_1 which exhibits certain barbs. Since $\mathcal{C} \approx_r \mathcal{D}$ we know $\mathcal{C}||T \approx_r \mathcal{D}||T$ by the compositionality of $\approx_r$. Since $\approx_r$ is reduction-closed, there is some Γ_2 such that $\mathcal{D}||T \overset{\hat{\tau}}{\Longrightarrow} \Gamma_2$ and $\Gamma_1 \ (\approx_r)^\circ \ \Gamma_2$. Since $\approx_r$ is barb-preserving, Γ_2 must exhibit similar barbs as Γ_1. The careful design of T ensures that $\mathcal{D} \overset{\hat{\alpha}}{\Longrightarrow} \Theta$ for some Θ with $\Delta \ (\approx_r)^\circ \ \Theta$, and the last step involves Proposition 1. See [5] for more details. $\square$

4.3 Modal Characterisation

We extend the Hennessy-Milner logic by adding a probabilistic choice modality to express the behaviour of distributions, as in [7], and a super-operator modality to express trace-preserving super-operator application, as well as atomic formulae involving projectors for dealing with density operators.

Definition 7. *The class $\mathcal{L}$ of modal formulae over* Act*, ranged over by ϕ, is defined by the following grammar:*

$$\begin{aligned}\phi &:= E_{\tilde{q}}^{\geq p} \mid \textstyle\bigwedge_{i\in I} \phi_i \mid \langle\alpha\rangle\psi \mid \neg\phi \mid \mathcal{E}.\phi \\ \psi &:= \textstyle\bigoplus_{i\in I} p_i \cdot \phi_i\end{aligned}$$

where $\alpha \in \mathsf{Act}_\tau$, $\mathcal{E}$ is a super-operator, and E is a projector associated with a certain subspace of $\mathcal{H}_{\tilde{q}}$. We call ϕ a configuration formula *and ψ a* distribution formula. *Note that a distribution formula ψ only appears as the continuation of a diamond modality $\langle\alpha\rangle\psi$.*

The satisfaction relation *$\models \subseteq Con \times \mathcal{L}$ is defined by*

- *$\mathcal{C} \models E_{\tilde{q}}^{\geq p}$ if $qv(\mathcal{C}) \cap \tilde{q} = \emptyset$ and $\mathrm{tr}(E_{\tilde{q}}\rho) \geq p$ where $\mathcal{C} = \langle P, \rho\rangle$.*
- *$\mathcal{C} \models \bigwedge_{i\in I} \phi_i$ if $\mathcal{C} \models \phi_i$ for all $i \in I$.*

- $\mathcal{C} \models \langle\alpha\rangle\psi$ *if for some* $\Delta \in Dist(Con)$, $\mathcal{C} \overset{\hat{\alpha}}{\Longrightarrow} \Delta$ *and* $\Delta \models \psi$.
- $\mathcal{C} \models \neg\phi$ *if it is not the case that* $\mathcal{C} \models \phi$.
- $\mathcal{C} \models \mathcal{E}.\phi$ *if* $\mathcal{E} \in \mathcal{TSO}(\mathcal{H}_{\overline{qv(\mathcal{C})}})$ *and* $\mathcal{E}(\mathcal{C}) \models \phi$.
- $\Delta \models \bigoplus_{i\in I} p_i \cdot \phi_i$ *if there are* $\Delta_i \in Dist(Con)$ *for all* $i \in I$, *and for all* $\mathcal{D} \in \lceil\Delta_i\rceil$, *with* $\mathcal{D} \models \phi_i$, *such that* $\Delta = \sum_{i\in I} p_i \cdot \Delta_i$.

With a slight abuse of notation, we write $\Delta \models \psi$ above to mean that Δ satisfies the distribution formula ψ. A logical equivalence arises from the logic naturally: we write $\mathcal{C} =^{\mathcal{L}} \mathcal{D}$ if $\mathcal{C} \models \phi \Leftrightarrow \mathcal{D} \models \phi$ for all $\phi \in \mathcal{L}$. Using the logical equivalence, we provide a modal characterisation of reduction barbed congruence as follows.

Theorem 4. *$\mathcal{C} \approx_r \mathcal{D}$ if and only if $\mathcal{C} =^{\mathcal{L}} \mathcal{D}$.*

Proof. (*Schema*) In view of Theorems 2 and 3, it suffices to prove that $\mathcal{C} \approx_o \mathcal{D}$ if and only if $\mathcal{C} =^{\mathcal{L}} \mathcal{D}$. For one direction, we show that $\mathcal{C} \models \phi \Leftrightarrow \mathcal{D} \models \phi$ for all $\phi \in \mathcal{L}$ by structural induction on ϕ; for the other direction, we show that $=^{\mathcal{L}}$ is an open bisimulation by using Proposition 1. □

5 Examples

BB84, the first quantum key distribution protocol developed by Bennett and Brassard in 1984 [2], provides a provably secure way to create a private key between two parties, say, Alice and Bob. Its security relies on the basic property of quantum mechanics that information gain about a quantum state is only possible at the expense of changing the state, if the states to be distinguished are not orthogonal. The basic BB84 protocol goes as follows:

(1) Alice randomly creates two strings of bits $\tilde{B}_a$ and $\tilde{K}_a$, each with size n.
(2) Alice prepares a string of qubits $\tilde{q}$, with size n, such that the ith qubit of $\tilde{q}$ is $|x_y\rangle$ where x and y are the ith bits of $\tilde{B}_a$ and $\tilde{K}_a$, respectively, and $|0_0\rangle = |0\rangle$, $|0_1\rangle = |1\rangle$, $|1_0\rangle = |+\rangle$, and $|1_1\rangle = |-\rangle$. Here the symbols $|+\rangle$ and $|-\rangle$ have their usual meaning: $|+\rangle \overset{def}{=} (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle \overset{def}{=} (|0\rangle - |1\rangle)/\sqrt{2}$.
(3) Alice sends the qubit string $\tilde{q}$ to Bob.
(4) Bob randomly generates a string of bits $\tilde{B}_b$ with size n.
(5) Bob measures each qubit received from Alice according to a basis determined by the bits he generated: if the ith bit of $\tilde{B}_b$ is k then he measures with $\{|k_0\rangle, |k_1\rangle\}$, $k = 0, 1$. Let the measurement results be $\tilde{K}_b$, which is also a string of bits with size n.
(6) Bob sends his choice of measurement bases $\tilde{B}_b$ back to Alice, and upon receiving the information, Alice sends her bases $\tilde{B}_a$ to Bob.
(7) Alice and Bob determine at which positions the bit strings $\tilde{B}_a$ and $\tilde{B}_b$ are equal. They discard the bits in $\tilde{K}_a$ and $\tilde{K}_b$ where the corresponding bits of $\tilde{B}_a$ and $\tilde{B}_b$ do not match.

After the execution of the basic BB84 protocol above, the remaining bits of $\tilde{K}_a$ and $\tilde{K}_b$, denoted by $\tilde{K}'_a$ and $\tilde{K}'_b$ respectively, should be the same, provided that the channels used are perfect, and no eavesdropper exists.

To detect a potential eavesdropper Eve, Alice and Bob proceed as follows:

(8) Alice randomly chooses $\lceil k/2 \rceil$, where k is the size of $\tilde{K}'_a$, bits of $\tilde{K}'_a$, denoted by $\tilde{K}''_a$, and sends Bob $\tilde{K}''_a$ and their indexes in the original string $\tilde{K}'_a$.

(9) Upon receiving the information from Alice, Bob sends back to Alice his substring $\tilde{K}''_b$ of $\tilde{K}'_b$ according to the indexes received from Alice.

(10) Alice and Bob check if the strings $\tilde{K}''_a$ and $\tilde{K}''_b$ are equal. If yes, then the remaining substring $\tilde{K}^f_a$ (resp. $\tilde{K}^f_b$) of $\tilde{K}'_a$ (resp. $\tilde{K}'_b$) by deleting $\tilde{K}''_a$ (resp. $\tilde{K}''_b$) is the secure key shared by Alice (reps. Bob). Otherwise, an eavesdropper is detected, and the protocol halts without generating any secure keys.

For simplicity, we omit the processes of information reconciliation and privacy amplification. Now we describe the above protocol in qCCS. To ease the notations, we assume a special measurement $Ran[\tilde{q};\tilde{x}]$ which can create a string of n random bits, independent of the initial states of the $\tilde{q}$ system, and store it to $\tilde{x}$. In effect, $Ran[\tilde{q};\tilde{x}] = Set^n_+[\tilde{q}].M^n_{0,1}[\tilde{q};\tilde{x}].Set^n_0[\tilde{q}]$ where Set^n_+ (resp. Set^n_0) is the super-operator which sets each of the n qubits it applies on to $|+\rangle$ (resp. $|0\rangle$), $M^n_{0,1}[\tilde{q};\tilde{x}]$ is the quantum measurement on $\tilde{q}$ according to the basis $\{|0\rangle, |1\rangle\}$, and stores the result into $\tilde{x}$. Then the basic BB84 protocol can be defined as

$$\begin{aligned}
Alice &\overset{def}{=} Ran[\tilde{q};\tilde{B}_a].Ran[\tilde{q};\tilde{K}_a].Set_{\tilde{K}_a}[\tilde{q}].H_{\tilde{B}_a}[\tilde{q}].\mathsf{A2B}!\tilde{q}.WaitA(\tilde{B}_a,\tilde{K}_a)\\
WaitA(\tilde{B}_a,\tilde{K}_a) &\overset{def}{=} b2a?\tilde{B}_b.a2b!\tilde{B}_a.key_a!cmp(\tilde{K}_a,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}\\
Bob &\overset{def}{=} \mathsf{A2B}?\tilde{q}.Ran[\tilde{q}';\tilde{B}_b].M_{\tilde{B}_b}[\tilde{q};\tilde{K}_b].b2a!\tilde{B}_b.WaitB(\tilde{B}_b,\tilde{K}_b)\\
WaitB(\tilde{B}_b,\tilde{K}_b) &\overset{def}{=} a2b?\tilde{B}_a.key_b!cmp(\tilde{K}_b,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}\\
BB84 &\overset{def}{=} (Alice\|Bob)\backslash\{a2b, b2a, \mathsf{A2B}\}
\end{aligned}$$

where $Set_{\tilde{K}_a}[\tilde{q}]$ sets the ith qubit of $\tilde{q}$ to the state $|\tilde{K}_a(i)\rangle$, $H_{\tilde{B}_a}[\tilde{q}]$ applies H or does nothing on the ith qubit of $\tilde{q}$ depending on whether the ith bit of $\tilde{B}_a$ is 1 or 0, and $M_{\tilde{B}_b}[\tilde{q};\tilde{K}_b]$ is the quantum measurement on $\tilde{q}$ according to the basis determined by $\tilde{B}_b$, i.e., for each $1 \le k \le n$, it measures q_k with respect to the basis $\{|0\rangle,|1\rangle\}$ (reps. $\{|+\rangle,|-\rangle\}$) if $\tilde{B}_b(k) = 0$ (resp. 1), and stores the result into $\tilde{K}_b(k)$. We also abuse the notation slightly by writing $\mathcal{E}_{\tilde{B}}[\tilde{q}].P$ when we mean $\sum_{\tilde{x}=0^n}^{1^n}(\mathbf{if}\ \tilde{B} = \tilde{x}\ \mathbf{then}\ \mathcal{E}_{\tilde{x}}[\tilde{q}].P)$ where i^n is the all i string of size n, $i = 0, 1$. The function cmp takes a triple of strings $\tilde{x},\tilde{y},\tilde{z}$ with the same size as inputs, and returns the substring of $\tilde{x}$ where the corresponding bits of $\tilde{y}$ and $\tilde{z}$ match. When $\tilde{y}$ and $\tilde{z}$ match nowhere, we let $cmp(\tilde{x},\tilde{y},\tilde{z}) = \epsilon$, the empty string.

To show the correctness of this basic form of BB84 protocol, we let

$$\begin{aligned}
BB84_{spe} \overset{def}{=}\ & Ran[\tilde{q};\tilde{B}_a].Ran[\tilde{q};\tilde{K}_a].Ran[\tilde{q}';\tilde{B}_b].\\
& (key_a!cmp(\tilde{K}_a,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}\|key_b!cmp(\tilde{K}_a,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}).
\end{aligned}$$

The pLTSs of $BB84$ and $BB84_{spe}$ for the special case of $n = 2$ can be depicted as in Figure 3, where for simplicity, we only specify the branch where $\tilde{B}_a = \tilde{K}_a = 00$. Each arrow in the graph denotes a sequence of τ actions, and all probabilistic

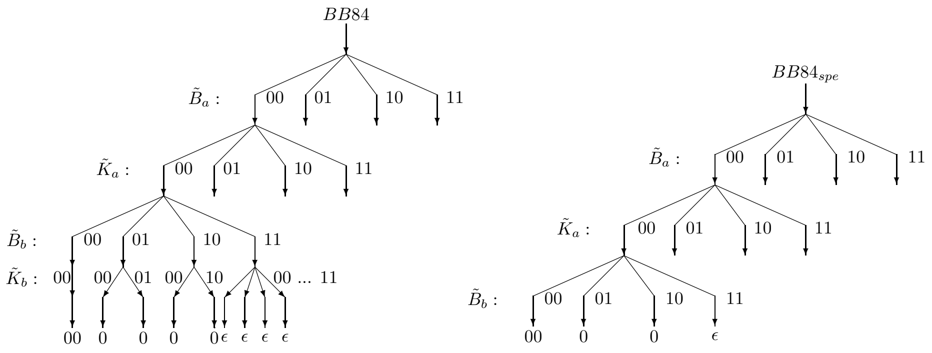


Fig. 3. pLTSs for $BB84$ and $BB84_{spe}$

distributions are uniform. The strings at the bottom line are the outputs of the protocol. Then it can be easily checked from the pLTSs that $BB84 \approx_o BB84_{spe}$. The key is, for each extra branch in $BB84$ caused by the measurement of Bob (the $\tilde{K}_b$ line), the final states are bisimilar; they all output the same string.

Now we proceed to describe the protocol with an eavesdropper. Let

$$\begin{aligned}
Alice' \overset{def}{=} &(Alice\|key_a?\tilde{K}'_a.Pstr_{|\tilde{K}'_a|}[\tilde{q}_a;\tilde{x}].a2b!\tilde{x}.a2b!SubStr(\tilde{K}'_a,\tilde{x}).b2a?\tilde{K}''_b.\\
&(\mathbf{if}\ SubStr(\tilde{K}'_a,\tilde{x}) = \tilde{K}''_b\ \mathbf{then}\ key'_a!RemStr(\tilde{K}'_a,\tilde{x}).\mathbf{nil}\\
&\qquad\qquad\qquad\qquad \mathbf{else}\ alarm_a!0.\mathbf{nil})))\backslash\{key_a\}\\
Bob' \overset{def}{=} &(Bob\|key_b?\tilde{K}'_b.a2b?\tilde{x}.a2b?\tilde{K}''_a.b2a!SubStr(\tilde{K}'_b,\tilde{x}).\\
&(\mathbf{if}\ SubStr(\tilde{K}'_b,\tilde{x}) = \tilde{K}''_a\ \mathbf{then}\ key'_b!RemStr(\tilde{K}'_b,\tilde{x}).\mathbf{nil}\\
&\qquad\qquad\qquad\qquad \mathbf{else}\ alarm_b!0.\mathbf{nil}))\backslash\{key_b\}
\end{aligned}$$

where $|\tilde{x}|$ is the size of $\tilde{x}$, the function $SubStr(\tilde{K}'_a,\tilde{x})$ returns the substring of $\tilde{K}'_a$ at the indexes specified by $\tilde{x}$, and $RemStr(\tilde{K}'_a,\tilde{x})$ returns the remaining substring of $\tilde{K}'_a$ by deleting $SubStr(\tilde{K}'_a,\tilde{x})$. The special measurement $Pstr_m$, which is similar to *Ran*, randomly generates a $\lceil m/2\rceil$-sized string of indexes from $1,\ldots,m$.

To get a taste of the security of $BB84$, we consider a special case where Eve's strategy is to simply measure the qubits sent by Alice, according to randomly guessed bases, to get the keys. She then prepares and sends to Bob a fresh sequence of qubits, employing the same method Alice used to encode keys, but using her own guess of bases and the keys she obtained. That is, we define

$$\begin{aligned}
Eve &\overset{def}{=} \mathsf{A2E}?\tilde{q}.Ran[\tilde{q}'';\tilde{B}_e].M_{\tilde{B}_e}[\tilde{q};\tilde{K}_e].Set_{\tilde{K}_e}[\tilde{q}].H_{\tilde{B}_e}[\tilde{q}].\mathsf{E2B}!\tilde{q}.key'_e!\tilde{K}_e,\\
BB84_E &\overset{def}{=} (Alice'[f_a]\|Eve\|Bob'[f_b])\backslash\{a2b, b2a, \mathsf{A2E}, \mathsf{E2B}\}.
\end{aligned}$$

where $f_a(\mathsf{A2B}) = \mathsf{A2E}$, and $f_b(\mathsf{A2B}) = \mathsf{E2B}$. Let

$$TestBB84 \overset{def}{=} (BB84_E \| key'_a?\tilde{x}.key'_b?\tilde{y}.key'_e?\tilde{z}.$$
$$(\textbf{if } \tilde{x} \neq \tilde{y} \textbf{ then } fail!0.\textbf{nil} \textbf{ else } key_e!\tilde{z}.skey!\tilde{x}.\textbf{nil}))\backslash K$$

where $K = \{key'_a, key'_b, key'_e\}$. It is generally very complicated to prove the security of the full $BB84$ protocol. Here we choose to reduce $TestBB84$ to a simpler process which is easier for further verification. To be specific, we can show that $TestBB84$ is bisimilar to the following process:

$$TB \overset{def}{=} Ran[\tilde{q}; \tilde{B}_a].Ran[\tilde{q}; \tilde{K}_a].Ran[\tilde{q}''; \tilde{B}_e].Ran'_{\tilde{B}_a,\tilde{B}_e,\tilde{K}_a}[\tilde{q}; \tilde{K}_e].Ran[\tilde{q}'; \tilde{B}_b].$$
$$Ran'_{\tilde{B}_e,\tilde{B}_b,\tilde{K}_e}[\tilde{q}; \tilde{K}_b].Pstr_{|\tilde{K}_{ab}|}[\tilde{q}_a; \tilde{x}].$$
$$(\textbf{if } \tilde{K}_{ab} = \tilde{K}_{ba} \textbf{ then } key_e!\tilde{K}_e.skey!RemStr(\tilde{K}_{ab}, \tilde{x}).\textbf{nil}$$
$$\textbf{else } (\textbf{if } \tilde{K}^{\tilde{x}}_{ab} \neq \tilde{K}^{\tilde{x}}_{ba} \textbf{ then } alarm_a!0.\textbf{nil} \| alarm_b!0.\textbf{nil} \textbf{ else } fail!0.\textbf{nil}))$$

where to ease the notations, we let $\tilde{K}_{ab} = cmp(\tilde{K}_a, \tilde{B}_a, \tilde{B}_b)$, $\tilde{K}_{ba} = cmp(\tilde{K}_b, \tilde{B}_a, \tilde{B}_b)$, $\tilde{K}^{\tilde{x}}_{ab} = SubStr(\tilde{K}_{ab}, \tilde{x})$, and $\tilde{K}^{\tilde{x}}_{ba} = SubStr(\tilde{K}_{ba}, \tilde{x})$. Similar to Ran, the special measurement Ran' here, which takes three parameters, delivers a string of n bits. For example, $Ran'_{\tilde{B}_a,\tilde{B}_e,\tilde{K}_a}[\tilde{q}; \tilde{K}_e]$ will first generate a string of $n - |\tilde{K}_{ae}|$ random bits $\tilde{x}$, replace with $\tilde{x}$ the substring of $\tilde{K}_a$ at the positions where $\tilde{B}_a$ and $\tilde{B}_e$ do not match, and store the string after the replacement in $\tilde{K}_e$.

6 Conclusion and Related Work

In our opinion, bisimulation should be considered as a proof methodology for demonstrating behavioural equivalence between systems, rather than providing the definition of the extensional behavioural equivalence itself. We have adapted the well-known *reduction barbed congruence* to obtain a touchstone extensional behavioural equivalence for quantum processes considered in [9], and equipped it with a coinductive proof technique and a modal characterisation.

Below we briefly compare our open bisimulation with other bisimulations for quantum processes proposed in the literature. A branching bisimulation was defined for QPAlg [21,22]. However, it cannot always distinguish different quantum operations, as quantum states are only compared when they are input or output. And the derived bisimilarity is not a congruence; it is not preserved by restriction. Bisimulation defined in [8] indeed distinguishes different quantum operations but it works well only for finite processes. Again, it is not preserved by restriction. In [31], a congruent (strong) bisimulation was proposed for a special model where no classical datum is involved. However, as many important quantum communication protocols such as superdense coding and teleportation cannot be described in that model, its applicability is very limited. Furthermore, as all quantum operations are regarded as visible in [31], the bisimulation is too strong to identify some intuitively equivalent quantum processes.

The first general (both classical and quantum data are involved, and recursive definition is allowed), weak (quantum operations are regarded as invisible, thus can be combined arbitrarily), and congruent bisimulation for quantum processes was defined in [9]. It differentiates quantum input from other actions because, to match a quantum input, an arbitrarily chosen super-operator should be considered. The open bisimulation in this paper makes a step further by treating the super-operator application in an *open* style: applying super-operators before an action to be matched is selected. This makes it possible to separate ground bisimulation and the closedness under super-operator application, and by doing so, we are able to provide not only a neater and simpler definition, but also a powerful technique for proving bisimilarity. Comparing our open bisimulation with the bisimulation in [9], there are two main differences:

1. In [9] a non-standard weak transition $\Longrightarrow\xrightarrow{\underline{c}?q}$ is used to match the transition $\xrightarrow{\underline{c}?q}$. This is for a purely technical reason but makes possible the following example which demonstrates that open bisimulation is strictly coarser. Let $P = \underline{c}?q.(\tau + c!0)$ and $Q = P + \underline{c}?q$. Then P and Q are open bisimilar but not bisimilar in the sense of [9]. This is actually a *classical* example, however, as no quantum operation is included; restricting to this special form of transitions also makes classical bisimulation strictly stronger.
2. In [9] any super-operator application is performed on $\mathcal{H}_{\overline{qv(\mathcal{C}')-q}}$, provided that $\mathcal{C} \xrightarrow{\underline{c}?q} \mathcal{C}'$; while in open bisimulation of this paper, it is performed on $\mathcal{H}_{\overline{qv(\mathcal{C})}}$. As $qv(\mathcal{C}') - q$ can be a proper subset of $qv(\mathcal{C})$, there are more choices of super-operators in the former case. This observation suggests letting $P = \underline{c}?q.\mathcal{E}[q, \widetilde{r}_1] + I[\widetilde{r}_2]$ and $Q = \underline{c}?q.\mathcal{F}[q, \widetilde{r}_1] + I[\widetilde{r}_2]$. We conjecture that by taking suitable $\mathcal{E}$ and $\mathcal{F}$, we will have a real *quantum* example showing that open bisimilarity in this paper is strictly coarser than the bisimilarity in [9].

References

1. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge University Press (1990)
2. Bennett, C.H., Brassard, G.: Quantum cryptography: Public-key distribution and coin tossing. In: Proceedings of the IEEE International Conference on Computer, Systems and Signal Processing, pp. 175–179 (1984)
3. Davidson, T.A.S.: Formal Verification Techniques using Quantum Process Calculus. PhD thesis, University of Warwick (2011)
4. Deng, Y., Du, W.: Logical, metric, and algorithmic characterisations of probabilistic bisimulation. Technical Report CMU-CS-11-110. Carnegie Mellon University (March 2011)
5. Deng, Y., Feng, Y.: Open bisimulation for quantum processes. Full Version of the current paper, http://arxiv.org/abs/1201.0416
6. Deng, Y., Hennessy, M.: On the Semantics of Markov Automata. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 307–318. Springer, Heidelberg (2011)
7. Deng, Y., van Glabbeek, R., Hennessy, M., Morgan, C.: Testing Finitary Probabilistic Processes (Extended Abstract). In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 274–288. Springer, Heidelberg (2009)

8. Feng, Y., Duan, R., Ji, Z., Ying, M.: Probabilistic bisimulations for quantum processes. Information and Computation 205(11), 1608–1639 (2007)
9. Feng, Y., Duan, R., Ying, M.: Bisimulation for quantum processes. In: Proc. POPL 2011, pp. 523–534. ACM (2011)
10. Fournet, C., Gonthier, G.: A hierarchy of equivalences for asynchronous calculi. Journal of Logic and Algebraic Programming 63(1), 131–173 (2005)
11. Gay, S.J., Nagarajan, R.: Types and typechecking for communicating quantum processes. Mathematical Structures in Computer Science 16(03), 375–406 (2006)
12. Gay, S.J., Nagarajan, R.: Communicating quantum processes. In: Palsberg, J., Abadi, M. (eds.) Proc. POPL 2005, pp. 145–157 (2005)
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. ACM STOC, pp. 212–219 (1996)
14. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Physical Review Letters 78(2), 325 (1997)
15. Hennessy, M.: A proof system for communicating processes with value-passing. Formal Aspects of Computer Science 3, 346–366 (1991)
16. Hennessy, M., Ingólfsdóttir, A.: A theory of communicating processes value-passing. Information and Computation 107(2), 202–236 (1993)
17. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32(1), 137–161 (1985)
18. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
19. Honda, K., Tokoro, M.: On reduction-based process semantics. Theoretical Computer Science 151(2), 437–486 (1995)
20. Jeffrey, A., Rathke, J.: Contextual equivalence for higher-order pi-calculus revisited. Logical Methods in Computer Science 1(1:4) (2005)
21. Jorrand, P., Lalire, M.: Toward a quantum process algebra. In: Proceedings of the 1st Conference on Computing Frontiers, pp. 111–119. ACM (2004)
22. Lalire, M.: Relations among quantum processes: Bisimilarity and congruence. Mathematical Structures in Computer Science 16(3), 407–428 (2006)
23. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
24. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, Parts I and II. Information and Computation 100, 1–77 (1992)
25. Nielsen, M., Chuang, I.: Quantum computation and quantum information. Cambridge University Press (2000)
26. Rathke, J., Sobociński, P.: Deriving Structural Labelled Transitions for Mobile Ambients. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 462–476. Springer, Heidelberg (2008)
27. Sangiorgi, D.: A theory of bisimulation for the pi-calculus. Acta Informatica 33(1), 69–97 (1996)
28. Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. In: Proc. LICS 2007, pp. 293–302. IEEE Computer Society (2007)
29. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
30. Shor, P.W.: Algorithms for quantum computation: discrete log and factoring. In: Proc. FOCS 1994, pp. 124–134 (1994)
31. Ying, M., Feng, Y., Duan, R., Ji, Z.: An algebra of quantum processes. ACM Transactions on Computational Logic 10(3), 1–36 (2009)

A Modular LTS for Open Reactive Systems*

Fabio Gadducci, Giacoma Valentina Monreale, and Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italy

Abstract. The theory of reactive systems (RSs) represents a fruitful proposal for deriving labelled transition systems (LTSs) from unlabelled ones. The synthesis of an LTS allows for the use of standard techniques in the analysis of systems, as witnessed by the widespread adoption of behavioral semantics. Recent proposals addressed one of the main drawbacks of RSs, namely, its restriction to the analysis of ground (i.e., completely specified) systems. A still unresolved issue concerns the lack of a presentation via inference rules for the derived LTS, thus hindering the modularity of the presentation. Our paper considers open RSs. We first introduce a variant of the current proposal based on "luxes": our technique is applicable to a larger number of case studies and, under some conditions, it synthesises a smaller LTS. Then, we illustrate how the LTS derived by using our approach can be equipped with a SOS-like presentation via an encoding into tile systems.

Keywords: Open reactive systems, labelled transitions, tile systems.

1 Introduction

The ever increasing diffusion of concurrent and distributed systems stimulated the development of novel formalisms for their specification. These formalisms usually provide an abstract presentation of the behaviour of such a system by resorting to some kind of operational description, possibly adopting also an observational equivalence over the system configurations.

At its simplest, the dynamics of a computational model is defined by means of a *reduction semantics*: a set representing the possible states of the system, plus an unlabelled relation among these states, denoting the potential evolutions of the system. The set of states is often provided by means of an equational specification, referred to as "structural congruence" in the process calculi literature, stating which presentations intuitively specify the same system, up to a syntactical rearrangement of its components.

Despite the advantage of conveying the semantics with relatively few rules, the main drawback of reduction semantics is that the dynamics of a system is described in a monolithic way. Thus, it can be observationally interpreted only by inserting a system in appropriate contexts, where a reduction may take place.

* Partly supported by the EU FP7-ICT IP ASCEns and by the MIUR PRIN SisteR.

To ease the analysis of systems, it is often necessary to consider descriptions allowing the analysis of the behaviour of each single subcomponent, thus increasing modularity and enhancing the opportunities for verification. In such a context, labelled transition systems (LTSs) represent the most widely used tool. LTSs open the way for the definition of observational equivalences, abstractly characterising when two systems have the same behaviour, thus allowing the possibility of verifying the properties of system composition. However, the identification of the "right" labels is a difficult task and it is usually left to the ingenuity of the researcher. A case at hand is the Calculus of Mobile Ambients (MA) [5]: despite its rapid acceptance and the intense scrutiny it was subject to, the development of a suitable labelled semantics defied the researchers until quite recently [11], and chances are that it might not be fully settled [1,15].

The theory of *reactive systems* (RSs) [9] represents one of the most successful meta-frameworks among those addressing the need of deriving suitable LTSs and behavioural equivalences starting from reduction semantics. The key idea is simple: a system h has a labelled transition $h \xrightarrow{c} h'$ if the system obtained by inserting h inside the "minimal" context c may reduce to h'. The framework uses categories to model the state space of a formalism and it exploits the categorical notion of relative pushout (RPO) to capture the intuitive notion of "minimal" environment into which a system specification has to be inserted, in order to allow for a reduction to occur. However, in several examples where calculi with even simple structural congruences are considered, the theory cannot be immediately applied. So, A futher advance is represented by G-reactive systems (GRSs), [16], a 2-categorical extension of RSs that allows for an easier representation of those calculi such that the structural congruence is an integral part of the theory.

A success story for GRSs is represented by the application to MA, and the proposal of slender LTSs, yet inducing the same behavioural semantics of the original proposal [11] (see [1,15]). Despite its applicability, the main limit suffered by GRSs is the restriction to the use of ground rules for describing the dynamics of a system: often a strong requirement, in the modelling of open-ended systems operating in an ever changing environment. Consider e.g. Milner's CCS, and the reduction rule modelling the communication over a channel a, namely $a.P \mid \bar{a}.Q \to P \mid Q$. In the GRS framework, it is represented by an infinite set of ground rules, one for each possible pair P and Q of processes. One would instead like to have a more general and succinct theory allowing to express parametric rules such as $a.1 \mid \bar{a}.2 \to 1 \mid 2$, for 1 and 2 placeholders that can be freely instantiated.

This paper aims at addressing this problem, by developing a theory considering also open terms and parametric rewriting rules. So now the transitions will be labelled not only with the minimal context but also with the most general instantiation allowing a reduction. More explicitly, $h \xrightarrow[x]{c} h'$ if h instantiated with the (possibly open) term x and inserted into the context c may evolve into a state h'. Quite intuitively, our proposal exploits the notions of RPO and the symmetric relative pullback (RPB), suitably extended to GRSs, to capture the notions of minimal context and most general instantiation, respectively.

The framework recalls the one in [8] based on *luxes*, the only approach we are aware of for an open variant of GRSs. Their formalism advances a notion of labelled transition that captures at once the requirements on context and substitution. Our proposal weakens the constraints and it can thus be applied to more case studies, as well as generating, under some conditions, a smaller LTS.

Another clear limitation of the standard GRSs approach is represented by the lack of a finitary presentation of the derived LTS, possibly via a set of inference rules according to the SOS-style. So, following the solution for closed GRSs suggested in [3], we draw the connection with the tile model [6], showing how to build a (double) category of "squares" that generates precisely the LTS derived by an open GRS according to the proposal offered in this paper.

The main concepts of the paper are illustrated by a small running example, based on a Simplified Calculus of Mobile Ambients (SMA). Despite its minimal syntax, with respect to more basic, CCS-like calculi, SMA will allow us to better show the usefulness of taking into account contexts and instantiations.

The paper is organized as follows. Sect. 2 briefly recalls 2-categories. Sect. 3 introduces open GRSs and the technique that we propose to derive LTSs, while Sect. 4 presents a preliminary comparison with the theory in [8]. Sect. 5 shows how to construct a tile system simulating the reductions of an open GRS. Sect. 6 concludes the paper by illustrating some venues for further works.

2 Some Background on 2-Categories

This section introduces the 2-categorical definitions used later on (see [4,7]).

Definition 1 (2-category). *A 2-category* $\mathbf{C}$ *consists of*

1. *a family of* objects $a, b, c, \ldots$*;*
2. *for each* $a, b \in \mathbf{C}$ *a category* $\mathbf{C}(a, b)$*. The objects of* $\mathbf{C}(a, b)$ *are called* 1-cells *or arrows and denoted by* $f : a \to b$*. Identity arrows are instead denoted by* $id_a : a \to a$*. The morphisms are called* 2-cells*, and are written* $\alpha : f \Rightarrow g : a \to b$*. Composition in* $\mathbf{C}(a, b)$ *is denoted by* $\bullet$ *and referred to as* vertical *composition. Identity 2-cells are denoted by* $1_f : f \Rightarrow f$*;*
3. *for each* $a, b, c \in \mathbf{C}$ *a functor* $* : \mathbf{C}(a, b) \times \mathbf{C}(b, c) \to \mathbf{C}(a, c)$*, called* horizontal *composition. It is associative and admits* 1_{id_a} *as identities.*

The functoriality of $*$ amounts to imposing the *exchange law* $(\alpha \bullet \gamma) * (\beta \bullet \delta) = (\alpha * \beta) \bullet (\gamma * \delta)$ for any 4-tuple α, β, γ and δ of composable cells.

2-categories are used to simulate the reduction semantics of formalisms [14,12], where reductions are modelled by 2-cells. Starting from an abstract presentation of the basic reduction steps of a system, the closure with respect to contexts is then obtained by the 2-categorical operation of whiskering [7].

Definition 2 (G-category). *A* groupoidal category *(G-category) is a 2-category whose 2-cells are invertible.*

We will show an example of G-category based on the PROP category [10].

Definition 3. *A product and permutation (PROP) category* $\mathbf{C}$ *has natural numbers* $0, 1, 2, \ldots$ *as objects and it is equipped with two further structures*

– *for each* n*, the group of permutations* $S(n)$ *of* n *elements is a subgroup of all the invertible elements of the homset* $\mathbf{C}(n, n)$*. The identity permutation is the identity morphism* $1_n : n \to n$*;*
– *a functor* $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$*, called* product *and written between its arguments, which acts as addition on the objects, i.e.,* $m \otimes n = m + n$*, and such that*
  1. *it is associative, i.e.,* $(f \otimes f') \otimes f'' = f \otimes (f' \otimes f'')$*;*
  2. *given* $\sigma \in S(n)$ *and* $\sigma' \in S(n')$*, we have* $\sigma \otimes \sigma' = \sigma \times \sigma' : n + n' \to n + n'$*, where* $\times$ *denotes the product of permutations;*
  3. *given* $f : m \to n$ *and* $f' : m' \to n'$*, we have that* $\gamma_{n,n'}(f \otimes f') = (f' \otimes f)\gamma_{m,m'}$*, where* $\gamma_{n,n'} : n + n' \to n + n'$ *denotes the permutation in* $S(n + n')$ *interchanging the first block of* n *and the second block of* n'*.*

A G-PROP is a PROP where the underlying category is a G-category.

Example 1 (GPROP PA2CP). We now show how a signature of a calculus (modulo term equations) may induce a G-PROP. Inspired by the characterization of (a fragment of) CCS in [8], we model a simplified version of Mobile Ambients (MA). The richer structure of this calculus, as well as the complexity of its basic reduction rules, will later allow us to highlight the need of considering open terms and parametric reduction rules.

Let us then denote SMA the finite, restriction free fragment of MA with only the *in* action. The syntax of the calculus is shown on the left of the upper row of Fig. 1. We assume a set $\mathcal{N}$ of *names* ranged over by $m, n, u, \ldots$ We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of closed processes, containing no holes.

The semantics is given by the combination of an equivalence between processes, the structural congruence, and a relation among them. The former, denoted by $\equiv$, is induced by the two rightmost axioms in the first row of Fig. 1. It is used to define the reduction relation $\rightsquigarrow$, which is inductively generated by the axioms and inference rules shown in the lower rows of Fig. 1. It is used only later on, but we introduce it here for the sake of presentation.

$P ::= \epsilon,\ n[P],\ in\,n.P,\ P_1|P_2$ $\qquad$ $(P|Q)|R \equiv P|(Q|R)$ $\qquad$ $P|Q \equiv Q|P$

$n[in\ m.P|Q]|m[R] \rightsquigarrow m[n[P|Q]|R]$ $\qquad$ if $P \rightsquigarrow Q$ then $P|R \rightsquigarrow Q|R$

if $P \rightsquigarrow Q$ then $n[P] \rightsquigarrow n[Q]$ $\qquad$ if $P' \equiv P, P \rightsquigarrow Q, Q \equiv Q'$ then $P' \rightsquigarrow Q'$

Fig. 1. Syntax, structural congruence and reduction relation of SMA

The signature corresponding to SMA is $\Sigma = \epsilon : 0, n[\ ] : 1, in\,n. : 1, | : 2$.

The G-PROP PA2CP has arrows $h : m \to n$, representing n-tuples of terms over Σ, quotiented by associativity, that altogether contain m distinct holes. Permutations in (n, n) are tuples built solely of holes, $\otimes$ acts on arrows as tuple juxtaposition, and arrow composition is term substitution.

To define the 2-cells of the category, an explicit representation of the arrows of the category is used. A term can be indeed represented as a finite, ordered tree with nodes of any degree, where an immediate child of a node of degree higher than 1 must have degree at most 1. Leaves of such a tree correspond to occurrences of the constant ϵ. Nodes of degree 1 correspond to an application either of an ambient operator or of a capability. In the former case the node is labelled with a name belonging to $\mathcal{N}$, while in the latter case it is labelled with the capability followed by an ambient name. Finally, nodes of higher degree correspond to term fragments built solely of the parallel operator. So, arrows can be represented as tuples of these trees.

A 2-cell from h to h' models the equivalence of h and h' according to the commutative axiom. So, it is a family, indexed by the nodes of (the explicit representation of) h, of permutations on the sets of their immediate children, such that the application of all these permutations to h yields h'.

3 A New LTS for Open Reactive Systems

This section presents an extension of *G-reactive systems* (GRSs) [16]. The theory aims at deriving labelled transition systems (LTSs) for specification formalisms whose operational semantics is provided by reduction rules. The technique was originally given for closed systems, that is, closed terms and ground reduction rules. As in [8], our proposal considers terms with variables and parametric rules. The idea is simple: a system specified by an open term h has a labelled transition $h \xrightarrow[x]{c} h'$ if h may evolve into a state h' after being instantiated with the (possibly open) term x and inserted into the context c.

A G-category $\mathbf{C}$ models the syntax of a formalism. An *(open) system* is an arrow $h : a_1 \to a_2$: it can be plugged into $g : a_2 \to a_3$ via arrow composition. Given arrows $h, g : a_1 \to a_2$, a 2-cell $\alpha : h \Rightarrow g$ represents an isomorphism (i.e., a proof of equivalence) between systems h and g. The semantics is given via *reduction rules*: pairs of systems $\langle l, r \rangle$ belonging to the same hom-set.

Definition 4 (Open GRS). *An* open G-reactive system *(GRS)* $\mathbb{C}$ *consists of*

1. *a G-category* $\mathbf{C}$*;*
2. *a composition-reflecting, 2-cell closed, subcategory* $\mathbf{D}$ *of* reactive contexts*;*
3. *a set* $\mathfrak{R} \subseteq \bigcup_{a_1,a_2 \in |\mathbf{C}|} \mathbf{C}(a_1, a_2) \times \mathbf{C}(a_1, a_2)$ *of* reduction rules.

Intuitively, reactive contexts are those arrows inside which a reduction can occur. By 2-cell closed we mean that $d \in \mathbf{D}$ and $\alpha : d \Rightarrow d'$ in $\mathbf{C}$ implies $d' \in \mathbf{D}$, while by composition-reflecting we mean that $d'; d \in \mathbf{D}$ implies $d, d' \in \mathbf{D}$.

Given an open GRS $\mathbb{C}$, the reduction relation over the terms of $\mathbb{C}$ is generated by closing the reduction rules under all reactive contexts, instantiations and 2-cells. Formally, the *reduction relation* is defined by taking $h \rightsquigarrow h'$ if there exist $\langle l, r \rangle \in \mathfrak{R}$, $d \in \mathbf{D}$, $x \in \mathbf{C}$, $\alpha : h \Rightarrow x; l; d$ and $\alpha' : h' \Rightarrow x; r; d$.

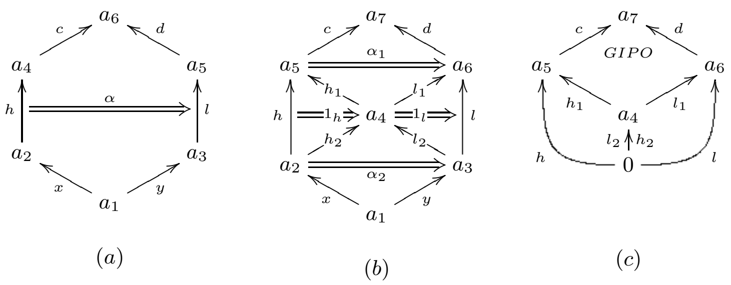
Example 2. Consider the G-category G-PROP PA2CP shown in Example 1. We can then construct an open GRS $\mathbb{C}_{SMA}$ over it by taking as the set of reduction rules the set $\bigcup_{n,m\in\mathcal{N}}\{\langle n[in\, m.1 \mid 2] \mid m[3], m[n[1 \mid 2] \mid 3]\rangle\}$ (denoting here and in the following, with an abuse of notation, id_i with i for $i = 1, 2, 3$), and as the subcategory of reactive contexts the smallest composition-reflecting, 2-cell closed subcategory including arrows of the shape $1 \mid P : 1 \to 1$ and $n[1] : 1 \to 1$. Note that it does not contain contexts with a hole after a capability.

The behaviour of an open GRS is given by an unlabelled transition system. To obtain a labelled one, we instantiate an open system h with a subterm x, plug the result into a context c and observe if a reduction occurs. Categorically, it means that $x; h; c$ is isomorphic to $y; l; d$ (there exists $\alpha : x; h; c \Rightarrow y; l; d$) for an instantiation x, a rule $\langle l, r\rangle$, and a reactive context d (Fig. 2(a)).

The resulting LTS is often infinite-branching, since any context (or instantiation) allowing a reduction may occur as label. It also has redundant transitions: the SMA open term $T = n[in\, m.1 \mid 2]$ would have both transitions $T \xrightarrow[\langle P,Q\rangle]{1|m[R]} m[n[P \mid Q] \mid R]$ and $T \xrightarrow[\langle P,Q\rangle]{1|m[R]|S} m[n[P \mid Q] \mid R] \mid S$, yet S does not "concur" to the reduction. We thus consider only "minimal contexts allowing a reduction", captured by the categorical notion of groupoidal-idem pushouts (GIPOs) in G-categories, and the "most general instantiations", modelled by the categorical notion of groupoidal-idem pullbacks (GIPBs). We refer to [16] for the notion of GIPO, the one of GIPB being perfectly symmetric.

Definition 5 (GIPO-GIPB LTS). *Let $\mathbb{C}$ be an open GRS and* **C** *its underlying G-category. The* GIPO-GIPB LTS GGLTS($\mathbb{C}$) *is defined as follows*

- *states: $h : a_1 \to a_2$ in* **C***, for arbitrary a_1 and a_2;*
- *transitions: $h \xrightarrow[x]{c} h'$ if there exist $d \in \mathbf{D}$ and $\langle l, r\rangle \in \mathfrak{R}$ such that the upper rhombus of the diagram in the middle of Fig. 2 is a GIPO, the lower one is a GIPB, and $h' = y; r; d$.*



$$\begin{tikzcd}
 & a_6 & \\
a_4 \arrow[ur, "c"] \arrow[rr, Rightarrow, "\alpha"] & & a_5 \arrow[ul, "d"'] \\
a_2 \arrow[u, "h"] & & a_3 \arrow[u, "l"'] \\
 & a_1 \arrow[ul, "x"] \arrow[ur, "y"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & a_7 & \\
a_5 \arrow[ur, "c"] \arrow[rr, Rightarrow, "\alpha_1"] & & a_6 \arrow[ul, "d"'] \\
 & a_4 \arrow[ul, "h_1"] \arrow[ur, "l_1"'] & \\
a_2 \arrow[uu, "h"] \arrow[ur, "h_2"] \arrow[rr, Rightarrow, "\alpha_2"] & & a_3 \arrow[uu, "l"'] \arrow[ul, "l_2"'] \\
 & a_1 \arrow[ul, "x"] \arrow[ur, "y"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & a_7 & \\
a_5 \arrow[ur, "c"] & GIPO & a_6 \arrow[ul, "d"'] \\
 & a_4 \arrow[ul, "h_1"] \arrow[ur, "l_1"'] & \\
 & 0 \arrow[u, "l_2", "h_2"'] \arrow[uul, bend left, "h"] \arrow[uur, bend right, "l"'] &
\end{tikzcd}$$

Fig. 2. The 2-cell $\alpha : x; h; c \Rightarrow y; l; d$, a labelled transition $h \xrightarrow[x]{c}$ and a GIPO

Example 3. Consider the open GRS $\mathbb{C}_{SMA}$ introduced in Example 2. The leftmost diagram of Fig. 3 shows the derived transition $\langle n[in\, m.1 \mid Q], 3\rangle \xrightarrow{1|m[3]} m[n[1 \mid Q] \mid 3]$ (omitting here and in the following any identity instantiation such as $\langle 1, 3\rangle$ in a label). The initial state offers an ambient containing a capability $in\, m$ in parallel with a closed process Q, while the environment provides the ambient m which is going to be in parallel with n, so the reduction can occur. Only the identity instantiation is instead provided.

The transition $\langle n[1 \mid Q], 3\rangle \xrightarrow[\langle in\, m.1,3\rangle]{1|m[3]} m[n[1 \mid Q] \mid 3]$ is shown in the diagram on the right of Fig. 3: it needs both a context and an instantiation. The initial open term again offers an ambient n, but now it contains a hole in parallel with Q. Therefore, the instantiation provides the capability $in\, m.1$, replacing the hole inside n, while the environment once more provides the ambient m.

This transition shows the importance of using both instantiations and contexts. In this case, we can note the interaction of the three parts occurring in the reduction: the initial state, the instantiation and the context. This situation never occurs in simpler calculi such as CCS, where we can only have the synchronization between two systems in parallel.

Remark 1. The original notion of closed GRS [16] is very similar to the one for open GRSs. One difference is that in the category $\mathbf{C}$ there exists a chosen object 0, which is the codomain of no arrow and denotes the lack of holes. Arrows having 0 as domain are deemed to represent closed terms. Another difference is that the rules are closed, i.e., they are pairs of closed terms, and also the states of the derived (GIPO) LTS are so: $h \xrightarrow{c} h'$ if there exist $d \in \mathbf{D}$ and $\langle l, r\rangle \in \mathfrak{R}$, such that the square h, c, l, d is a GIPO and $h' = r; d$.

Our proposal may be specialized to consider either closed terms or closed rules, and it then subsumes the standard GRS formalism. Indeed, let us assume that rules and terms are ground. This means that in the diagram (b) of Fig. 2 the objects a_1, a_2 and a_3 are equal to 0. We therefore obtain the diagram (c) of Fig. 2, and if e.g. l_2 is an epimorphism then the external square of diagram (c) is a GIPO, and so $h \xrightarrow{c}$ in the standard GIPO LTS.

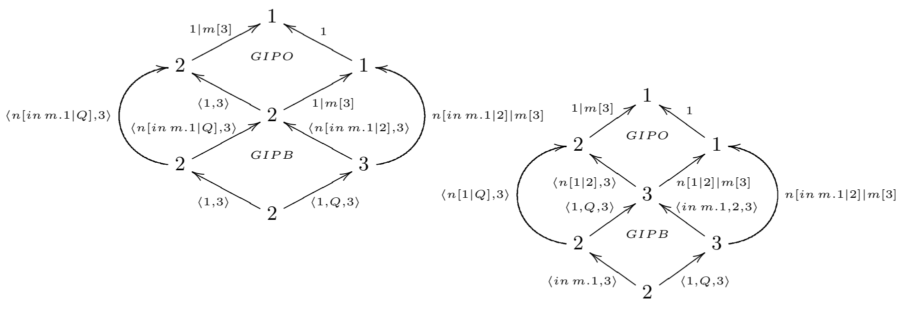


Fig. 3. Transitions $\langle n[in\, m.1 \mid Q], 3\rangle \xrightarrow{1|m[3]} T$ and $\langle n[1 \mid Q], 3\rangle \xrightarrow[\langle in\, m.1,3\rangle]{1|m[3]} T$

4 On G-Luxes

This section presents a preliminary comparison between the technique presented in the previous section and the one in [8]. These proposals share the same idea: labels should include both the minimal context and the most general instantiation allowing a reduction. As we will argue, the main conceptual difference is that in [8] the authors introduce the novel notion of *G-locally universal hexagon* (*G-lux*) in order to capture at the same time the two components of a label via a suitable universal property, while our proposal considers contexts and instantiations separately, thus resorting to standard GIPOs and GIPBs, respectively.

Our definition turns out to encompass a wider range of case studies. Indeed, categories with G-luxes always have GIPOs and GIPBs [8, Theorem 1], and from that fact our statement follows. We now recall the notion of G-lux LTS adopting a simple characterization via GIPOs and GIPBs [8, Lemma 10].

Definition 6 (G-lux). *The hexagon (a) in Fig. 2 is a G-locally universal hexagon (G-lux) if in Fig. 4, (a) is a GIPO and (b) is a GIPB.*

A G-category has G-luxes if every hexagon as diagram (a) in Fig. 2 has an inner G-lux as diagram (c) of Fig. 4, and $\alpha = \delta * 1_h * \beta \bullet 1_z * \psi * 1_e \bullet \varphi * 1_l * \gamma$.

Definition 7 (GLUX Transition System). *Let* $\mathbb{C}$ *be an open GRS and* $\mathbf{C}$ *its underlying G-category. The* GLUX LTS *(*$\text{LLTS}(\mathbb{C})$*) is defined as follows*

- *states:* $h : a_1 \to a_2$ *in* $\mathbf{C}$*, for arbitrary* a_1 *and* a_2*;*
- *transitions:* $h \xrightarrow[x]{c} h'$ *if there exist* $d \in \mathbf{D}$*,* $\langle l, r\rangle \in \mathfrak{R}$ *and* $\alpha : x; h; c \Rightarrow y; l; d$ *such that the diagram (a) of Fig. 2 is a G-lux and* $h' = y; r; d$*.*

The diagrams of Fig. 5 show two examples of G-luxes in the category PA2CP. The left-most one gives the labelled transition $n[in\, m.1 \mid 2] \xrightarrow{1|m[R]} m[n[1 \mid 2] \mid R]$. This transition shows one of the main problems luxes suffer from, already discussed in the last paragraph of [8, Example 5]: the intertwining of context and instantiation may sometimes result in the offering of components that are not necessary for the reduction. In particular, in the example R is redundant, since it appears both in the context on the left and in the instantiation on the right. Moreover, since R is arbitrary, the derived LTS is infinitely branching.

$$\begin{tikzcd}
 & a_6 & \\
a_4 \arrow[ur, "c"] & & a_5 \arrow[ul, "d"'] \\
 & a_1 \arrow[ul, bend left, "x;h"] \arrow[ur, bend right, "y;l"'] &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & a_6 & \\
a_2 \arrow[ur, bend left, "h;c"] & & a_3 \arrow[ul, bend right, "l;d"'] \\
 & a_1 \arrow[ul, "x"] \arrow[ur, "y"'] &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & a_6 & \\
a_4 \arrow[ur, "c"] \arrow[r, "\beta", "c'"'] & a_8 \arrow[u, "e"] & a_5 \arrow[ul, "d"'] \arrow[l, "\gamma"', "d'"] \\
a_2 \arrow[u, "h"] \arrow[rr, phantom, "\Rightarrow \psi"] & a_7 \arrow[l, "x'"', "\delta"] \arrow[r, "y'"] & a_3 \arrow[u, "l"'] \\
 & a_1 \arrow[ul, "x"] \arrow[u, "z", "\varphi"'] \arrow[ur, "y"'] &
\end{tikzcd}$$

(c)

Fig. 4. G-lux

$$\begin{tikzcd}
 & 1 & \\
1 \arrow[ur, "1|m[R]"] & & 1 \arrow[ul, "1"'] \\
0 \arrow[u, "n[in\, m.1|2]"] \arrow[rr, phantom, "\Longrightarrow"] & & 3 \arrow[u, "n[in\, m.1|2]|m[3]"'] \\
 & 2 \arrow[ul, "\langle 1{,}2\rangle"] \arrow[ur, "\langle 1{,}2{,}R\rangle"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & 1 & \\
1 \arrow[ur, "1|n[in\, m.P|Q]|m[R]"] & & 1 \arrow[ul, "n[in\, m.S|T]|1"'] \\
2 \arrow[u, "n[in\, m.1|2]"] \arrow[rr, phantom, "\Longrightarrow"] & & 3 \arrow[u, "n[in\, m.1|2]|m[3]"'] \\
 & 0 \arrow[ul, "\langle S{,}T\rangle"] \arrow[ur, "\langle P{,}Q{,}R\rangle"'] &
\end{tikzcd}$$

Fig. 5. G-luxes transitions originating from $n[in\, m.1 \mid 2]$

If we apply our theory to derive the labelled transitions for the same process $n[in\, m.1 \mid 2]$, it is easy to verify that no labelled transition originates from it. For every splitting of the process and the right-hand side of the reduction rule, it is indeed never possible to obtain a GIPO-GIPB diagram.

Another example of G-lux in the category PA2CP is depicted on the right of Fig. 5. It represents the labelled transition $n[in\, m.1 \mid 2] \xrightarrow[\langle S,T\rangle]{1|n[in\, m.P|Q]|m[R]} n[in\, m.S \mid T] \mid m[n[P \mid Q] \mid R]$, where the context offers all the components that are needed to the reduction. Moreover, there is once more redundancy: processes S and T appear both in the instantiation on the left and in the context on the right, and processes P, Q and R appear both in the instantiation on the right and in the context on the left. Instead, by applying our theory we cannot derive this transition, and indeed, it is not an interesting transition: it is not relevant for the bisimilarity, since it is possible to derive it for any term.

From the examples above, it seems that our theory performs better at obtaining LTSs with less redundancy. This can be stated formally under certain conditions, as shown by the theorem below.

Definition 8. *Let* $\mathbf{C}$ *be a G-category,* $f_1, f_2 : a_1 \to a_2$ *two morphisms of* $\mathbf{C}$ *and* $\alpha : f_1 \Rightarrow f_2$ *a 2-cell. We say that it is a right-factor if for all* $g_1, g_2 : a_0 \to a_1$ *and* $\beta : g_1; f_1 \Rightarrow g_2; f_2$ *there exists* $\gamma : g_1 \Rightarrow g_2$ *such that* $\gamma * \alpha = \beta$*.*

Let $f : a_1 \to a_2$ *be a morphism of* $\mathbf{C}$*. We say that it is a 2-monomorphism if for all* $g_1, g_2 : a_0 \to a_1$ *and* $\gamma, \gamma' : g_1 \Rightarrow g_2$*,* $\gamma * 1_f = \gamma' * 1_f$ *implies* $\gamma = \gamma'$*.*

The dual notions of right-factor and 2-mono are respectively the one of left-factor and 2-epi, respectively, with 2-iso defined as obvious.

Theorem 1. *Let* $\mathbb{C}$ *be an open GRS such that its underlying category has G-luxes and all its arrows are 2-iso. If* $h \xrightarrow[x]{c} h'$ *belongs to* GGLTS$(\mathbb{C})$ *(Fig. 2 (b)),* α_1 *is a right- and* α_2 *a left-factor, then* $h \xrightarrow[x]{c} h'$ *also belongs to* LLTS$(\mathbb{C})$*.*

5 From Open G-Reactive Systems to Tile Systems

In this section we show how an open GRS can be used to generate a 2-category: this fact is going to be used to associate to a GRS a tile system, hence to obtain a finitary presentation of the derived LTS via a set of inference rules.

From Open GRSs to 2-Categories. In order to derive a 2-category from an open GRS, we adapt the construction used in [3] for RSs with closed rules.

Definition 9 (2-category of interactions). *Let $\mathbb{C}$ be an open GRS $\langle \mathbf{C}, \mathbf{D}, \mathfrak{R} \rangle$. Then, $\mathbf{C}_i$ denotes the 2-category freely generated from $\langle \mathbf{C}, \mathfrak{R} \rangle$.*

The 2-cells in $\mathbf{C}_i$ are freely generated from the G-category $\mathbf{C}$ and the reduction rules $\mathfrak{R}$, by adding the identity cells and closing under vertical and horizontal composition, subject to the exchange law. A 2-cell of $\mathbf{C}_i$ may not denote a meaningful computation in $\mathbb{C}$, since also reductions inside non-reactive contexts are allowed. Differently from [3], it indeed seems impossible to identify a suitable sub-2-category of $\mathbf{C}_i$ precisely characterizing the arrows of the open LTS: the problem is that reductions can occur only inside open terms representing reactive contexts, while any open term can represent a possible instantiation. We therefore need a way to distinguish the two types of arrows, and to this end, it is necessary to resort to *double* categories.

From 2-Categories to Double Categories. We now recall a construction for deriving a double category of squares from a 2-category [13]. As shown later, it suggests an automatic generation of a labelled relation (abstracted by a double category) starting from an unlabelled one (abstracted by a 2-category).

In the following, we fix a chosen 2-category $\mathbf{C}$.

Definition 10 (C squares). *Let a, a', b, b' objects and $h : a \to a'$, $g : b \to b'$ 1-cells of $\mathbf{C}$. The four sets of $\mathbf{C}$ squares, each square characterized by a 6-tuple (signature; left, right; top, bottom; inside) of data from $\mathbf{C}$, are defined below.*

Square $S_{-/-}$ from 1-cells $v_- : a \to b$ and $u_- : a' \to b'$ and 2-cell $\alpha_{-/-} : h; u_- \Rightarrow v_-; g$ (first diagram in Fig. 6). We indicate the data for $S_{-/-}$ by the 6-tuple $d(S_{-/-}) = (-/-; v_-, u_-; h, g; \alpha_{-/-})$ of data from $\mathbf{C}$. The left identity $l(S_{-/-})$ for $S_{-/-}$ is $(v_-, v_-; id_a, id_b; 1_{v_-})$, the right one is $r(S_{-/-}) = (u_-, u_-; id_{a'}, id_{b'}; 1_{u_-})$, the top one is $t(S_{-/-}) = (id_a, id_{a'}; h, h; 1_h)$ and the bottom one is $b(S_{-/-}) = (id_b, id_{b'}; g, g; 1_g)$.

Square $S_{+/+}$ from 1-cells $v_+ : a \leftarrow b$ and $u_+ : a' \leftarrow b'$ and 2-cell $\alpha_{+/+} : v_+; h \Rightarrow g; u_+$ (second diagram in Fig. 6), with data $(+/+; v_+, u_+; h, g, \alpha_{+/+})$ and $l(S_{+/+}) = (v_+, v_+; id_a, id_b; 1_{v_+})$, $r(S_{+/+}) = (u_+, u_+; id_{a'}, id_{b'}; 1_{u_+})$, $t(S_{+/+}) = (id_a, id_{a'}; h, h; 1_h)$, and $b(S_{+/+}) = (id_b, id_{b'}; g, g; 1_g)$.

Square $S_{+/-}$ from 1-cells $v_+ : a \leftarrow b$ and $u_- : a' \to b'$ and 2-cell $\alpha_{+/-} : v_+; h; u_- \Rightarrow g$ (third diagram in Fig. 6), with data $(+/-; v_+, u_-; h, g, \alpha_{+/-})$ and $l(S_{+/-}) = (v_+, v_+; id_a, id_b; 1_{v_+})$, $r(S_{+/-}) = (u_-, u_-; id_{a'}, id_{b'}; 1_{u_-})$, $t(S_{+/-}) = (id_a, id_{a'}; h, h; 1_h)$, and $b(S_{+/-}) = (id_b, id_{b'}; g, g; 1_g)$.

Square $S_{-/+}$ from 1-cells $v_- : a \to b$ and $u_+ : a' \leftarrow b'$ and 2-cell $\alpha_{-/+} : v_-; g; u_+ \Leftarrow h$ (last diagram in Fig. 6), with data $(-/+; v_-, u_+; h, g; \alpha_{-/+})$ and $l(S_{-/+}) = (v_-, v_-; id_a, id_b; 1_{v_-})$, $r(S_{-/+}) = (u_+, u_+; id_{a'}, id_{b'}; 1_{u_+})$, $t(S_{-/+}) = (id_a, id_{a'}; h, h; 1_h)$, and $b(S_{-/+}) = (id_b, id_{b'}; g, g; 1_g)$.

$$\begin{tikzcd}
a \arrow[r, "h"] \arrow[d, "v_-"'] & a' \arrow[d, "u_-"] \arrow[dl, Rightarrow, "\alpha_{-/-}"] \\
b \arrow[r, "g"'] & b'
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, "h"] \arrow[dr, Rightarrow, "\alpha_{+/+}"] & a' \\
b \arrow[u, "v_+"] \arrow[r, "g"'] & b' \arrow[u, "u_+"']
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[rr, "h"] & {} \arrow[d, Rightarrow, "\alpha_{+/-}"] & a' \arrow[d, "u_-"] \\
b \arrow[u, "v_+"] \arrow[rr, "g"'] & {} & b'
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[rr, "h"] \arrow[d, "v_-"'] & {} \arrow[d, Rightarrow, "\alpha_{-/+}"] & a' \\
b \arrow[rr, "g"'] & {} & b' \arrow[u, "u_+"']
\end{tikzcd}$$

Fig. 6. **C** squares

In the following we define the vertical and the horizontal composition.

Definition 11 (Vertical Composition). *Let S and S' be two* **C** *squares with arbitrary signatures a/b and c/d, respectively, and such that $b(S)$ equals $t(S')$. This forces the equality of signatures $a = c$ and $b = d$. So, the data of S and S' have respectively the form $(a/b; v, u; h, g; \alpha)$ and $(a/b; v', u'; g, g'; \alpha')$, while the one for the composition $S \bullet S'$ are $(a/b; v'', u''; h, g'; \alpha'')$. Items v'', u'' and α'' vary with the signature. Since there are four possibilities for original signatures (both a and b can have $+$ and $-$ as possible values) there are four cases of vertical composition listed in the leftmost table of Fig. 7.*

C *squares form a category $(\mathbf{C}\,Sq)^\bullet$ under $\bullet$ composition.*

a/b	v''	u''	α''
$+/+$	$v';v$	$u';u$	$(1_{v'} * \alpha) \bullet (\alpha' * 1_u)$
$+/-$	$v';v$	$u;u'$	$(1_{v'} * \alpha * 1_{u'}) \bullet \alpha'$
$-/+$	$v;v'$	$u';u$	$\alpha \bullet (1_v * \alpha' * 1_u)$
$-/-$	$v;v'$	$u;u'$	$(\alpha * 1_{u'}) \bullet (1_v * \alpha')$

Vertical composition

a/b	α''
$(+,+,+)$	$(\alpha * 1_{h'}) \bullet (1_g * \alpha')$
$(+,+,-)$	$(\alpha * 1_{h'} * 1_{u'}) \bullet (1_g * \alpha')$
$(+,-,+)$	$(1_v * 1_h * \alpha') \bullet (\alpha * 1_{g'} * 1_{u'})$
$(+,-,-)$	$(1_v * 1_h * \alpha') \bullet (\alpha * 1_{g'})$
$(-,+,+)$	$(\alpha * 1_{h'}) \bullet (1_v * 1_g * \alpha')$
$(-,+,-)$	$(\alpha * 1_{h'} * 1_{u'}) \bullet (1_v * 1_g * \alpha')$
$(-,-,+)$	$(1_h * \alpha') \bullet (\alpha * 1_{g'} * 1_{u'})$
$(-,-,-)$	$(1_h * \alpha') \bullet (\alpha * 1_{g'})$

Horizontal Composition

Fig. 7. Instances of vertical and horizontal compositions

Note that in both tables the vertical and horizontal compositions used in the last columns are the ones of the 2-category **C**.

Definition 12 (Horizontal Composition). *Let S and S' be two* **C** *squares with arbitrary signatures a/b and c/d, respectively, and such that $r(S)$ equals $l(S')$. This forces the equality of signatures $b = c$. So, the data of S and S' have respectively the form $(a/b; v, u; h, g; \alpha)$ and $(b/d; u, u'; h', g'; \alpha')$, while the one for the composition $S * S'$ are $(a/d; v, u'; h; h', g; g'; \alpha'')$. Item α'' varies with the signature. Since there are eight possibilities for original signatures (each of a, b and d can have $+$ and $-$ as possible values) there are eight cases of horizontal composition, which are listed in the rightmost table of Fig. 7.*

C *squares form a category $(\mathbf{C}\,Sq)^*$ under $*$ composition.*

$$\begin{tikzcd}
a \arrow[r, "h"] \arrow[dr, Rightarrow, "\alpha"] & a' \\
b \arrow[u, "v"] \arrow[r, "g"] \arrow[dr, Rightarrow, "\alpha'"] & b' \arrow[u, "u"'] \\
c \arrow[u, "v'"] \arrow[r, "g'"'] & c' \arrow[u, "u'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, "h"] & a' \\
b \arrow[u, "v"] & \\
c \arrow[u, "v'"] &
\end{tikzcd}
\xRightarrow{1_{v'}*\alpha}
\begin{tikzcd}
 & a' \\
b \arrow[r, "g"'] & b' \arrow[u, "u"'] \\
c \arrow[u, "v'"] &
\end{tikzcd}
\xRightarrow{\alpha'*1_u}
\begin{tikzcd}
 & a' \\
 & b' \arrow[u, "u"'] \\
c \arrow[r, "g'"'] & c' \arrow[u, "u'"']
\end{tikzcd}$$

$$\begin{tikzcd}
a \arrow[r, "h"] \arrow[dr, Rightarrow, "\alpha''"] & a' \\
c \arrow[u, "v';v"] \arrow[r, "g'"'] & c' \arrow[u, "u';u"']
\end{tikzcd}
\qquad
v';v;h \xRightarrow{1_{v'}*\alpha} v';g;u \xRightarrow{\alpha'*1_u} g';u';u$$

$$\begin{tikzcd}
a \arrow[r, "h"] \arrow[dr, Rightarrow, "\alpha"] & a' \arrow[r, "h'"] \arrow[dr, Rightarrow, "\alpha'"] & a'' \\
b \arrow[u, "v"] \arrow[r, "g"'] & b' \arrow[u, "u"'] \arrow[r, "g'"'] & b'' \arrow[u, "u'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, "h;h'"] \arrow[dr, Rightarrow, "\alpha''"] & a'' \\
b \arrow[u, "v"] \arrow[r, "g;g'"'] & b'' \arrow[u, "u'"']
\end{tikzcd}$$

$$\begin{tikzcd}
a \arrow[r, "h"] & a' \arrow[r, "h'"] & a'' \\
b \arrow[u, "v"] & &
\end{tikzcd}
\xRightarrow{\alpha*1_{h'}}
\begin{tikzcd}
 & a' \arrow[r, "h'"] & a'' \\
b \arrow[r, "g"'] & b' \arrow[u, "u"'] &
\end{tikzcd}
\xRightarrow{1_g*\alpha'}
\begin{tikzcd}
 & & a'' \\
b \arrow[r, "g"'] & b' \arrow[r, "g'"'] & b'' \arrow[u, "u'"']
\end{tikzcd}$$

$$v;h;h' \xRightarrow{\alpha*1_{h'}} g;u;h' \xRightarrow{1_g*\alpha'} g;g';u'$$

Fig. 8. Vertical and horizontal composition of two $+/+$ **C** squares

The upper part of Fig. 8 depicts the first case of vertical composition of the leftmost table of Fig. 7. The leftmost diagram on the top shows the two squares to be composed, the leftmost diagram on the bottom depicts the resulting square, and the remaining part shows how the 2-cell α'' is obtained.

The lower part of Fig. 8 depicts an example of horizontal composition. Again, we consider two **C** squares with signature $+/+$. So, the leftmost diagram on the top shows the two squares to be composed, the diagram on the right depicts the resulting square, and the remaining part shows how the 2-cell α'' is obtained.

Proposition 1. *Squares and compositions above form a double category* $\mathbf{C}\,Sq$*, for* $(\mathbf{C}\,Sq)^*$ *and* $(\mathbf{C}\,Sq)^\bullet$ *the horizontal and vertical category, respectively.*

From GRSs to double categories. Some cells of $\mathbf{C}\,Sq$ do not represent labelled transitions derived by the GIPO-GIPB mechanism. We thus introduce a way to cut them, precisely characterising the LTS synthesised from an open GRS.

Definition 13 (Observational double category). *Let* $\mathbb{C} = \langle \mathbf{C}, \mathbf{D}, \mathfrak{R} \rangle$ *be an open GRS. The* observational double category *of* $\mathbb{C}$*, denoted* $\mathcal{O}(\mathbb{C})$*, is the smallest sub-double category of* $\mathbf{C}_i$ *square* ($\mathbf{C}_i\,Sq$) *which includes the cells in Fig. 9 where the cells of type (i) correspond to GIPBs in* $\mathbf{C}$*, those of type (ii) to the rules of* $\mathfrak{R}$*, with* $l_2; l_1 = l$ *for* l *the left-hand side of a rule in* $\mathfrak{R}$*, and those of type (iii) to GIPOs in* $\mathbf{C}$*, with* $g \in \mathbf{D}$*.*

The squares of $\mathcal{O}(\mathbb{C})$ wth signature $+/-$ correspond to the derivations of GGLTS($\mathbb{C}$), as stated by the theorem below.

Theorem 2. *Let* $\mathbb{C}$ *be an open GRS. The* GGLTS($\mathbb{C}$) *has a transition* $h \xrightarrow[x]{c} h'$ *if and only if in* $\mathcal{O}(\mathbb{C})$ *there is a* $\mathbf{C}$ *square with data* $(+/-; x, c; h, h'; \alpha)$*.*

$$\begin{tikzcd}
a \arrow[r, "h"] \arrow[dr, Rightarrow, "\alpha"] & a' \\
b \arrow[u, "v"] \arrow[r, "g"'] & b' \arrow[u, "u"']
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, "id_a"] \arrow[d, phantom, "\Downarrow \alpha"] & a \arrow[d, "l_1"] \\
b \arrow[u, "l_2"] \arrow[r, "r"'] & b'
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, "h"] \arrow[d, "v"'] & a' \arrow[d, "u"] \arrow[dl, Rightarrow, "\alpha"] \\
b \arrow[r, "g"'] & b'
\end{tikzcd}$$

(i) (ii) (iii)

Fig. 9. The basic cells generating the observational double category of $\mathbb{C}$

Example 4. Consider the open GRS $\mathbb{C}_{SMA} = \langle \mathbf{C}, \mathbf{D}, \mathfrak{R} \rangle$ introduced in Example 2. Its observational double category $\mathcal{O}(\mathbb{C}_{SMA})$ is the smallest sub-double category of the double category $\mathbf{C}_i\, Sq$ which includes the cells of type (i), (ii) and (iii), as stated in the previous definition.

In particular, it contains the cells corresponding to the only rule of $\mathfrak{R}$, some of which are shown in Fig. 10. Among the cells of type (i) and (iii) there are the ones in the upper row of Fig. 11, since they are respectively two GIPBs and two GIPOs, and moreover the context offering just a hole 1 is reactive. Therefore, it is easy to see that from the composition of the above cells we obtain the cells in the upper row of Fig. 12, corresponding to the labelled transitions in Fig. 3.

Also the lower cell represents a labelled transition. It can be obtained from the one above it by composing this with the leftmost and rightmost squares of the lower transition, which are respectively a GIPB and a GIPO. It is easy to see that it represents the transition $\langle n[1 \mid Q] \mid S, 3 \mid R \rangle \xrightarrow[\langle in\, m.1,3 \rangle]{1|m[3]} m[n[1 \mid Q] \mid 3 \mid R] \mid S$. Indeed, the cell in the middle of the diagram is of type (ii), and since the two rightmost cells are GIPOs, it is obvious that also the cell obtained composing them is so; similarly, the cell obtained composing the two leftmost squares is a GIPB, since the two squares are so.

Remark 2. Let us now consider a closed GRS. Then, for any cell of type (ii) the object b always coincide with the chosen object 0. In order to allow horizontal composition, also for any cell of type (i) both b and b' should always coincide with 0 and, should we consider only closed terms, also a would be 0. In the generation of the observation double category it would then suffice to consider as cells of type (i) only those $\mathbf{C}$ squares with data $(+/+; id_0, l_2; l_2, id_0; \alpha)$, for l_2 an epimorphim. We thus recover the observational double category for ground rules and terms proposed in [3, Definition 11].

$$\begin{tikzcd}
1 \arrow[r, "1"] & 1 \arrow[d, "1"] \\
3 \arrow[u, "{n[in\, m.1|2]|m[3]}"] \arrow[r, "{m[n[1|2]|3]}"'] & 1
\end{tikzcd}
\qquad
\begin{tikzcd}
2 \arrow[r, "{\langle 1,3 \rangle}"] & 2 \arrow[d, "{1|m[3]}"] \\
3 \arrow[u, "{\langle n[in\, m.1|2],3 \rangle}"] \arrow[r, "{m[n[1|2]|3]}"'] & 1
\end{tikzcd}
\qquad
\begin{tikzcd}
3 \arrow[r, "{\langle 1,2,3 \rangle}"] & 3 \arrow[d, "{n[1|2]|m[3]}"] \\
3 \arrow[u, "{\langle in\, m.1,2,3 \rangle}"] \arrow[r, "{m[n[1|2]|3]}"'] & 1
\end{tikzcd}$$

Fig. 10. Some cells corresponding to the rule $\langle n[in\, m.1 \mid 2] \mid m[3], m[n[1 \mid 2] \mid 3] \rangle$

$$\begin{tikzcd}
2 \arrow[r, "\langle 1,Q,3\rangle"] & 3 & 2 \arrow[r, "\langle n[in\, m.1|Q],3\rangle"] & 2 & 3 \arrow[r, "\langle n[1|2],3\rangle"] \arrow[d, "n[1|2]|m[3]"] & 2 \arrow[d, "1|m[3]"] & 2 \arrow[r, "\langle 1,3\rangle"] \arrow[d, "1|m[3]"] & 2 \arrow[d, "1|m[3]"] \\
2 \arrow[r, "\langle 1,Q,3\rangle"'] \arrow[u, "\langle in\, m.1,3\rangle"] & 3 \arrow[u, "\langle in\, m.1,2,3\rangle"] & 2 \arrow[r, "\langle 1,Q,3\rangle"'] \arrow[u, "\langle 1,3\rangle"] & 3 \arrow[u, "\langle n[in\, m.1|2],3\rangle"] & 1 \arrow[r, "1"'] & 1 & 1 \arrow[r, "1"'] & 1
\end{tikzcd}$$

Fig. 11. Cells of $\mathcal{O}(\mathbb{C}_{SMA})$ of type (i) and (iii)

$$\begin{tikzcd}
2 \arrow[r, "\langle n[in\ m.1|Q],3\rangle"] & 2 \arrow[r, "\langle 1,3\rangle"] & 2 \arrow[r, "\langle 1,3\rangle"] \arrow[d, "1|m[3]"] & 2 \arrow[d, "1|m[3]"] \\
2 \arrow[r, "\langle 1,Q,3\rangle"'] \arrow[u, "\langle 1,3\rangle"] & 3 \arrow[r, "m[n[1|2]|3]"'] \arrow[u, "\langle n[in\, m.1|2],3\rangle"] & 1 \arrow[r, "1"'] & 1
\end{tikzcd}
\qquad
\begin{tikzcd}
2 \arrow[r, "\langle 1,Q,3\rangle"] & 3 \arrow[r, "\langle 1,2,3\rangle"] & 3 \arrow[r, "\langle n[1|2],3\rangle"] \arrow[d, "n[1|2]|m[3]"] & 2 \arrow[d, "1|m[3]"] \\
2 \arrow[r, "\langle 1,Q,3\rangle"'] \arrow[u, "\langle in\, m.1,3\rangle"] & 3 \arrow[r, "m[n[1|2]|3]"'] \arrow[u, "\langle in\, m.1,2,3\rangle"] & 1 \arrow[r, "1"'] & 1
\end{tikzcd}$$

$$\begin{tikzcd}
2 \arrow[r, "\langle 1,3|R\rangle"] & 2 \arrow[r, "\langle 1,Q,3\rangle"] & 3 \arrow[r, "\langle 1,2,3\rangle"] & 3 \arrow[r, "\langle n[1|2],3\rangle"] \arrow[d, "n[1|2]|m[3]"] & 2 \arrow[r, "\langle 1|S,3\rangle"] \arrow[d, "1|m[3]"] & 2 \arrow[d, "1|m[3]"'] \\
3 \arrow[r, "\langle 1,3|R\rangle"'] \arrow[u, "\langle in\, m.1,3\rangle"] & 2 \arrow[r, "\langle 1,Q,3\rangle"'] \arrow[u, "\langle in\, m.1,3\rangle"] & 3 \arrow[r, "m[n[1|2]|3]"'] \arrow[u, "\langle in\, m.1,2,3\rangle"] & 1 \arrow[r, "1"'] & 1 \arrow[r, "1|S"'] & 1
\end{tikzcd}$$

Fig. 12. Cells of $\mathcal{O}(\mathbb{C}_{SMA})$ representing labelled transitions

6 Conclusions and Further Works

We presented a mechanism to derive LTSs for GRSs with open terms and parametric rules. With respect to the only alternative proposal we are aware of [8], our approach almost straightforwardly exploits the well-known categorical notions of GIPO and GIPB to respectively capture the notions of minimal context and most general instantiation that allow for a reduction to occur.

We illustrated the main concepts by using a small running example, a simplified version of Mobile Ambients. The use of SMA allowed us to better highlight the role of instantiations and contexts in labelled transitions for open terms. It also gave the chance of performing an informal comparison between our approach and the one of [8]. This is made precise by observing that our approach can be used for a wider range of case studies and that it seems to derive in general a more concise LTS (even if this has been formally proved so far only under quite stringent conditions). However, we leave as future work the comparison of the results obtained by applying the two techniques to real-life formalisms, as well as the comparison between our technique and the more complex synthesis mechanism based on irredundant G-luxes (also championed in [8]).

As in [3], our paper draws a connection with the tile model [6]: the derivation from an open GRS $\mathbb{C}$ of a (freely constructed) double category is illustrated, exploiting a functorial construction proposed in [13]. It results in a finitary, SOS-like presentation of the LTS derived from that same $\mathbb{C}$ via a suitable double category. Preliminary obervations suggest that the LTS synthesised via G-luxes cannot be offered the same modular presentation: the simultaneous requirement on contexts and instantiations seems to require an infinite set of basic tiles in the observational double category, in order to model G-lux transitions via cells.

Despite its features, our framework suffers from the same problem as the synthesis mechanism based on irredundant G-luxes: the bisimilarity over the derived LTS is not a congruence. Consider e.g. the SMA process $n[in\, m.1 \mid 2]$. As said in Section 4, no labelled transition starting from it can be derived. This implies that it is equivalent to $n[in\, o.1 \mid 2]$, for any channel name o. Clearly, this equivalence is not preserved by all contexts. We leave as future work the study of this problem. It could lead towards a further refinement of our synthesis mechanism for LTSs, possibly by acting directly on the generating cells of the observational double category. Otherwise, one could consider an alternative definition of bisimilarity: as it has been carried out for the closed GRSs approach [2], one could study barbed saturated bisimilarity for open terms, by giving a labelled characterization of it through the GGLTS.

References

1. Bonchi, F., Gadducci, F., Monreale, G.V.: Labelled Transitions for Mobile Ambients (As Synthesized Via a Graphical Encoding). In: EXPRESS 2008. ENTCS, vol. 242(1), pp. 73–98. Elsevier, Amsterdam (2009)
2. Bonchi, F., Gadducci, F., Monreale, G.V.: Reactive Systems, Barbed Semantics, and the Mobile Ambients. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 272–287. Springer, Heidelberg (2009)
3. Bruni, R., Gadducci, F., Montanari, U., Sobociński, P.: Deriving Weak Bisimulation Congruences from Reduction Systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 293–307. Springer, Heidelberg (2005)
4. Bruni, R., Meseguer, J., Montanari, U.: Symmetric Monoidal and Cartesian Double Categories as a Semantics Framework for tile Logic. MSCS 12(1), 53–90 (2002)
5. Cardelli, L., Gordon, A.: Mobile Ambients. TCS 240(1), 177–213 (2000)
6. Gadducci, F., Montanari, U.: The Tile Model. In: Proof, Language and Interaction: Essays in Honour of Robin Milner, pp. 133–166. MIT Press, Cambridge (2000)
7. Kelly, G.M., Street, R.: Review of the Elements of 2-Categories. In: Sydney Category Seminar. LNM, vol. 420, pp. 75–103. Springer, Heidelberg (1974)
8. Klin, B., Sassone, V., Sobociński, P.: Labels from Reductions: Towards a General Theory. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 30–50. Springer, Heidelberg (2005)
9. Leifer, J.J., Milner, R.: Deriving Bisimulation Congruences for Reactive Systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
10. MacLane, S.: Categorical Algebra. Bull. Amer. Math. Soc. 71, 40–106 (1965)
11. Merro, M., Zappa Nardelli, F.: Behavioral Theory for Mobile Ambients. Journal of the ACM 52(6), 961–1023 (2005)
12. Meseguer, J.: Conditional Rewriting Logic as a Unified Model of Concurrency. TCS 96(1), 73–155 (1992)
13. Palmquist, P.H.: The Double Category of Adjoint Squares. In: Midwest Category Seminar. LNM, vol. 195, pp. 123–153. Springer, Heidelberg (1971)
14. Power, A.J.: An Abstract Formulation for Rewrite Systems. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) CTCS 1989. LNCS, vol. 389, pp. 300–312. Springer, Heidelberg (1989)
15. Rathke, J., Sobociński, P.: Deriving Structural Labelled Transitions for Mobile Ambients. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 462–476. Springer, Heidelberg (2008)
16. Sassone, V., Sobocinski, P.: Deriving Bisimulation Congruences Using 2-Categories. Nordic Journal of Computing 10(2), 163–183 (2003)

Unidirectional Channel Systems Can Be Tested

Petr Jančar[1,⋆], Prateek Karandikar[2,⋆⋆], and Philippe Schnoebelen[3,⋆⋆⋆]

[1] Techn. Univ. Ostrava
[2] Chennai Mathematical Institute
[3] LSV, ENS Cachan, CNRS

Abstract. "Unidirectional channel systems" (Chambart & Schnoebelen, CONCUR 2008) are systems where one-way communication from a sender to a receiver goes via one reliable and one unreliable (unbounded fifo) channel. Equipping these systems with the possibility of testing regular properties on the contents of channels makes verification undecidable. Decidability is preserved when only emptiness and nonemptiness tests are considered: the proof relies on a series of reductions eventually allowing us to take advantage of recent results on Post's Embedding Problem.

1 Introduction

Channel systems are a family of computational models where several, usually finite-state, agents communicate via usually unbounded fifo communication channels [1]. These models are well-suited to the formal specification and algorithmic analysis of asynchronous communication protocols [2–5]. They are sometimes called *queue automata* when there is only one agent using the channels as fifo memory buffers.

A particularly interesting class of channel systems are the *lossy channel systems*, "LCS" for short, popularized by Abdulla, Bouajjani, Jonsson, Finkel, *et al.* [6–8]. Lossy channels are unreliable and can lose messages nondeterministically and without any notification. A bit surprisingly, this makes lossy systems easier to analyse: safety, inevitability and several more properties are decidable for this model [6, 7, 9–11] while they are undecidable when channels are reliable.

It should be stressed that LCS's have also been very useful outside the field of communicating systems and distributed computing. During the last decade, they have been used to show the decidability, or (more often) the hardness, of problems on Timed Automata, Metric Temporal Logic, modal logics, etc. [12–16]. With other unreliable computational models, lossy channel systems are now an important tool for the complexity analysis of algorithms that rely on well-quasi-ordering theory [17–19].

Unidirectional channel systems, "UCS" for short, are a variant of LCS's where a Sender process communicates to a Receiver process via one reliable and one lossy channel. Fig. 1 gives an example.

⋆ Supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), and by the project GAČR:P202/11/0340.
⋆⋆ Partially funded by Tata Consultancy Services.
⋆⋆⋆ Currently visiting Oxford Univ. Comp. Sci. Dept, supported by Grant ANR-11-BS02-001 and by the Leverhulme Trust.

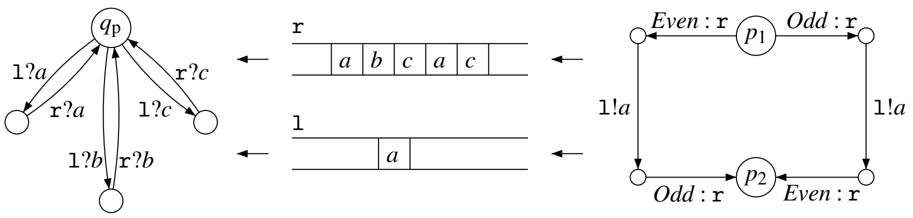


Fig. 1. Unidirectional channels: Sender on the right, Receiver on the left

The presence of one reliable channel put UCS's beyond plain LCS's. On the other hand, the unidirectionality (there is no channel from Receiver to Sender) is a limitation that LCS's do not share.

UCS's were first studied by Chambart and Schnoebelen who considered *mixed* channel systems (i.e., communicating systems using both reliable and lossy channels in arbitrary combinations) and showed how to reduce safety and reachability problems for arbitrary network topologies to reachability problems on either queue automata (undecidable), or LCS's (decidable), or the previously unidentified UCS's [20].

The reachability problem for UCS's is quite challenging: it was proved decidable by reformulating it more abstractly as PEP, aka the *Regular Post Embedding Problem*, which is easier to analyze [21–23]. We want to stress that, while PEP is a natural variant of Post's Correspondence Problem, it was only identified through questions on UCS's. Recently, PEP has proved useful in other areas, starting with Graph Logics [24].

Testing channel contents. Basic channel machines are not allowed to inspect the contents of the channels. However, it is natural to enrich the basic setup with tests (conditions on channel contents) as in, e.g., Fig. 1 where some sender's actions depend on (the parity of) the number of messages currently in r. Adding tests goes smoothly and painlessly for LCS's where the main decidability results extend directly with almost unchanged algorithms [10, sect. 3.3]. Adding even simple tests to UCS's is a completely different story, as we discovered. One meets two obstacles when trying to extend the approach that worked for UCS's:

1. The "reformulation" of UCS reachability as a Post Embedding Problem is a nontrivial reduction that *reorders* the events in a UCS run, relying on the independence (in a concurrency-theoretical sense) of sendings wrt readings. Tests on channel contents introduce global dependencies that are not reflected in PEP problems.
2. One is then led to consider *extensions* of PEP where said dependencies can be reflected, raising a new question: how to show the decidability of these extensions?

Our contribution. We extend UCS's with the possibility of testing channel contents with simple regular predicates. This makes reachability undecidable even with restricted sets of simple tests. Our main result is that reachability remains decidable when only emptiness and non-emptiness tests are allowed. The proof goes through a series of reductions that leave us with UCS's extended by only emptiness tests on a single side of a single channel (called "Z_1^1 tests"). This minimal extension can then be reformulated as

$\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, or "PEP with partial codirectness", a nontrivial extension of PEP that was recently proved decidable [25].

Outline of the paper. Unidirectional channel systems with tests are defined in Section 2. Section 3 shows how undecidability creeps in when regular tests are allowed. Section 4 presents protocols for simulating (non-)emptiness in UCST's with only emptiness tests by Sender, thus reducing UCST to UCST[Z_1]. Section 5 proves decidability for UCST[Z_1^1] by reducing to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$. This is then leveraged in section 6 to account for the whole of UCST[Z_1]. Finally, Section 7 proves that (non-)emptiness tests strictly enrich the basic UCS model.

2 Unidirectional Channel Systems

Structure. Formally, a *UCST* (for *Unidirectional Channel System with Tests*) is a tuple $S = (\mathtt{Ch}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, where $\mathtt{M}$ is a finite alphabet of *messages*, Q_1, Q_2 are disjoint finite sets of *states* of Sender and Receiver, respectively, and Δ_1, Δ_2 are finite sets of *rules* of Sender and Receiver, respectively. $\mathtt{Ch} = \{\mathtt{r}, \mathtt{l}\}$ is a fixed set of channel (names), $\mathtt{r}$ being reliable and $\mathtt{l}$ unreliable, so called "lossy".

A rule $\delta \in \Delta_i$ is a tuple $(q, \mathtt{c}, \alpha, q') \in Q_i \times \mathtt{Ch} \times Act \times Q_i$ where the set of actions Act contains *tests* R (checking whether the contents of $\mathtt{c} \in \mathtt{Ch}$ belongs to R, a regular language) and *communications* w (sending a sequence of messages to $\mathtt{c}$ in the case of Sender's actions, reading it for Receiver's) and is thus given by $Act \overset{\text{def}}{=} \mathsf{Reg}(\mathtt{M}) \cup \mathtt{M}^*$.

We write $q \xrightarrow{R:\mathtt{c}} q'$ for a rule where the action is a test on $\mathtt{c}$, and $q \xrightarrow{\mathtt{c}!w} q'$ (resp., $q \xrightarrow{\mathtt{c}?w} q'$) when the action is a communication by Sender (resp., by Receiver).

In graphical representations like Fig. 1, Sender and Receiver are depicted as two disjoint directed graphs, where states appear as nodes and where rules $q \xrightarrow{\alpha} q'$ appear as edges from q to q' with action and channel name labeling the edge. We may omit the label, or just use $\top$, for trivial tests, $R = \mathtt{M}^*$, or empty communications, $w = \varepsilon$.

Remark 2.1 (On separating tests from communications). Our definition requires that an action is a test *or* a communication. It does not allow performing both atomically inside a single step (but they can be chained using intermediary states). This choice, which is no real loss of generality, lets us focus on simulating tests by other constructs (or other tests) without having to account for accompanying communications. □

Operational Semantics. The behaviour of S is defined via an operational semantics defined along standard lines. A *configuration* of $S = (\mathtt{Ch}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$ is a tuple $C \in Conf_S \overset{\text{def}}{=} Q_1 \times Q_2 \times \mathtt{M}^* \times \mathtt{M}^*$. In $C = (q_1, q_2, u, v)$, q_1 and q_2 are the current states of, respectively, Sender and Receiver, while u and v are the current contents of, respectively, $\mathtt{r}$ and $\mathtt{l}$.

Rules give rise to transitions in the expected way. We start with so-called "reliable" steps where the effect of a rule is deterministic. Formally, given two configurations $C = (q_1, q_2, u, v)$, $C' = (q'_1, q'_2, u', v')$ and a rule $\delta = (q, \mathtt{c}, \alpha, q')$, there is a reliable step denoted $C \xrightarrow{\delta}_{\text{rel}} C'$ if, and only if, the following four conditions are satisfied:

states: $q = q_1$ and $q' = q'_1$ and $q_2 = q'_2$ (for Sender rules), or $q = q_2$ and $q' = q'_2$ and $q_1 = q'_1$ (for Receiver rules);

tests: if δ is a test rule $q \xrightarrow{R:\mathtt{c}} q'$, then $\mathtt{c} = \mathtt{r}$ and $u \in R$, or $\mathtt{c} = \mathtt{l}$ and $v \in R$, and furthermore $u' = u$ and $v' = v$;

writes: if δ is a writing rule $q \xrightarrow{\mathtt{c}!w} q'$, then $\mathtt{c} = \mathtt{r}$ and $u' = uw$ and $v' = v$, or $\mathtt{c} = \mathtt{l}$ and $u' = u$ and $v' = vw$;

reads: if δ is a reading rule $q \xrightarrow{\mathtt{c}?w} q'$, then $\mathtt{c} = \mathtt{r}$ and $u = wu'$ and $v' = v$, or $\mathtt{c} = \mathtt{l}$ and $u' = u$ and $v = wv'$.

Now to unreliable, aka lossy, steps denoted $C \xrightarrow{\delta}_{\text{los}} C'$. As is standard, a lossy step is defined as a combination of message losses (where the contents of $\mathtt{l}$ may be replaced with a subword) with a reliable step. For $v_1, v_2 \in \mathtt{M}^*$, we write $v_1 \sqsubseteq v_2$ when v_1 is a subword of v_2, i.e., a (scattered) subsequence. In particular, $\varepsilon \sqsubseteq v_2$ and $v_2 \sqsubseteq v_2$ for any v_2. This is extended to configurations and we write $C \sqsubseteq D$ when $C = (q_1, q_2, u, v)$ and $D = (q_1, q_2, u, v')$ with $v \sqsubseteq v'$.[1] We now define:

$$C \xrightarrow{\delta}_{\text{los}} C' \overset{\text{def}}{\Leftrightarrow} \exists D, D' : C \sqsupseteq D \wedge D \xrightarrow{\delta}_{\text{rel}} D' \wedge D' \sqsupseteq C' . \tag{1}$$

In other words, a lossy step is a reliable step sandwiched between arbitrary message losses on $\mathtt{l}$. In particular, reliable steps are a special case of lossy steps. In the rest of this paper, we consider reachability via lossy steps, and often write simply $C \xrightarrow{\delta} C'$ without a "los" subscript. (When we refer to reliable steps and runs, we always use "rel" subscript.)

Remark 2.2 (On reliable steps). As is usual with lossy channel systems, the reliable semantics plays a key role even though the object of our study is reachability via unreliable steps. First $\rightarrow_{\text{rel}}$ is a normative yardstick from which the unreliable semantics depart: $\rightarrow_{\text{los}}$ is defined as a modification of $\rightarrow_{\text{rel}}$. Then many hardness results on lossy systems are proved with reductions where a lossy system simulates in some way the reliable (and Turing-powerful) behaviour. □

A *run* from C_0 to C_n is a sequence of chained steps $C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \cdots \xrightarrow{\delta_n} C_n$, abbreviated as $C_0 \xrightarrow{*} C_n$ (or $C_0 \xrightarrow{+} C_n$ when we rule out zero-length runs).

Definition 2.3. *The* Reachability Problem *is the question, given a UCST S and some states $p_{\text{in}}, p_{\text{fi}} \in Q_1$, $q_{\text{in}}, q_{\text{fi}} \in Q_2$, whether S has a (lossy) run $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, \varepsilon, \varepsilon) \xrightarrow{*} C_{\text{fi}} = (p_{\text{fi}}, q_{\text{fi}}, \varepsilon, \varepsilon)$.*

The Extended Reachability Problem *asks, further given regular languages $U, V, U', V' \subseteq \mathtt{M}^*$, whether there exist $u \in U$, $v \in V$, $u' \in U'$, and $v' \in V'$ such that S has a (lossy) run $(p_{\text{in}}, q_{\text{in}}, u, v) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$.*

In the following we only consider reachability problems with empty channels in C_{in} and C_{fi} since this is technically convenient. There is no loss of generality:

Lemma 2.4. *The extended reachability problem many-one reduces to the reachability problem.*

[1] Note that $(\mathit{Conf}, \sqsubseteq)$ is *not* a well-quasi-order since $C \sqsubseteq D$ requires equality on channel $\mathtt{r}$.

Roughly speaking, we can transform an instance of the extended reachability problem to an "empty-channel" instance by letting Sender start with generating some $u \in U$, $v \in V$ into the channels, and by letting Receiver read some $u' \in U'$, $v' \in V'$ in the end.

The two problems are thus equivalent. Moreover, the reduction does not need to introduce any new tests, and the equivalence thus also holds for UCST's with restricted sets of tests which we will consider.

3 Testing Channels and the Undecidability of Reachability

Despite their similarities, UCS's and LCS's (lossy channel systems) behave differently. The algorithms deciding reachability for LCS's can easily accommodate regular (or even more expressive) tests [10, Sect. 3.3]. By contrast, this section gives several versions of the following result:

Theorem 3.1. *Reachability is undecidable for UCST.*

3.1 Simulating Queue Automata

We now show how even simple tests lead to undecidability. The main technique we use is to simulate queue automata which are a Turing-powerful model already with a single reliable channel.

UCS's already have a reliable channel but Sender (or Receiver) cannot both read *and* write from/to it. If Sender could somehow read from the head of `r` as well as write to its tail, it would be as powerful as a queue automaton. Now, with regular tests on channels, there exists a simple protocol making Receiver act as a proxy for Sender and implement read actions on its behalf.

Described informally, the protocol is the following[2]:

1. Channel `l` is initially empty.
2. In order to "read" from `r`, Sender checks and records whether the length of the contents of `r` is odd or even, using a regular test on `r`.
3. It then sends on `l` the message, say a, that it wants to read.
4. It checks that (equivalently, waits until) the parity of the contents of `r` has changed, and on detecting this change, concludes that the read was successful.
5. Receiver waits in its initial q_{proxy} (or q_p) state and tries to read from `l`. When it reads a message a from `l`, it understands it as an request telling it to read a from `r` on behalf of Sender. Once it has performed this read on `r`, it returns to q_{proxy} and waits for the next instruction.
6. `l` is now empty and the simulation of a read by Sender is concluded.

If no messages are lost on `l`, the protocol allows Sender to read on `r`. If a message is lost on `l`, the protocol deadlocks. Also, Sender deadlocks if it attempts to read a message that is not at the head of `r` (it has to guess correctly). We note that these deadlocks do not make the simulation incorrect since we are only concerned with reachability.

[2] We describe the protocol informally but Fig. 1 page 150 depicts exactly how Receiver implements a proxy on $\mathtt{M} = \{a,b,c\}$ and how Sender simulates a rule $p_1 \xrightarrow{\mathtt{r}?a} p_2$ for a queue automaton.

3.2 Restricted Sets of Tests

In the above reduction only parity tests were used. When $T \subseteq \mathsf{Reg}(\mathtt{M})$, we write UCST[$T$] to denote the class of UCST's where only tests belonging to T are allowed. Thus UCST and UCS coincide with UCST[$\mathsf{Reg}(\mathtt{M})$] and UCST[$\varnothing$], respectively.

More interestingly, defining $Odd, Even \in \mathsf{Reg}(\mathtt{M})$ with $Even \overset{\text{def}}{=} (\mathtt{M}.\mathtt{M})^*$ and $Odd \overset{\text{def}}{=} \mathtt{M}.Even$, and letting $P \overset{\text{def}}{=} \{Even, Odd\}$ denote the parity tests, section 3.1 shows that reachability is undecidable already for UCST[P].

We further observe that in Fig. 1 only the sender uses tests, and only $\mathtt{r}$ is submitted to tests. We denote such restricted uses of tests by qualifying test sets like T with a subscript 1 (for Sender) or 2 (for Receiver), and/or by a superscript $\mathtt{r}$ or $\mathtt{l}$. We can now state the following stronger form of Theorem 3.1:

Theorem 3.2. *Reachability is undecidable for UCST[$P_1^{\mathtt{r}}$].*

In the rest of this paper, we single out other simple test sets by letting:

$$Z \overset{\text{def}}{=} \{\varepsilon\}, \qquad N \overset{\text{def}}{=} \mathtt{M}^+, \qquad H_a \overset{\text{def}}{=} a.\mathtt{M}^*, \qquad H \overset{\text{def}}{=} \{H_x \mid x \in \mathtt{M}\}.$$

In other words, Z is the *emptiness* (or "zero") test, N is the *non-emptiness* test and H are the *head* tests (that allow checking what is the first message in a channel without consuming it). Note that non-emptiness tests can be simulated with head tests, hence are weaker. Below we abuse notation and, when $R, R' \in \mathsf{Reg}(\mathtt{M}^*)$, we write UCST[$R$] and UCST[$R, R'$] rather than UCST[$\{R\}$] and UCST[$\{R, R'\}$].

One difference with parity tests and the Z, N, H tests is that parity tests are "global" in that their outcome depends on the entire contents of a channel. With H tests, only one message at the head needs be scanned. Still, "local" H tests are sufficient for undecidability:

Theorem 3.3. *Reachability is undecidable for UCST[$H_1^{\mathtt{r}}$].*

Proof (Idea). Sender can simulate parity tests $P_1^{\mathtt{r}}$ by using two copies of the message alphabet, say using different colors. It alternates strictly between the two colors when writing on $\mathtt{r}$. This requires an extra bit of memory, encoded in local states. Then the parity of the length of $\mathtt{r}$ contents can be tested by looking at the first message, using $H_1^{\mathtt{r}}$ tests, and comparing with the color of the last written message. (This assumes that $\mathtt{r}$ is never completely emptied, otherwise deadlocks will occur, but this is no loss of generality.) □

4 Simulating UCST[Z, N] by Using Sender's Emptiness Tests Only

This section describes two simulations that, put together, entail Theorem 4.1.
Remark. The simulations are tailored to the reachability problem. They may not preserve, e.g., termination or deadlock-freedom.

Theorem 4.1. *Reachability for UCST[Z,N] many-one reduces to reachability for UCST[Z_1].*

4.1 Reducing UCST[Z,N] to UCST[Z_1,N_1]

We now explain how to eliminate Z and N tests by Receiver. W.l.o.g. we assume that x in $\mathtt{c}!x$ and $\mathtt{c}?x$ is always one symbol ($x \in \mathtt{M}$), and we use two special new messages, "$\mathtt{z}$" and "$\mathtt{n}$", with which Sender will signal to Receiver about the status, empty or not, of the channels.

Formally, for $S \in$UCST[Z,N], where $S = (\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$, we construct S' arising from S as follows (see Fig. 2):

- S' uses the special new messages $\mathtt{z},\mathtt{n}$, and it thus has alphabet $\mathtt{M}' \stackrel{\text{def}}{=} \mathtt{M} \cup \{\mathtt{n},\mathtt{z}\}$;
- for each channel $\mathtt{c} \in \{\mathtt{r},\mathtt{l}\}$ and each sender state $p \in Q_1$ we add new states $p^1_{\mathtt{c}}$, $p^2_{\mathtt{c}}$ and an "*(emptiness) testing loop*" $p \xrightarrow{Z:\mathtt{c}} p^1_{\mathtt{c}} \xrightarrow{\mathtt{c}!\mathtt{z}} p^2_{\mathtt{c}} \xrightarrow{Z:\mathtt{c}} p$;
- for every sender rule θ of the form $p \xrightarrow{\mathtt{c}!x} p'$ we add a new state p_θ, and the rule is replaced in S' by the following three rules: $p \xrightarrow{\top} p_\theta$, $p_\theta \xrightarrow{\mathtt{c}!\mathtt{n}} p_\theta$ (a "*padding loop*"), and $p_\theta \xrightarrow{\mathtt{c}!x} p'$;
- every receiver rule $q \xrightarrow{Z:\mathtt{c}} q'$ testing emptiness of $\mathtt{c}$ is replaced by $q \xrightarrow{\mathtt{c}?\mathtt{z}} q'$;
- every receiver rule $q \xrightarrow{N:\mathtt{c}} q'$ testing non-emptiness of $\mathtt{c}$ is replaced by $q \xrightarrow{\mathtt{c}?\mathtt{n}} q'$.

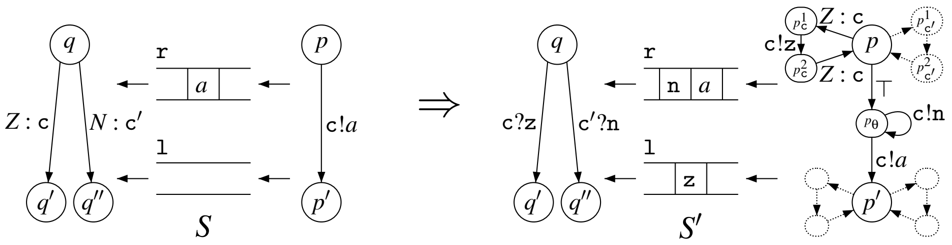


Fig. 2. From S to S': eliminating Receiver's N and Z tests

Lemma 4.2 (Correctness of the reduction). *S has a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ if, and only if, S' has a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$.*

Proof (Sketch). The "$\Rightarrow$" direction. Suppose a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ of S. For each concrete occurrence o of a message $x \in \mathtt{M}$ which is written to a channel $\mathtt{c}$ there is a number k_o such that the run uses k_o steps where Receiver tests $\mathtt{c}$ for non-emptiness (i.e., performs transitions $q \xrightarrow{N:\mathtt{c}} q'$) in the situation when o is the first symbol (the head) in $\mathtt{c}$. We can use this for constructing a run of S' which mimics the above run of S. Any original Sender's step $p \xrightarrow{\mathtt{c}!x} p'$, writing an occurrence o of x to $\mathtt{c}$, is replaced by $p \xrightarrow{\top} p_\theta \xrightarrow{\mathtt{c}!\mathtt{n}} p_\theta \xrightarrow{\mathtt{c}!\mathtt{n}} \cdots \xrightarrow{\mathtt{c}!\mathtt{n}} p_\theta \xrightarrow{\mathtt{c}!x} p'$ where the padding loop is used k_o times; any Receiver's step $q \xrightarrow{N:\mathtt{c}} q'$ is replaced by $q \xrightarrow{\mathtt{c}?\mathtt{n}} q'$. Any original Receiver's step $q \xrightarrow{Z:\mathtt{c}} q'$, when Sender is in state p, is replaced with the sequence of steps corresponding to $p \xrightarrow{Z:\mathtt{c}} p^1_{\mathtt{c}} \xrightarrow{\mathtt{c}!\mathtt{z}} p^2_{\mathtt{c}}$, $q \xrightarrow{\mathtt{c}?\mathtt{z}} q'$, $p^2_{\mathtt{c}} \xrightarrow{Z:\mathtt{c}} p$. The inserted $\mathtt{n}$'s (and $\mathtt{z}$'s) are never lost; other message losses are the same as originally.

The "$\Leftarrow$" direction. Suppose a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ of S'. It is convenient to consider the run as a sequence of *fine-grained steps*, i.e., $C_{\text{in}} \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \xrightarrow{\delta_3} \cdots C_{n-1} \xrightarrow{\delta_n} C_{\text{fi}}$, where each step is either a reliable step $C_{i-1} \xrightarrow{\delta_i}_{\text{rel}} C_i$ or the loss of a single message. The idea is to repeatedly switch two consecutive steps conveniently so that the validity of the obtained (fine-grained) runs is kept, with the aim to achieve a "*straight run*" which can be easily translated to a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ of S. Imagine first that we give the priority to Sender's (reliable) steps: whenever some (current) δ_i is a Receiver's step or a loss and δ_{i+1} is a Sender's step then we switch the steps if the result is still a valid run. It is easy to observe that the writing steps in the resulting run (in which no above switches are possible) are only in the segments corresponding to $p \xrightarrow{\top} p_\theta \xrightarrow{\mathtt{c!n}} p_\theta \xrightarrow{\mathtt{c!n}} \cdots \xrightarrow{\mathtt{c!n}} p_\theta \xrightarrow{\mathtt{c!}x} p'$ (uninterrupted by Receiver or message losses). Regarding the steps corresponding to testing loops, we get segments $p \xrightarrow{\mathtt{Z:c}} p^1_{\mathtt{c}} \xrightarrow{\mathtt{c!z}} p^2_{\mathtt{c}}$, σ, $p^2_{\mathtt{c}} \xrightarrow{\mathtt{Z:c}} p$ where σ is a sequence of Receiver's steps and/or message losses. Now we can switch (anyhow) inside such a segment, with the aim to get $\sigma_1, p \xrightarrow{\mathtt{Z:c}} p^1_{\mathtt{c}} \xrightarrow{\mathtt{c!z}} p^2_{\mathtt{c}}, \sigma_2, p^2_{\mathtt{c}} \xrightarrow{\mathtt{Z:c}} p$ for a shortest σ_2. It turns out that σ_2 is, in fact, one step, either $q \xrightarrow{\mathtt{c?z}} q'$ or a loss of $\mathtt{z}$. There is a final issue: we arrange that the finally achieved run is also "*head-lossy*", i.e. any loss-step loses the first message (the head) of $\mathtt{l}$; thus we never have only $\mathtt{n}$'s in $\mathtt{l}$ when Sender tests the non-emptiness of $\mathtt{l}$. It is then straightforward to translate the finally achieved run of S' to the corresponding run of S. □

4.2 Reducing UCST[Z_1, N_1] to UCST[Z_1]

When there are no receiver tests, N_1 tests can be eliminated by a buffering technique on Sender's side. With any $S \in$UCST[Z_1, N_1] we associate a derived system S' as follows:

For each channel $\mathtt{c} \in \mathtt{Ch}$, S' uses an auxiliary 1-place buffer between Sender and the channel $\mathtt{c}$. In any S' configuration, a buffer is empty (containing no messages) or full (containing a single message). Now the sender does not write to the channels, it can only directly write to the auxiliary buffers, and it may only write to a buffer when it is empty, making it full. Buffers may be nondeterministically flushed at any time, transferring their contents to the actual channel (in a potentially lossy way for $\mathtt{l}$). Finally, the buffers are not actual extra peripherals, rather they are encoded in the finite control of Sender, which also simulates the lossy behavior of writing to channel $\mathtt{l}$. Within this setup, an $N_1^{\mathtt{c}}$ test translates to "$\mathtt{c}$'s auxiliary buffer is full", and a $Z_1^{\mathtt{c}}$ test translates to "$\mathtt{c}$'s buffer is empty *and* $\mathtt{c}$ is empty".

Finally, $S' \in$UCST[Z_1] simulates S without any need of N_1 tests, as stated by the following lemma.

Lemma 4.3 (Correctness of the reduction). *S has a run $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ if, and only if, S' has a run $C'_{\text{in}} \xrightarrow{*}_{\text{los}} C'_{\text{fi}}$ (where C'_{in} and C'_{fi} are the configurations in S' corresponding to C_{in} and C_{fi} with empty auxiliary buffers).*

5 Reachability for UCST[Z_1^1] via Post's Embedding Problem

This section develops a many-one reduction from the reachability problem for UCST[Z_1^1] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, a generalization of Post's Embedding Problem.

Definition 5.1 (Post embedding with partial codirectness [25]). $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *is the question, given two finite alphabets* Σ,Γ*, two morphisms* $u,v:\Sigma^*\to\Gamma^*$*, and two regular languages* $R,R'\in\mathsf{Reg}(\Sigma)$*, whether there is* $\sigma\in R$ *(called a* solution*) such that* $u(\sigma)\sqsubseteq v(\sigma)$*, and such that furthermore* $u(\sigma')\sqsubseteq v(\sigma')$ *for all suffixes* σ' *of* σ *that belong to* R'*.*[3]

The above definition uses the same subword relation, "$\sqsubseteq$", that captures message losses. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ can be compared with Post's Correspondence Problem, where the question is whether there exists $\sigma\in\Sigma^+$ such that $u(\sigma)=v(\sigma)$.

Since $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is decidable [25], we deduce:

Corollary 5.2. *Reachability is decidable for UCST[Z_1^1].*

The reduction from UCST[Z_1^1] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ extends an earlier reduction from UCS to PEP [22]. Here the presence of Z_1^1 tests creates new difficulties.

We fix an instance $S=(\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$, $C_{\text{in}}=(p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon)$, $C_{\text{fi}}=(p_{\text{fi}},q_{\text{fi}},\varepsilon,\varepsilon)$ of the reachability problem for UCST[Z_1^1]. (We again assume $x\in\mathtt{M}$ in each $\mathtt{c}!x$, $\mathtt{c}?x$.) We construct a $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $\mathcal{P}=(\Sigma,\Gamma,u,v,R,R')$ intended to express the existence of a run from C_{in} to C_{fi}.

We first put $\Sigma\overset{\text{def}}{=}\Delta_1\cup\Delta_2$ and $\Gamma\overset{\text{def}}{=}\mathtt{M}$ so that words $\sigma\in\Sigma^*$ are sequences of UCST rules and their images $u(\sigma),v(\sigma)\in\Gamma^*$ are sequences of messages. With any $\delta\in\Sigma$, we associate $\mathit{write\_}\mathtt{r}(\delta)$ defined by $\mathit{write\_}\mathtt{r}(\delta)=x$ if δ is a sender rule of the form $.\xrightarrow{\mathtt{r}!x}.$, and $\mathit{write\_}\mathtt{r}(\delta)=\varepsilon$ in all other cases. This is extended to sequences with $\mathit{write\_}\mathtt{r}(\delta_1\cdots\delta_n)=\mathit{write\_}\mathtt{r}(\delta_1)\cdots\mathit{write\_}\mathtt{r}(\delta_n)$. In a similar way we define $\mathit{write\_}\mathtt{l}(\sigma)\in\mathtt{M}^*$, the sequence written to $\mathtt{l}$ by the sequence σ, and $\mathit{read\_}\mathtt{r}(\sigma)$ and $\mathit{read\_}\mathtt{l}(\sigma)$, the sequences read by σ from $\mathtt{r}$ and $\mathtt{l}$, respectively. We define $E_{\mathtt{r}}\in\mathsf{Reg}(\Sigma)$ where $E_{\mathtt{r}}\overset{\text{def}}{=}E_1\cup E_2$ and

$$E_1\overset{\text{def}}{=}\{\delta\in\Sigma\mid \mathit{write\_}\mathtt{r}(\delta)=\mathit{read\_}\mathtt{r}(\delta)=\varepsilon\},$$
$$E_2\overset{\text{def}}{=}\{\delta_1\cdot\delta_2\in\Sigma^2\mid \mathit{write\_}\mathtt{r}(\delta_1)=\mathit{read\_}\mathtt{r}(\delta_2)\neq\varepsilon\}.$$

In other words, E_1 gathers the rules that do not write to or read from $\mathtt{r}$, and E_2 contains all pairs of sender/receiver rules that write/read a same letter to/from $\mathtt{r}$.

Let now $P_1\subseteq\Delta_1^*$ be the set of all sequences of sender rules of the form $p_{\text{in}}=p_0\xrightarrow{}p_1\xrightarrow{}p_2\cdots\xrightarrow{}p_n=p_{\text{fi}}$, i.e., sequences which take the sender state from p_{in} to p_{fi}.[4] Similarly, let $P_2\subseteq\Delta_2^*$ be the set of all sequences of receiver rules which take the receiver component from q_{in} to q_{fi}. Since P_1 and P_2 are defined by finite state systems, they are regular languages. We write $P_1\|P_2$ to denote the set of all interleavings (shuffles) of a word in P_1 with a word in P_2. This operation is regularity-preserving, so $P_1\|P_2\in\mathsf{Reg}(\Sigma)$. Let $Z_{\mathtt{l}}\subseteq\Delta_1$ be the set of all sender rules which test the emptiness of $\mathtt{l}$ (which are the only test rules in S). We define R and R' as the following regular languages:

$$R=E_{\mathtt{r}}^*\cap(P_1\|P_2),\qquad\qquad R'=Z_{\mathtt{l}}\cdot\big(\Delta_1\cup\Delta_2\big)^*.$$

Finally, the morphisms $u,v:\Sigma^*\to\Gamma^*$ are given by $u\overset{\text{def}}{=}\mathit{read\_}\mathtt{l}$ and $v\overset{\text{def}}{=}\mathit{write\_}\mathtt{l}$.

[3] This problem is actually called $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ in [25].

[4] I.e., all paths from p_{in} to p_{fi} in the directed graph of the sender, seeing rules as directed edges.

Lemma 5.3 (Correctness). *S has a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ iff $\mathcal{P}$ has a solution.*

Proof. We first introduce a notion bridging the difference between runs of S and solutions of $\mathcal{P}$. We call $\sigma \in (\Delta_1 \cup \Delta_2)^*$ a *pre-solution* if all the following conditions hold:

1. $\sigma \in P_1 \| P_2$;
2. $\mathit{read\_}\mathtt{r}(\sigma) = \mathit{write\_}\mathtt{r}(\sigma)$;
3. $\mathit{read\_}\mathtt{r}(\sigma_1)$ is a prefix of $\mathit{write\_}\mathtt{r}(\sigma_1)$ for each prefix σ_1 of σ;
4. $\mathit{read\_}\mathtt{l}(\sigma) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma)$;
5. for each factorization $\sigma = \sigma_1 z \sigma_2$ where $z \in Z_1$ we have $\mathit{read\_}\mathtt{l}(\sigma_2) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma_2)$.

A pre-solution σ has a *receiver-advancing switch* if $\sigma = \sigma_1 \delta\delta' \sigma_2$ where δ is a sender rule, δ' is a receiver rule, and $\sigma' = \sigma_1 \delta' \delta \sigma_2$ is a pre-solution. A *receiver-postponing switch* is defined analogously, for δ being a receiver rule and δ' being a sender rule.

It is obvious that if there is a pre-solution σ then there is an *advance-stable pre-solution* σ', which means that σ' has no receiver-advancing switch; there is also a *postpone-stable pre-solution* σ'' which has no receiver-postponing switch.

Claim. Any advance-stable pre-solution σ is in $E_{\mathtt{r}}^*$, and it is thus a solution of $\mathcal{P}$.

Proof of the claim. Let us write an advance-stable pre-solution σ as $\sigma_1\sigma_2$ where σ_1 is the longest prefix such that $\sigma_1 \in E_{\mathtt{r}}^*$; hence $\mathit{read\_}\mathtt{r}(\sigma_1) = \mathit{write\_}\mathtt{r}(\sigma_1)$ by the definition of $E_r = E_1 \cup E_2$. Now suppose $\sigma_2 \neq \varepsilon$. Then $\sigma_2 = \delta_1\delta_2 \cdots \delta_k$ where $\delta_1 \notin E_1$. Since now $\sigma_1 \in E_{\mathtt{r}}^*$, hence $\mathit{read\_}\mathtt{r}(\sigma_1) = \mathit{write\_}\mathtt{r}(\sigma_1)$, δ_1 must be of the form $. \xrightarrow{\mathtt{r}!x} .$ (to keep 3.). Let us pick the smallest ℓ such that $\delta_\ell = . \xrightarrow{\mathtt{r}?x} .$ (which must exist by 2.) and note that $\ell \geq 3$ since $\sigma_1\delta_1\delta_2 \notin E_{\mathtt{r}}^*$. If we now pick the first j with $1 \leq j \leq \ell - 1$ and such that δ_j is a sender rule and δ_{j+1} is a receiver rule, switching δ_j, δ_{j+1} leads again to a pre-solution (as can be checked by inspecting 1.–5.). This contradicts the assumption that σ is an advance-stable pre-solution.

Claim. Any postpone-stable pre-solution σ corresponds to a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ of S.

Proof of the claim. Consider a presentation $\sigma = \sigma_1\sigma_2 z \sigma_3$ where $z \in Z_1$, σ_2 contains no rules from Z_1, and σ_1 is either empty or finishes with some $z' \in Z_1$; recall that $\mathit{read\_}\mathtt{l}(\sigma_2 z \sigma_3) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma_2 z \sigma_3)$ and $\mathit{read\_}\mathtt{l}(\sigma_3) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma_3)$. We then must have $\mathit{read\_}\mathtt{l}(\sigma_2) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma_2)$: otherwise we had $\sigma_2 = \sigma'\delta\sigma''$ where δ is of the form $. \xrightarrow{\mathtt{l}?x} .$, σ'' contains no $\mathtt{l}$-reading rules and $\mathit{read\_}\mathtt{l}(\delta\sigma'' z \sigma_3) \sqsubseteq \mathit{write\_}\mathtt{l}(\sigma_3)$; then switching the leftmost receiver-sender pair in $\delta\sigma'' z$ would lead to a pre-solution, as can be easily checked. Moreover, in σ_2 each sender rule precedes all receiver rules. It is now easy to verify that there is a run $C_{\text{in}} \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} \cdots C_{n-1} \xrightarrow{\delta_n} C_{\text{fi}}$ of S where $\delta_1\delta_2 \cdots \delta_n = \sigma$.

Finally we observe that if $C_{\text{in}} \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} \cdots C_{n-1} \xrightarrow{\delta_n} C_{\text{fi}}$ is a run of S then $\sigma = \delta_1\delta_2 \cdots \delta_n$ is a pre-solution; then there is also an advance-stable pre-solution, i.e. a solution of $\mathcal{P}$. On the other hand, if σ is a solution of $\mathcal{P}$ then σ is a pre-solution, and then there is a postpone-stable pre-solution, which corresponds to a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ of S. □

Actually, $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and UCST[Z_1^1] are equivalent (inter-reducible) problems:

Theorem 5.4. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *many-one reduces to the Reachability Problem for UCST[$Z_i^{\mathtt{c}}$] for any $i \in \{1,2\}$ and $\mathtt{c} \in \mathtt{Ch}$.*

Proof (Idea). These reductions are easy and follow basically the same pattern: a UCST system nondeterministically guesses a solution and validates it. As an example, let us informally describe the simplest one and show how to solve a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance with a UCST[$Z_1^{\mathtt{r}}$] system. We recall from [25] that $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the question whether there is a $\sigma \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ and furthermore $u(\sigma') \sqsubseteq v(\sigma')$ for all *prefixes* of σ that belong to R' (thus $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are equivalent problems and one switches from one to the other by taking the mirror images of u, v, R, R').

Given $(\Sigma, \Gamma, u, v, R, R')$ we build an UCST where Sender nondeterministically generates a $\sigma \in R$, sending $u(\sigma)$ on channel $\mathtt{r}$ and $v(\sigma)$ on channel $\mathtt{l}$. A subword of $v(\sigma)$ is written on $\mathtt{l}$. Receiver checks that $\mathtt{l}$ and $\mathtt{r}$ contain exactly the same sequence of messages, that is, $u(\sigma)$. Whenever the prefix of σ generated so far (call it σ') is in R', Sender waits for $\mathtt{r}$ to be empty before going on with the generation of σ. This forces Receiver to match $u(\tau)$ with a prefix of $v(\tau)$, or more precisely, with a prefix of the subword of $v(\tau)$ that ends up in $\mathtt{l}$ after message losses may have occurred.

The other three reductions are similar. □

6 Reducing UCST[Z_1] to UCST[$Z_1^{\mathtt{l}}$]

In this section we prove the decidability of reachability for UCST[Z_1] by reducing to UCST[$Z_1^{\mathtt{l}}$]. Since this involves eliminating Z tests on $\mathtt{r}$, the configurations in which $\mathtt{r}$ is empty are of interest. For a UCST S, we let $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ be the subset of configurations (p, q, ε, v) in which $\mathtt{r}$ is empty. We abuse terminology and say that a subset $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is *regular* if there are some state-indexed regular languages $(V_{p,q})_{p\in Q_1, q\in Q_2}$ in $\mathsf{Reg}(\mathtt{M})$ such that $W = \{(p, q, \varepsilon, v) \mid v \in V_{p,q}\}$. Such regular subsets of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ can be finitely represented using, e.g., regular expressions or finite-state automata.

We have put $C = (p, q, u, v) \sqsubseteq C' = (p', q', u', v')$ iff $p = p'$, $q = q'$, $u = u'$, and $v \sqsubseteq v'$. $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ is thus a well-quasi order under $\sqsubseteq$, unlike Conf.

$W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is *upward-closed* (in $\mathit{Conf}_{\mathtt{r}=\varepsilon}$) if $C \in W$, $C \sqsubseteq C'$ and $C' \in \mathit{Conf}_{\mathtt{r}=\varepsilon}$ imply $C' \in W$. It is *downward-closed* if $\mathit{Conf}_{\mathtt{r}=\varepsilon} \smallsetminus W$ is upward-closed. The upward-closure $\uparrow W$ of $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is the smallest upward-closed set that contains W. A well-known consequence of Higman's Lemma is that upward-closed and downward-closed subsets of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ are regular, and that upward-closed subsets can be canonically represented by their finitely many minimal elements.

For $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$, we let $\mathrm{Pre}^*(W) \overset{\text{def}}{=} \{C \in \mathit{Conf}_{\mathtt{r}=\varepsilon} \mid \exists D \in W : C \xrightarrow{*} D\}$: observe that $\mathrm{Pre}^*(W)$ only contains configurations with empty $\mathtt{r}$.

Lemma 6.1. *If W is an upward-closed subset of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ and if S is a UCST[$Z_1^{\mathtt{l}}$], then* $\mathrm{Pre}^*(W)$ *is upward-closed and is computable uniformly from S and W.*

Proof (Sketch). That $\mathrm{Pre}^*(W)$ is upward-closed is an immediate consequence of the definition of lossy steps in Eq. (1). That it is computable from S and W is more interesting: this is an application of the VJGL Lemma: "an upward-closed set U is computable if one can decide $C \in U$ and $V \cap U \neq \varnothing$ for arbitrary configurations C and regular sets V" (see [26, Theorem 2] for details). Here the two questions, "$C \in U$?" and "$V \cap U \neq \varnothing$?", i.e., "$C \xrightarrow{*} W$?" and "$U \xrightarrow{*} W$?", reduce to instances of the extended reachability problem for UCST[$Z_1^{\mathtt{l}}$], hence are decidable. □

Theorem 6.2. *Reachability is decidable for UCST[Z_1].*

Proof (Sketch). Given a UCST[Z_1] S, a run $\pi = C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ can be presented in the form

$$(C_{\text{in}} =)\, C_0 \xrightarrow{*}_{S'} D_1 \xrightarrow{Z:\mathtt{r}} C_1 \xrightarrow{*}_{S'} D_2 \xrightarrow{Z:\mathtt{r}} C_2 \cdots \xrightarrow{*}_{S'} D_m \,(= C_{\text{fi}})$$

where the $D_i \to C_i$ steps gather all occurrences of $Z_1^{\mathtt{r}}$ tests: note that necessarily D_i and C_i are in $Conf_{\mathtt{r}=\varepsilon}$. The $C_{i-1} \xrightarrow{*}_{S'} D_i$ subruns can be seen as runs of a new system S', which is obtained from S by removing all $Z_1^{\mathtt{r}}$ testing rules from Δ_1. The point is that we can apply Lemma 6.1 to S' since it is a UCST[Z_1^{1}].

So for $k = 0,1,\ldots$, we define T'_k and T_k by letting $T'_0 = \uparrow C_{\text{fi}}$, $T_k = \text{Pre}^*_{S'}(T'_k)$ and $T'_{k+1} = T'_k \cup \{C \mid \exists D \in T_k : C \xrightarrow{Z:\mathtt{r}} D\}$ (note that T_k is defined with a $\text{Pre}^*_{S'}$ restricted to S'). T_k collects all configurations $C \in Conf_{\mathtt{r}=\varepsilon}$ from which one can reach T_0 with at most k uses of a $Z : \mathtt{r}$ test. We observe that all T'_k, T_k are upward-closed subsets of $Conf_{\mathtt{r}=\varepsilon}$, that T_k is computable from T'_k by Lemma 6.1, and that T'_{k+1} is obviously computable from T_k and T'_k. Furthermore, the T_k's are increasing: $T_0 \subseteq T_1 \subseteq \cdots T_k \subseteq T_{k+1} \cdots$. Since they are upward-closed, they eventually stabilize by the well-quasi-ordering property: letting $T_\omega \overset{\text{def}}{=} \bigcup_{k\in\mathbb{N}} T_k$, there is n such that $T_n = T_{n+1} = T_\omega$. Since there is a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ of S iff $C_{\text{in}} \in T_\omega$, the proof is finished. □

Observe that Lemma 6.1 and Theorem 6.2 exhibit a Turing reduction (from reachability for UCST[Z_1] to reachability for UCST[Z_1^1]) and not a many-one reduction like all the other reductions in this paper.

With the results of sections 4 and 5, one obtains the following corollary.

Theorem 6.3. *Reachability is decidable for UCST[Z,N].*

Remark 6.4 (On complexity). Based on known results on the complexity of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ (see [17, 25]), our reductions prove that reachability for UCST[Z,N] is at level $\mathcal{F}_{\omega^\omega}$ in the extended Grzegorczyck hierarchy, and at level $\mathcal{F}_{\omega^{m-1}}$, where $m = |\mathtt{M}|$, when we restrict to systems with a fixed-sized alphabet of messages. □

7 Some Undecidable Problems for UCST[Z,N]

The main result of this paper is that reachability is decidable for UCST[Z,N] (Theorem 6.3). In this section we argue that emptiness and non-emptiness tests strictly add to the expressive power of UCS's. This point is made in two different ways.

We start with recurrent reachability. Formally, the Recurrent Reachability Problem asks whether a given S has an infinite run $C_{\text{in}} \xrightarrow{+} (p,q,u_1,v_1) \xrightarrow{+} (p,q,u_2,v_2) \xrightarrow{+} \cdots$ visiting infinitely often a given control pair $(p,q) \in Q_1 \times Q_2$ (but with no constraints on channel contents).

Theorem 7.1. *Recurrent reachability is undecidable for UCST[$Z_1^{\mathtt{r}}$].*

Proof (Idea). We prove Theorem 7.1 by reducing from the undecidable question whether a length-preserving string rewrite system (aka semi-Thue system) has a loop

$x \xrightarrow{+} x$. We design a UCST S where Sender guesses a word y_0, writes it on `l`, and then guesses pairs x_i, y_i for $i = 1, 2, \ldots$ such that each $x_i \to y_i$ is a rewrite step. It writes x_i on `r` and y_i on `l`. Receiver's job is to check that $y_{i-1} = x_i$. With $Z_1^{\mathtt{r}}$ tests, Sender can wait for a check on x_i to be concluded before issuing the next pair. This way we ensure progress of the checking phase and avoid confusion between pairs if a separator is lost. Since the rewrite system is length-preserving, any infinite run of S must eventually stop losing messages and witness a loop. □

Since recurrent reachability is decidable for UCS (see [22]), Theorem 7.1 shows that Z tests, even just $Z_1^{\mathtt{r}}$ tests, cannot be simulated in UCS's.

As another illustration, we consider UCST's with *write-lossy semantics*, that is, UCST's with the assumption that messages are only lost during steps that (attempt to) write them to `l`. Once they are in `l`, they are never lost. This is formalized via a new transition relation $C \to_{\text{wrlo}} C'$ (definition omitted, but as expected) that is intermediary between $\to_{\text{rel}}$ and $\to_{\text{los}}$.

In many cases the two lossy semantics coincide:

Lemma 7.2. *Assume S is a UCST[Z] system. Then $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ iff $C_{\text{in}} \xrightarrow{*}_{\text{wrlo}} C_{\text{fi}}$.*

Proof (Idea). Prove that $C \xrightarrow{\delta}_{\text{los}} C'$ iff $D \xrightarrow{\delta}_{\text{wrlo}} C'$ for some $D \sqsubseteq C$. Deduce $C_{\text{in}} \xrightarrow{n+1}_{\text{los}} C'$ iff $C_{\text{in}} \xrightarrow{n+1}_{\text{wrlo}} C'$ by induction on n. See [18, App. A]. □

Corollary 7.3. *Reachability is decidable for UCST[Z] with write-lossy semantics.*

Remark 7.4. Write-lossy semantics is meaningful when modeling unreliability of the writing actions as opposed to unreliability of the channels. However, in the literature, write-lossy semantics is mostly used as a way of restricting the nondeterminism of lossy channel systems without losing any essential generality, as stated by Lemma 7.2. □

Write-lossy and (plain) lossy semantics do not coincide when N tests are allowed. In fact, Theorem 6.3 does not extend to write-lossy systems.

Theorem 7.5. *Reachability is undecidable for UCST[$Z_1^{\mathtt{l}}, N_1^{\mathtt{l}}$] with write-lossy semantics.*

Proof (Idea). As before, Sender simulates queue automata using tests and the help of Receiver. See Fig. 3. Channel `l` is initially empty. To read a from `r`, Sender does the following: (1) write a on `l`; (2) check that `l` is nonempty (hence the write was not lost); (3) check that, or wait until, `l` is empty. Meanwhile, Receiver reads identical letters from `r` and `l`. □

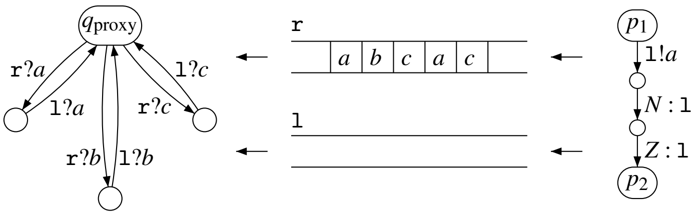


Fig. 3. Write-lossy Sender simulates "$p_1 \xrightarrow{\mathtt{r}?a} p_2$" with N and Z tests and proxy Receiver

Thus, at least in the write-lossy setting, we can separate UCST[Z] and UCST[Z, N_1^1] w.r.t. decidability of reachability.

8 Conclusion

UCS's are communicating systems where a Sender can send messages to a Receiver via one reliable and one unreliable, lossy, channel, but where no direct communication is possible in the other direction. We introduced UCST, an extension of UCS's where steps can be guarded by tests, i.e., regular predicates on channel contents. This extension introduces limited but real possibilities for synchronization between Sender and Receiver. For example, Sender (or Receiver) may use tests to detect whether the other agent has read (or written) some message. As a consequence, adding tests leads to undecidable reachability problems in general. Our main result is that reachability remains decidable when only emptiness and non-emptiness tests are allowed. The proof goes through a series of reductions from UCST[Z, N] to UCST[Z_1^1] and finally to $\mathsf{PEP}_{\text{codir}}^{\text{partial}}$, an extension of Post's Embedding Problem that was motivated by the present paper and whose decidability was recently proved by the last two authors [25].

We see two main directions for future works:

1. The limits of decidability: is it possible to characterize precisely the families of tests $T \subseteq \mathsf{Reg}(\mathtt{M})$ for which UCST[T] has a decidable reachability problem? We gave positive and negative examples, but a precise characterization would help understand the phenomenon at hand.
2. Beyond reachability: we focused on reachability questions since they are the most natural starting point as far as verification is concerned. However several other natural verification problems, e.g., termination, are known to be decidable for UCS's.

References

1. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
2. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. Formal Methods in System Design 14(3), 237–255 (1999)
3. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theor. Comp. Sci. 221(1–2), 211–250 (1999)
4. Atig, M.F., Bouajjani, A., Touili, T.: On the Reachability Analysis of Acyclic Networks of Pushdown Systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 356–371. Springer, Heidelberg (2008)
5. Muscholl, A.: Analysis of Communicating Automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 50–57. Springer, Heidelberg (2010)
6. Cécé, G., Finkel, A., Purushothaman Iyer, S.: Unreliable channels are easier to verify than perfect channels. Inf. Comp. 124(1), 20–31 (1996)
7. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comp. 127(2), 91–101 (1996)
8. Abdulla, P.A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B.: Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design 25(1), 39–65 (2004)

9. Abdulla, P.A., Bertrand, N., Rabinovich, A., Schnoebelen, P.: Verification of probabilistic systems with faulty communication. Inf. Comp. 202(2), 141–165 (2005)
10. Baier, C., Bertrand, N., Schnoebelen, P.: On Computing Fixpoints in Well-Structured Regular Model Checking, with Applications to Lossy Channel Systems. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 347–361. Springer, Heidelberg (2006)
11. Baier, C., Bertrand, N., Schnoebelen, P.: Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. ACM Trans. Comput. Logic 9(1) (2007)
12. Abdulla, P.A., Deneux, J., Ouaknine, J., Worrell, J.B.: Decidability and Complexity Results for Timed Automata via Channel Machines. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1089–1101. Springer, Heidelberg (2005)
13. Ouaknine, J., Worrell, J.B.: On Metric Temporal Logic and Faulty Turing Machines. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 217–230. Springer, Heidelberg (2006)
14. Kurucz, A.: Combining modal logics. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logics, vol. 3, ch. 15, pp. 869–926. Elsevier Science (2006)
15. Konev, B., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Dynamic topological logics over spaces with continuous functions. In: Advances in Modal Logic, vol. 6, pp. 299–318. College Publications (2006)
16. Lasota, S., Walukiewicz, I.: Alternating timed automata. ACM Trans. Computational Logic 9(2) (2008)
17. Schmitz, S., Schnoebelen, P.: Multiply-Recursive Upper Bounds with Higman's Lemma. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 441–452. Springer, Heidelberg (2011)
18. Chambart, P., Schnoebelen, P.: The ordinal recursive complexity of lossy channel systems. In: LICS 2008, pp. 205–216. IEEE Comp. Soc. Press (2008)
19. Haddad, S., Schmitz, S., Schnoebelen, P.: The ordinal-recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In: LICS 2012, pp. 355–364. IEEE Comp. Soc. Press (2012)
20. Chambart, P., Schnoebelen, P.: Mixing Lossy and Perfect Fifo Channels. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 340–355. Springer, Heidelberg (2008)
21. Chambart, P., Schnoebelen, Ph.: Post Embedding Problem Is Not Primitive Recursive, with Applications to Channel Systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)
22. Chambart, P., Schnoebelen, P.: The ω-Regular Post Embedding Problem. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 97–111. Springer, Heidelberg (2008)
23. Chambart, P., Schnoebelen, P.: Pumping and Counting on the Regular Post Embedding Problem. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 64–75. Springer, Heidelberg (2010)
24. Barceló, P., Figueira, D., Libkin, L.: Graph logics with rational relations and the generalized intersection problem. In: LICS 2012, pp. 115–124. IEEE Comp. Soc. Press (2012)
25. Karandikar, P., Schnoebelen, P.: Cutting through Regular Post Embedding Problems. In: Lepistö, A. (ed.) CSR 2012. LNCS, vol. 7353, pp. 229–240. Springer, Heidelberg (2012)
26. Goubault-Larrecq, J.: On a generalization of a result by Valk and Jantzen. Research Report LSV-09-09, Laboratoire Spécification et Vérification, ENS Cachan, France (May 2009)

On Properties and State Complexity of Deterministic State-Partition Automata

Galina Jirásková[1,⋆] and Tomáš Masopust[2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 01 Košice, Slovak Republic
jiraskov@saske.sk
[2] Institute of Mathematics, Academy of Sciences of the Czech Republic
Žižkova 22, 616 62 Brno, Czech Republic
masopust@math.cas.cz

Abstract. A deterministic automaton accepting a regular language L is a state-partition automaton with respect to a projection P if the state set of the deterministic automaton accepting the projected language $P(L)$, obtained by the standard subset construction, forms a partition of the state set of the automaton. In this paper, we study fundamental properties of state-partition automata. We provide a construction of the minimal state-partition automaton for a regular language and a projection, discuss closure properties of state-partition automata under the standard constructions of deterministic automata for regular operations, and show that almost all of them fail to preserve the property of being a state-partition automaton. Finally, we define the notion of a state-partition complexity, and prove the tight bound on the state-partition complexity of regular languages represented by incomplete deterministic automata.

Keywords: Regular languages, finite automata, descriptional complexity, projections, state-partition automata.

1 Introduction

A deterministic finite automaton G accepting a regular language L is a *state-partition automaton* with respect to a projection P if the state set of the deterministic automaton accepting the projected language $P(L)$, obtained by the standard subset construction [5,23], forms a partition of the state set of the automaton G. This means that the projection of a string uniquely specifies the state of the projected automaton. Therefore, all projected strings of a language with the same observation, that is, with the same projections, lead to the same state of the projected automaton. This property immediately implies that the size of the minimal state-partition automaton is not smaller than the size of the minimal deterministic automaton accepting the projected language.

⋆ Research supported by the Slovak Research and Development Agency under contract APVV-0035-10 "Algorithms, Automata, and Discrete Data Structures".

⋆⋆ Research supported by the GAČR grant no. P202/11/P028 and by RVO: 67985840.

From the practical point of view, state-partition automata are of interest in engineering and computer science, especially in applications where the user, supervisor, or controller has only a partial observation of the whole behavior of a system, which is modeled by a projection. From the theoretical point of view, state-partition automata have found applications as a proof formalism for systems with partial observations. Namely, they have been successfully used to simplify constructions and proofs, and are useful in applications of natural projections to obtain or describe an abstraction of a system. Note that projections are sometimes generalized to so-called *causal reporter maps*, see [21,24]. We refer the reader to [3,4,11,12] for applications of state-partition automata in supervisory control of discrete-event systems. Note that state-partition automata are related to the Schützenberger covering. More specifically, the construction of a state-partition automaton is close to the Schützenberger construct [15].

A system represented by a state-partition automaton with respect to a projection that describes an abstraction or a partial observation has a projected automaton that is not larger than the original automaton. This is the most important property from the application point of view. Notice that, up to now, there is only one well-known condition ensuring that a projected automaton is smaller than the original automaton, an *observer property*, cf. [20]. The study of state-partition automata is thus a further step to the understanding and characterization of the class of automata useful for practical applications in, e.g., coordination or hierarchical supervisory control of discrete-event systems [1,9,10,17,18].

In this paper, we discuss fundamental properties of state-partition automata. In Section 3, we recall the known result proving that every regular language has a state-partition automaton with respect to a given projection. A procedure to construct this automaton is also known, see [3]. We repeat the construction here and use it to obtain the minimal state-partition automaton for a given language and a projection. The last result of this section describes a regular language and two projections with respect to which the language has no state-partition automaton. This negative result indicates that state-partition automata are useful for systems with either a partial observation or abstraction, but not with the combination of both.

Then, in Section 4, we study the closure properties of state-partition automata under the standard constructions of deterministic automata for the operations of complement, union, intersection, concatenation, Kleene star, reversal, cyclic shift, and left and right quotients. We show that almost all of them fail to preserve the property of being a state-partition automaton. Only two of the considered operations preserve this property, namely, the construction of a deterministic automaton for the right quotient of two regular languages, and the construction of a deterministic automaton for the complement of regular languages represented by complete deterministic automata.

Finally, in the last section of this paper, we introduce and study the *state-partition complexity* of regular languages with respect to a projection, defined as the smallest number of states in any state-partition automaton (with respect to the projection) accepting the language. The first result of this section shows that a language represented by a minimal incomplete deterministic automaton with n states has state-partition complexity at most $3n \cdot 2^{n-3}$. The second result then proves the tightness of this upper bound using a language defined over a three-letter alphabet and a projection on binary strings.

2 Preliminaries and Definitions

In this paper, we assume that the reader is familiar with the basic notions and concepts of formal languages and automata theory, and we refer the reader to [5,14,16] for all details and unexplained notions.

For a finite non-empty set Σ, called an alphabet, the set Σ^* represents the free monoid generated by Σ. A string over Σ is any element of Σ^*, and the unit of Σ^* is the empty string denoted by ε. A language over Σ is any subset of Σ^*. For a string w in Σ^*, let $|w|$ denote the length of w, and for a symbol a in Σ, let $|w|_a$ denote the number of occurrences of the symbol a in w. If $w = xyz$, for strings x, y, z, w in Σ^*, then x is a prefix of w, and y is a factor of w.

A *deterministic finite automaton* (a DFA, for short) is a quintuple $G = (Q, \Sigma, \delta, s, F)$, where Q is a finite non-empty set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $s \in Q$ is the initial (or start) state, and $F \subseteq Q$ is the set of final states. Note that we consider *incomplete* deterministic finite automata that are also called *generators* in the literature, cf. [2,22]. That is why we prefer to use G to denote an incomplete deterministic automaton. The transition function can be naturally extended to the domain $Q \times \Sigma^*$ by induction. The language *accepted* by the automaton G is the set of strings $L(G) = \{w \in \Sigma^* \mid \delta(s, w) \in F\}$. A state q of G is called *reachable* if $q = \delta(s, w)$ for a string w in Σ^*, and it is called *useful*, or *co-reachable*, if $\delta(q, w) \in F$ for a string w.

A *nondeterministic finite automaton* (an NFA, for short) is a quintuple $N = (Q, \Sigma, \delta, S, F)$, where Q, Σ, and F are as in a DFA, $S \subseteq Q$ is the set of initial states, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the nondeterministic transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language *accepted* by the NFA N is defined as the set $L(N) = \{w \in \Sigma^* \mid \delta(S, w) \cap F \neq \emptyset\}$. Notice that our NFAs may have ε-transitions and multiple initial states. However, ε-transitions and multiple initial states can be eliminated by a standard technique [5].

Two automata are equivalent if they accept the same language. Every NFA $N = (Q, \Sigma, \delta, S, F)$ without ε-transitions can be converted to an equivalent DFA $\det(N) = (2^Q, \Sigma, \delta_d, s_d, F_d)$ by an algorithm known as the "subset construction" [13], where we have

$$\begin{aligned} \delta_d(R, a) &= \delta(R, a) \text{ for each } R \text{ in } 2^Q \text{ and } a \text{ in } \Sigma, \\ s_d &= S, \text{ and} \\ F_d &= \{R \in 2^Q \mid R \cap F \neq \emptyset\}. \end{aligned}$$

We call the deterministic automaton $\det(N)$ the *subset automaton* corresponding to the automaton N. Notice that the state set of the subset automaton is the set of all subsets of Q, even though some of them may be unreachable from the initial state s_d.

Let Σ be an alphabet and $\Sigma_o \subseteq \Sigma$. A homomorphism P from Σ^* to Σ_o^* is called a *(natural) projection* if it is defined by $P(a) = a$ for each a in Σ_o and $P(a) = \varepsilon$ for each a in $\Sigma \setminus \Sigma_o$. The *inverse image* of P is a mapping P^{-1} from Σ_o^* to 2^{Σ^*} defined by $P^{-1}(w) = \{u \in \Sigma^* \mid P(u) = w\}$.

Let $G = (Q, \Sigma, \delta, s, F)$ be a DFA accepting a language L and P be the projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. From the DFA G, we construct an NFA N_G accepting the language $P(L)$ by replacing all transitions labeled by symbols from $\Sigma \setminus \Sigma_o$ with ε-transitions, and by eliminating these ε-transitions. Then the *projected automaton* for the language $P(L)$ is the deterministic automaton

$$P(G) = (Q', \Sigma_o, \delta', s', F')$$

that forms the reachable part of the subset automaton $\det(N_G)$. Thus, Q' is the set of all states of 2^Q reachable from the initial state s'. Notice that we do not eliminate states, from which no final state is reachable. This is due to applications in supervisory control, where this problem is known as the problem of *nonblockingness* [2].

A DFA $G = (Q, \Sigma, \delta, s, F)$ is a *state-partition automaton* (an SPA, for short) with respect to a projection P from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$ if the states of the projected automaton $P(G) = (Q', \Sigma_o, \delta', s', F')$ are pairwise disjoint as sets. Note that if all states of G are reachable, then the state set of the projected automaton $P(G)$ defines a partition of the state set of G.

For an automaton A (deterministic or nondeterministic), let $\mathrm{sc}(A)$ denote the number of states of the automaton A.

We immediately have the following result.

Lemma 1. *Let G be a DFA over an alphabet Σ that has no unreachable states. Let P be a projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. If G is a state-partition automaton with respect to P, then* $\mathrm{sc}(P(G)) \leq \mathrm{sc}(G)$. □

Now we define a parallel composition of two incomplete deterministic automata, which is basically the intersection of two automata defined over two different alphabets. Therefore, it is first necessary to unify their alphabets by adding the missing symbols.

For two deterministic finite automata $G_1 = (Q_1, \Sigma_1, \delta_1, s_1, F_1)$ and $G_2 = (Q_2, \Sigma_2, \delta_2, s_2, F_2)$, we define the *parallel composition* of G_1 and G_2, denoted by $G_1 \parallel G_2$, as the reachable part of the DFA $(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (s_1, s_2), F_1 \times F_2)$, where

$$\delta((p,q),a) = \begin{cases} (\delta_1(p,a), \delta_2(q,a)), & \text{if } \delta_1(p,a) \text{ is defined in } G_1 \text{ and} \\ & \quad \delta_2(q,a) \text{ is defined in } G_2; \\ (\delta_1(p,a), q), & \text{if } \delta_1(p,a) \text{ is defined in } G_1 \text{ and } a \notin \Sigma_2; \\ (p, \delta_2(q,a)), & \text{if } a \notin \Sigma_1 \text{ and } \delta_2(q,a) \text{ is defined in } G_2; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

From the language point of view, it can be shown that

$$L(G_1 \parallel G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2)),$$

where P_i is the projection from $(\Sigma_1 \cup \Sigma_2)^*$ to Σ_i^* for $i = 1, 2$.

Let us briefly recall definitions of the operations of reversal, cyclic shift, and left and right quotients for languages over an alphabet Σ. The *reversal of a string* w over Σ is defined by $\varepsilon^R = \varepsilon$ and $(va)^R = av^R$ for a symbol a in Σ and a string v in Σ^*. The *reversal of a language* L is the language $L^R = \{w^R \in \Sigma^* \mid w \in L\}$. The *cyclic shift* of a language L is defined as the language $L^{shift} = \{uv \in \Sigma^* \mid vu \in L\}$. The *left and right quotients* of a language L by a language K are the languages $K \backslash L = \{x \in \Sigma^* \mid \text{ there exists } w \in K \text{ such that } wx \in L\}$ and $L/K = \{x \in \Sigma^* \mid \text{ there exists } w \in K \text{ such that } xw \in L\}$, respectively. By L^c we denote the complement of a language L, that is, the language $\Sigma^* \setminus L$.

3 Minimal State-Partition Automata

The fundamental question is whether every regular language can be accepted by a state-partition automaton with respect to a given projection. If this is the case, can we construct such a state-partition automaton efficiently? The answer to this question is known, and we repeat it in the following theorem. Although a proof has been given in [3], we prefer to recall it here since some fundamental observations play a role later in the paper.

Theorem 1 ([3,4]). *Let P be a projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. Let L be a language over the alphabet Σ, and let G be a DFA accepting the language L. Then the automaton $P(G) \parallel G$ is a state-partition automaton with respect to the projection P that accepts the language L.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be a DFA accepting the language L, and let $P(G) = (Q', \Sigma_o, \delta', s', F')$ be the corresponding projected automaton. By definition of the parallel composition and the comment below the definition, we have that

$$L(P(G) \parallel G) = P^{-1}(P(L(G))) \cap L(G) = L(G)\,.$$

Hence, the automaton $P(G) \parallel G$ accepts the language L.

Let w be a string over the alphabet Σ_o. Then the state of the projected automaton $P(P(G) \parallel G)$ reached from the initial state by the string w is

$$\big\{(\delta'(s', w), q) \mid q \in \delta(s, P^{-1}(w))\big\}\ .$$

Since $\delta(s, P^{-1}(w)) = \delta'(s', w)$, by definition of the transition function of the automaton $P(G)$, the state reachable from its initial state by the string w in the DFA $P(P(G) \parallel G)$ is, in fact,

$$\{(\delta'(s', w), q) \mid q \in \delta'(s', w)\}\ .$$

It then follows that the states of the projected automaton $P(P(G) \parallel G)$ reachable by two different strings are either the same or disjoint. $\square$

Next we prove that the state-partition automaton constructed from a minimal DFA using the construction of the previous theorem is the minimal state-partition automaton with respect to the number of states. To prove this, we need the notion of isomorphic automata, and the result proved in the following lemma.

Let $G_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $G_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be two DFAs. Let f be a mapping from Q_1 to Q_2 such that

- $f(\delta_1(q, a)) = \delta_2(f(q), a)$ for each q in Q_1 and a in Σ,
- $f(s_1) = s_2$, and
- $q \in F_1$ if and only if $f(q) \in F_2$.

The mapping f is called a *homomorphism* from G_1 to G_2. If f is a bijection, then it is called an *isomorphism*, and G_1 and G_2 are said to be isomorphic.

The next lemma shows that the parallel composition of automata $P(G)$ and G is isomorphic to G for a state-partition automaton G.

Lemma 2. *Let G be an SPA with respect to a projection P from Σ^* to Σ_o^*, in which all states are reachable. Then the DFA $P(G) \parallel G$ is isomorphic to G.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be a state-partition automaton with respect to the projection P, and let $P(G) = (Q', \Sigma_o, \delta', s', F')$ be the corresponding projected automaton. Define a mapping $f : Q' \times Q \to Q$ by $f(X, q) = q$. Then it holds that $\delta(q, a) = \delta(f(X, q), a)$, and f is an isomorphism from $P(G) \parallel G$ to G. $\square$

The following result constructs the minimal state-partition automaton for a given regular language and a projection.

Theorem 2. *Let L be a regular language over an alphabet Σ, and let G be the minimal DFA accepting the language L. Let P be a projection from Σ^* to Σ_o^*. Then the DFA $P(G) \parallel G$ is the minimal state-partition automaton with respect to the projection P that accepts the language L.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be the minimal DFA accepting the language L, and let $G_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be a state-partition automaton with respect to the projection P that also accepts the language L. We may assume that all states of the DFA G_2 are reachable and useful; otherwise, we can remove unreachable and useless states from G_2 and obtain a smaller state-partition automaton.

Define a mapping $f : Q_2 \to Q$ as follows. For a state q in Q_2 that is reachable in the automaton G_2 from the initial state s_2 by a string w, set $f(q) = \delta(s, w)$, that is, $f(q)$ is a state in Q that is reachable in the automaton G from the initial state s by the string w. Notice that f is well-defined since if a state in Q_2 is reached by two different strings u and v, then states $\delta(s, u)$ and $\delta(s, v)$ must be equivalent in the automaton G, and since G is minimal, we must have $\delta(s, u) = \delta(s, v)$.

Next, we have $f(\delta_2(q, a)) = \delta(f(q), a)$ for each state q in Q_2 and symbol a in Σ, $f(s_2) = s$, and $q \in F_2$ if and only if $f(q) \in F$. Hence f is a homomorphism from G_2 to G.

Now, extend the mapping f to a mapping from the state set of the automaton $P(G_2) \parallel G_2$ to the state set of the automaton $P(G) \parallel G$ by setting

$$f(X, q) = (f(X), f(q))\,.$$

Then f is surjective. Since the automaton G_2 is a state-partition automaton with respect to the projection P, we have, using Lemma 2, that

$$\mathrm{sc}(P(G) \parallel G) \leq \mathrm{sc}(P(G_2) \parallel G_2) = \mathrm{sc}(G_2)\,.$$

This completes the proof. □

Corollary 1. *Let L be a regular language over an alphabet Σ, and let P be a projection from Σ^*. Then the minimal state-partition automaton accepting the language L is unique up to isomorphism.* □

It is natural to ask whether an automaton can be a state-partition automaton with respect to more than one projection. This property would be useful in applications, where both an abstraction and a partial observation are combined, cf. [1]. Unfortunately, the following result shows that this does not hold true in general [8].

Lemma 3. *There exist a language L and projections P and $\tilde{P}$ such that no DFA accepting the language L is a state-partition automaton with respect to both projections P and $\tilde{P}$.*

Proof. Let $\Sigma = \{a, b\}$. Let P and $\tilde{P}$ be projections from Σ^* onto $\{a\}^*$ and $\{b\}^*$, respectively. Consider the language $L = (ab)^*$. Assume that $G = (Q, \Sigma, \delta, s, F)$ is a state-partition automaton for both projections P and $\tilde{P}$ accepting the language L. Notice that the DFA G does not have any loop, that is, no state of G goes to itself on any symbol, because otherwise the automaton G would accept a string that does not belong to the language L.

Let w be a string of the language L of length at least $|Q|$. Then at least one state appears twice in the computation of the automaton G on the string w. Let p be the first such state. Then $w = xyz$, where x is the shortest prefix of w such that the initial state s goes to state p by x, and y is the shortest non-empty factor of w by which p goes to itself. Since the automaton G has no loops, the length of y is at least two. Therefore, $y = cy'd$, where $c, d \in \{a, b\}$. In addition, $c \neq d$ because $xyyz = xcy'dcy'dz$ belongs to the language L. Let q be the state of the automaton G that is reached from the state p on reading the string cy'. Fig. 1 illustrates the computation of G on the string w. Since x is the shortest prefix of w that moves G to state p, and y is the shortest non-empty factor of w by which p goes to itself, we have $p \neq q$.

In case $d = b$, we consider the projected automaton

$$P(G) = (Q', \{a\}, \delta', s', F')\,.$$

Let $X = \delta'(s', P(x))$ and $Y = \delta'(X, P(ay'))$ be two states of the automaton $P(G)$. Then $p \in X$ and $p, q \in Y$. Notice that $X = \delta(s, P^{-1}(P(x)))$. Since

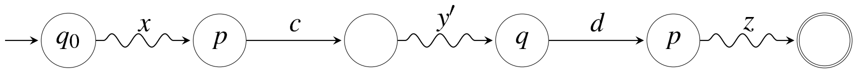


Fig. 1. The computation of G on the string $w = xcy'dz$

$c = a$ and $w \in L$, we have $x = (ab)^k$ for a non-negative integer k. Therefore, $P^{-1}(P(x)) = P^{-1}(a^k)$.

Assume that there exists a string u in $P^{-1}(a^k)$ that moves the automaton G from the initial state s to the state q. Then the string udz is accepted by the automaton G. Since $d = b$, we must have $u = (ab)^{k-1}a$. However, then the state q would be the first state in the computation on the string w that appears at least twice in it, which contradicts the choice of the state p. It follows that $q \notin X$, and, therefore, $X \neq Y$. Hence, the automaton G is not a state-partition automaton with respect to the projection P.

The case $d = a$ is similar. □

4 Closure Properties

Since every regular language has a state-partition automaton with respect to a given projection, the class of languages accepted by state-partition automata is closed under all regular operations. In the following, we consider the closure properties of state-partition automata under the standard *constructions* of deterministic automata for regular operations as described in the literature [5,16,19,23]. Hence, we investigate the following question: Given state-partition automata with respect to a projection, is the deterministic automaton resulting from the standard construction for a regular operation a state-partition automaton with respect to the same projection?

We prove that almost all standard constructions, except for the complement of complete state-partition automata and right quotient, fail to preserve the property of being a state-partition automaton.

Theorem 3. *State-partition automata are not closed under the operations of complement, intersection, union, concatenation, star, reversal, cyclic shift, and left quotient.*

Proof. We briefly recall the standard construction of a deterministic automaton for each operation under consideration. Let us emphasize that we do not minimize the resulting deterministic automata.

Complement: To get a deterministic automaton for complement from a possibly incomplete DFA G, add the dead state, if necessary, and interchange the final and non-final states. We prove that state-partition automata are not closed under this operation.

Consider the two-state DFA G in Fig. 2 (left). The DFA accepts the language ab^*. Let P be the projection from $\{a,b\}^*$ to $\{a\}^*$. Then G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$

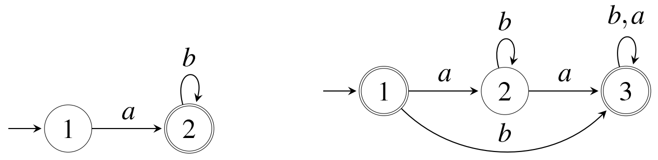


Fig. 2. SPA G (left), and DFA G^c for the complement of the language $L(G)$ (right); projection $P : \{a,b\}^* \to \{a\}^*$

is deterministic. However, the complement of G, the DFA G^c shown in Fig. 2 (right), is not a state-partition automaton with respect to the projection P because we have to add the dead state, 3, which then appears in two different reachable sets of the projected automaton $P(G^c)$, namely, in $\{1,3\}$ reached by ε and in $\{2,3\}$ reached by a. However, as the next theorem shows, the resulting DFA is a state-partition automaton if the given DFA is complete.

Intersection and Union: To get the deterministic automaton for intersection and union, we apply the standard cross-product construction.

Consider two automata G_1 and G_2 shown in Fig. 3, and their cross-product automaton $G_1 \times G_2$ depicted in Fig. 3. In the case of intersection, the only final state is state 3, while in the case of union, the final states are states 3 and 4. Let P be the projection from $\{a,b\}^*$ to $\{a\}^*$. Both G_1 and G_2 are state-partition automata with respect to the projection P. However, the automaton $G_1 \times G_2$ is not since the sets $\{2,3\}$ and $\{3,4\}$ are reachable in the projected automaton $P(G_1 \times G_2)$ by strings a and aa, respectively.

Concatenation: Recall that an NFA for concatenation of two DFAs G_1 and G_2 is obtained from G_1 and G_2 by adding ε-transitions from final states of G_1 to the initial state of G_2, and by setting the initial state to be the initial state of G_1, and final states to be final states of G_2. The corresponding subset automaton restricted to its reachable states provides the resulting DFA for concatenation.

Now, let G be the DFA shown in Fig. 4 (left). Let P be the projection from $\{a,b\}^*$ to $\{b\}^*$. The projected automaton $P(G)$ is a one-state automaton and, therefore, the DFA G is a state-partition automaton with respect to the projection P. The DFA $G \cdot G$ for concatenation is depicted in Fig. 4 (right), and states $\{1,2,3\}$ and $\{1,2,3,4\}$ are reachable in the projected automaton $P(G \cdot G)$ by strings ε and b, respectively. Hence, the DFA $G \cdot G$ for concatenation is not a state-partition automaton for the projection P.

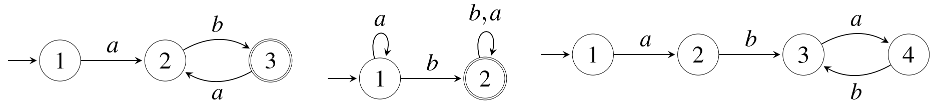


Fig. 3. SPAs G_1 (left) and G_2 (middle), and their cross-product $G_1 \times G_2$ (right); projection $P : \{a,b\}^* \to \{a\}^*$

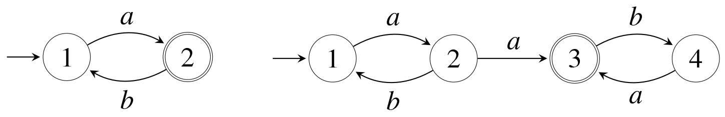


Fig. 4. SPA G (left) and DFA $G \cdot G$ for concatenation of the languages $L(G) \cdot L(G)$ (right); projection $P : \{a, b\}^* \to \{b\}^*$

Star: To construct an NFA for star of a DFA G, add a new initial and final state and ε-transitions from all final states, including the new one, to the original initial state of the automaton G. The subset construction results in a DFA for star.

Consider the DFA G in Fig. 5 (left), and the projection P from $\{a, b, c\}^*$ to $\{a, b\}^*$. The automaton G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$ is deterministic. However, the deterministic automaton G^* for star, shown in Fig. 5 (right), is not a state-partition automaton with respect to the projection P because the sets $\{3\}$ and $\{3, 4\}$ are reachable in the projected automaton $P(G^*)$ by strings ab and aba, respectively.

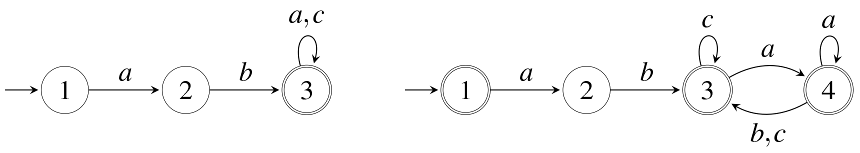


Fig. 5. SPA G (left), and DFA G^* for the star of the language $L(G)$ (right); projection $P : \{a, b, c\}^* \to \{a, b\}^*$

Reversal: We can get an NFA for reversal from a DFA G by swapping the roles of initial and final states, and by reversing all transitions. After the application of the subset construction, we obtain a DFA for reversal.

Consider the DFA G in Fig. 6 (left), and the projection P from $\{a, b, c\}^*$ to $\{a, c\}^*$. The DFA G is a state-partition automaton with respect to P since the states of the projected automaton $P(G)$ are $\{2, 3\}$ and $\{1\}$. On the other hand, the DFA G^R in Fig. 6 (right) is not a state-partition automaton with respect to the projection P because the sets $\{2\}$ and $\{2, 3\}$ are reachable in the projected automaton $P(G^R)$ by strings a and ac, respectively.

Cyclic Shift: For the construction of an NFA for cyclic shift, we refer to [7]. Fig. 7 (middle) shows an NFA for the cyclic shift of the language accepted by the DFA G of Fig. 7 (left). Let P be the projection from $\{a, b\}^*$ to $\{b\}^*$. Then G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$ has just one state $\{1, 2\}$. However, the automaton G^{shift} in Fig. 7 (right) is not a state-partition automaton with respect to the projection P since states $\{1, 2, 3\}$ and $\{2, 3, 4, 5, 6, 7, 8\}$ are reachable by strings ε and b, respectively.

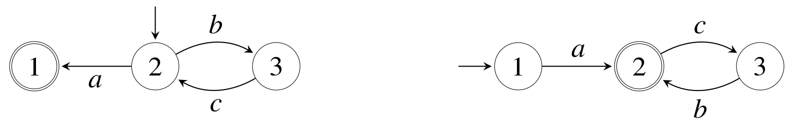


Fig. 6. SPA G (left), and DFA G^R for the reversal of the language $L(G)$ (right); projection $P : \{a,b,c\}^* \to \{a,c\}^*$

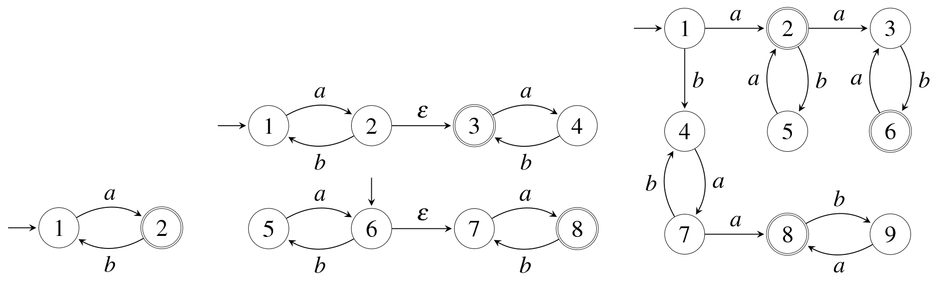


Fig. 7. SPA G (left), NFA for shift$(L(G))$ (middle), and DFA G^{shift} (right); projection $P : \{a,b\}^* \to \{b\}^*$

Left Quotient: Construct a DFA for left quotient by a string w from a DFA G by making the state reached after reading the string w initial.

Consider the DFA G shown in Fig. 8 (left) and the projection P from $\{a,b\}^*$ to $\{b\}^*$. The automaton G is a state-partition automaton with respect to the projection P as in the case of cyclic shift. The automaton $a\backslash G$ for the left quotient by the string a is shown in Fig. 8 (right). It is not a state-partition automaton with respect to the projection P since the sets $\{2\}$ and $\{1,2\}$ are reachable in the projected automaton by strings ε and b, respectively. □

The following theorem demonstrates that if the structure of the automaton is not changed after an operation, then the automaton remains state-partition with respect to the same projection.

Theorem 4. *State-partition automata are closed under the operations of right quotient and complement of complete state-partition automata.*

Proof. Let G be a complete state-partition automaton. Construct a deterministic automaton G^c for the complement of $L(G)$ from the DFA G by interchanging final and non-final states. The result now follows from the fact that the states

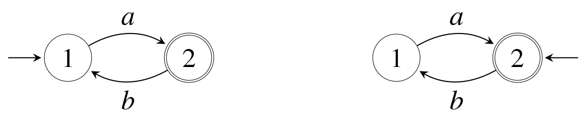


Fig. 8. SPA G (left) and DFA $a\backslash G$ for the left quotient by the string a (right); projection $P : \{a,b\}^* \to \{b\}^*$

of the projected automaton $P(G^c)$ are the same as the states of the projected automaton $P(G)$ since the structure of the automaton G^c is the same as the structure of the automaton G.

Now, consider the right quotient of a language $L(G)$ by a language K; here, the DFA G may be incomplete. Construct an automaton for the right quotient $L(G)/K$ from the automaton G by replacing the set of final states with the set of states of G from which a string of the language K is accepted. Again, the structure of the automaton remains the same; we only change the set of final states. $\square$

5 State-Partition Complexity

Let L be a regular language over an alphabet Σ, and let P be a projection from Σ^* to Σ_o^*. We define the *state-partition complexity* of the language L, denoted by $\operatorname{spc}(L)$, as the smallest number of states in any automaton accepting the language L that is a state-partition automaton with respect to the projection P. By Theorem 2, the state-partition complexity of the language L is the number of states of the DFA $P(G) \parallel G$, where G is the minimal incomplete DFA accepting the language L.

Now, we give the upper bound on the state-partition complexity of regular languages, and prove that this bound is tight. We omit the proof due to space constraints.

Theorem 5. *Let L be a language over an alphabet Σ accepted by the minimal incomplete DFA G with n states. Let P be a projection from Σ^* to Σ_o^*. Then* $\operatorname{spc}(L) \leq 3n \cdot 2^{n-3}$. $\square$

Finally, we prove that the bound proved in the previous theorem is tight.

Theorem 6. *For every integer $n \geq 3$, there exists a regular language L accepted by the minimal incomplete DFA G with n states such that* $\operatorname{spc}(L) = 3n \cdot 2^{n-3}$.

Proof. Consider the language L accepted by the DFA G depicted in Fig. 9 and the projection P from $\{a,b,c\}^*$ to $\{a,b\}^*$. We need to prove that all subsets of the state set $\{0,1,\ldots,n-1\}$, except for the sets that contain $n-1$ and do not contain 0, are states of the automaton $P(G)$. Notice that if X is reachable in $P(G)$ by a string u over $\{a,b\}$ and $q \in X$, then state q is reachable in the automaton G by a string w in $P^{-1}(u)$. This means that (X,q) is a reachable state in the automaton $P(G) \parallel G$ since $(X,q) = \big(\delta(s, P^{-1}(P(w))),\ \delta(s,w)\big)$. First, we construct an NFA accepting the language $P(L)$ as shown in Fig. 10. Let us show that all subsets of the state set $\{0,1,\ldots,n-1\}$ containing state 0, as well as all non-empty subsets of the set $\{1,2,\ldots,n-2\}$ are reachable.

The proof is by induction on the size of subsets. Each set $\{i\}$, where $i \leq n-2$, is reached from $\{0\}$ by the string a^i. Let $2 \leq k \leq n$. Assume that each subset of size $k-1$, satisfying the above mentioned conditions, is reachable. Let $X = \{i_1, i_2, \ldots, i_k\}$, where $0 \leq i_1 < i_2 < \cdots < i_k \leq n-1$, be a subset of size k. Consider two cases:

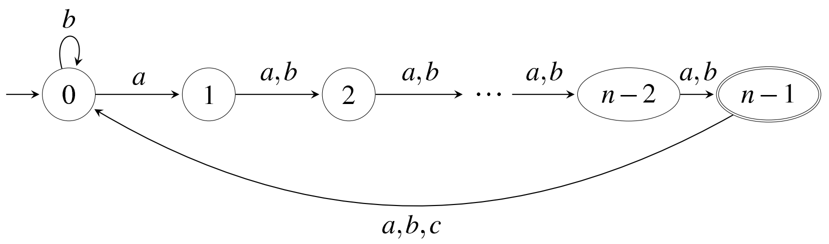


Fig. 9. The minimal incomplete DFA G meeting the upper bound $3n \cdot 2^{n-3}$

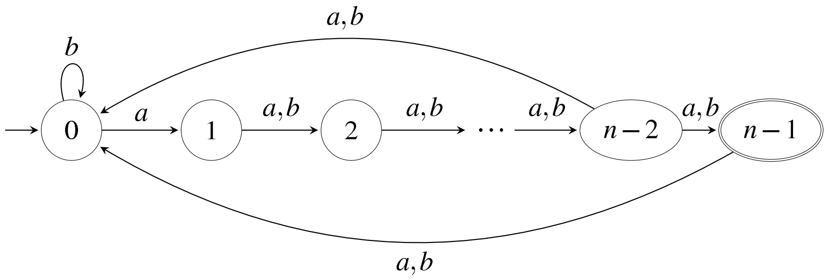


Fig. 10. An NFA for language $P(L(G))$, where G is shown in Fig. 9

(i) $i_1 = 0$. Take $Y = \{i_j - i_2 - 1 \mid 3 \le j \le k\} \cup \{n - 2\}$. Then Y is of size $k-1$ and it does not contain state $n-1$. Therefore, it is reachable by the induction hypothesis. The subset Y goes to X on the string aab^{i_2-1} since we have

$$\begin{aligned} Y &\xrightarrow{a} \{0, n-1\} \cup \{i_j - i_2 \mid 3 \le j \le k\} \\ &\xrightarrow{a} \{0, 1\} \cup \{i_j - i_2 + 1 \mid 3 \le j \le k\} \\ &\xrightarrow{b^{i_2-1}} X. \end{aligned}$$

(ii) $i_1 \ge 1$. Then $i_k \le n-2$. Take $Y = \{0\} \cup \{i_j - i_1 \mid 2 \le j \le k\}$. Then the subset Y is of size k and contains state 0. Therefore, it is reachable as shown in case (i). The subset Y goes to X on the string a^{i_1}.

This proves the reachability of all $3 \cdot 2^{n-2} - 1$ subsets of the automaton $P(G)$.

The number of all reachable pairs (X, q) with $q \in X$ of the automaton $P(G) \parallel G$ is $\sum_{i=0}^{n-1} \binom{n-1}{i}(i+1) + \sum_{i=0}^{n-2} \binom{n-2}{i} i = 3n \cdot 2^{n-3}$, which proves the theorem. $\square$

6 Conclusions and Discussion

We investigated deterministic state-partition automata with respect to a given projection. The state set of such an automaton is partitioned into disjoint subsets that are reachable in the projected automaton. Using a result from the literature that every regular language has a state-partition automaton with respect to a given projection, we provided the construction of the minimal state-partition automaton for a regular language and a projection. We also described a regular language and two projections such that no automaton accepting this language is a state-partition automaton with respect to both projections.

Next, we studied closure properties of state-partition automata under the standard constructions of deterministic automata for the operations of complement, union, intersection, concatenation, star, reversal, cyclic shift, and left and right quotients. We showed that except for the right quotient and complement of complete deterministic automata, all other constructions fail to preserve the property of being a state-partition automaton.

Finally, we defined the notion of the state-partition complexity of a regular language as the smallest number of states of any state-partition automaton with respect to a given projection accepting the language. We proved that the tight bound on the state-partition complexity of a language represented by an incomplete deterministic automaton with n states is $3n \cdot 2^{n-3}$. To prove the tightness of this bound, we used a language defined over the ternary alphabet $\{a, b, c\}$ and the projection from $\{a, b, c\}^*$ to $\{a, b\}^*$. Note that it follows from the results of [6] that this bound cannot be reached using a smaller alphabet or a projection to a singleton.

State-partition complexity of regular operations may be investigated in the future. We only know that state-partition complexity of a language and its complement differs by one in the case of complete deterministic automata, and by $3n$ if the automata are incomplete. Defining nondeterministic state-partition automata and investigating their properties may also be of interest.

Acknowledgements. We wish to thank Jan Komenda, Klaus Schmidt, and Jan H. van Schuppen for a discussion on state-partition automata.

References

1. Boutin, O., Komenda, J., Masopust, T., Schmidt, K., van Schuppen, J.H.: Hierarchical control with partial observations: Sufficient conditions. In: Proc. of IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011), Orlando, Florida, USA, pp. 1817–1822 (2011)
2. Cassandras, C.G., Lafortune, S.: Introduction to discrete event systems, 2nd edn. Springer (2008)
3. Cho, H., Marcus, S.I.: On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. Mathematics of Control, Signals, and Systems 2, 47–69 (1989)
4. Cho, H., Marcus, S.I.: Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. Theory of Computing Systems 22(1), 177–211 (1989)
5. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Boston (2003)
6. Jirásková, G., Masopust, T.: On a structural property in the state complexity of projected regular languages. Theoretical Computer Science 449C, 93–105 (2012)

7. Jirásková, G., Okhotin, A.: State complexity of cyclic shift. RAIRO – Theoretical Informatics and Applications 42(2), 335–360 (2008)
8. Komenda, J., Masopust, T., Schmidt, K., van Schuppen, J.H.: Personal communication (2011)
9. Komenda, J., Masopust, T., van Schuppen, J.H.: Synthesis of controllable and normal sublanguages for discrete-event systems using a coordinator. Systems & Control Letters 60(7), 492–502 (2011)
10. Komenda, J., Masopust, T., van Schuppen, J.H.: Supervisory control synthesis of discrete-event systems using a coordination scheme. Automatica 48(2), 247–254 (2012)
11. Komenda, J., van Schuppen, J.H.: Supremal normal sublanguages of large distributed discrete-event systems. In: Proc. of International Workshop on Discrete Event Systems (WODES 2004), Reims, France, pp. 73–78 (2004)
12. Komenda, J., van Schuppen, J.H.: Modular control of discrete-event systems with coalgebra. IEEE Transactions on Automatic Control 53(2), 447–460 (2008)
13. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 114–125 (1959)
14. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages, vol. 1–3. Springer (1997)
15. Sakarovitch, J.: A construction on finite automata that has remained hidden. Theoretical Computer Science 204(1-2), 205–231 (1998)
16. Salomaa, A.: Formal languages. Academic Press, New York (1973)
17. Schmidt, K., Breindl, C.: Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Transactions on Automatic Control 56(4), 723–737 (2011)
18. Schmidt, K., Moor, T., Perk, S.: Nonblocking hierarchical control of decentralized discrete event systems. IEEE Transactions on Automatic Control 53(10), 2252–2265 (2008)
19. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston (1997)
20. Wong, K.: On the complexity of projections of discrete-event systems. In: Proc. of Workshop on Discrete Event Systems (WODES 1998), Cagliari, Italy, pp. 201–206 (1998)
21. Wong, K.C., Wonham, W.M.: Hierarchical control of discrete-event systems. Discrete Event Dynamic Systems: Theory and Applications 6(3), 241–273 (1996)
22. Wonham, W.M.: Supervisory control of discrete-event systems, lecture notes, University of Toronto (2011) (Online), `http://www.control.utoronto.ca/DES/`
23. Yu, S.: Regular languages. In: Handbook of Formal Languages, vol. I, pp. 41–110. Springer (1997)
24. Zhong, H., Wonham, W.M.: On the consistency of hierarchical supervision in discrete-event systems. IEEE Transactions on Automatic Control 35(10), 1125–1134 (1990)

On Union-Free and Deterministic Union-Free Languages

Galina Jirásková[1,⋆] and Benedek Nagy[2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences,
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

[2] Department of Computer Science, Faculty of Informatics,
University of Debrecen, Debrecen, Hungary
nbenedek@inf.unideb.hu

Abstract. The paper continues the study of union-free and deterministic union-free languages. In contrast with the fact that every regular language can be described as a finite union of union-free languages, we show that the finite unions of deterministic union-free languages define a proper subfamily of regular languages. Then we examine the properties of this subfamily.

1 Introduction

The regular languages are the most common, well-known, and well-applicable languages. They are the simplest languages in the Chomsky-hierarchy, and can be represented by regular expressions or finite automata. Nowadays, some subfamilies of the regular languages have become important in various fields [4–6]. One of them is the family of *union-free languages*: those languages described by regular expressions without the union operation.

Since the Parikh images of regular languages coincide with semi-linear sets, it is an interesting question if the Parikh images of languages in a subregular class still contain every semi-linear set. We address this question in Section 3, and show that only certain special semi-linear sets — the so-called *conditional-linear sets* — can be obtained by Parikh images of union-free languages.

The union-free languages are accepted by special nondeterministic finite automata, the so called *one-cycle-free-path automata*, in which there is exactly one cycle-free path from each state to the final state [10]. The deterministic versions of one-cycle-free-path automata are not as powerful, and define the class of deterministic union-free languages [8].

⋆ Research supported by the Slovak Research and Development Agency under contract APVV-0035-10 "Algorithms, Automata, and Discrete Data Structures".

⋆⋆ Research supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

One of the most important results on union-free languages states that every regular language can be expressed as a union of a finite number of union-free languages [9]. Moreover, the minimal number of required union-free languages is algorithmically computable [3].

Motivated by these results, we address an analogous question for deterministic union-free languages, and provide a negative answer by describing a regular language which cannot be expressed as a union of a finite number of deterministic union-free languages. Our proof uses the representation of deterministic union-free languages by so-called balloon automata.

We also define the classes dU_n as the classes of regular languages that can be described as a union of n deterministic union-free languages, and show that they define a proper hierarchy.

Finally, we consider the class dU_* as the union of classes dU_n, and conclude the paper with some closure properties.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of formal languages and automata theory. For all unexplained notions, we refer the reader to [7, 11].

For a finite non-empty set of symbols Σ, called an alphabet, Σ^* denotes the set of all strings over Σ including the empty string ε. A *language* over Σ is any subset of Σ^*. We denote the size of a finite set A by $|A|$ and its powerset by 2^A.

A *regular expression* over an alphabet Σ is defined inductively as follows: $\emptyset$, ε, and a, for a in Σ, are regular expressions. If r and t are regular expressions, then also $(r + t)$, $(r \cdot t)$, and $(r)^*$ are regular expressions. A regular expression is *union-free* if no symbol + occurs in it. A regular language is *union-free* if there exists a union-free regular expression describing the language.

A *nondeterministic finite automaton* (nfa) is a quintuple $M = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, s is the initial state, F is the set of accepting states, and δ is the transition function that maps $Q \times (\Sigma \cup \{\varepsilon\})$ into 2^Q. The transition function is naturally extended to the domain $Q \times \Sigma^*$. The language *accepted* by nfa M is $L(M) = \{w \in \Sigma^* \mid \delta(s, w) \cap F \neq \emptyset\}$.

Automaton M is *deterministic* (dfa) if it has no ε-transitions, and $|\delta(q, a)| \leq 1$ for all states q in Q and symbols a in Σ. Hence, we consider incomplete dfa's, in which some transitions may be undefined.

A path from state p to state q in an nfa/dfa M is a sequence $p_0 a_1 p_1 a_2 \cdots a_n p_n$, where $p_0 = p$, $p_n = q$, and $p_i \in \delta(p_{i-1}, a_i)$ for $i = 1, 2, \ldots, n$. The length of this path is n; note that p_0 is a path of length 0. The path is a *cycle* if $n > 0$ and $p_0 = p_n$. The path is called a *cycle-free accepting path* if p_n is an accepting state, and either the path is of length 0, or $p_i \neq p_j$ whenever $i \neq j$. An nfa/dfa is a *one-cycle-free-path* (1cfp) nfa/dfa if there is a unique accepting cycle-free path from each of its states. It is known that the family of union-free languages coincides with the set of languages accepted by 1cfp nfa's [10]. The 1cfp dfa's define the family of *deterministic union-free* languages [8].

The accepting path from the initial state of a 1cfp automaton defines the *backbone* of the automaton, and the string accepted by this path is called the *backbone string.* The backbone string is contained in each string of the accepted language in a scattered way, and therefore the shortest string is *unique* in every union-free language [10].

3 On Parikh Images of Union-Free Languages

The Parikh image of a string w over an ordered alphabet $\{a_1, \dots, a_k\}$ is the vector $\Psi(w) = (m_1, \dots, m_k)$ of non-negative integers such that m_i is the number of occurrences of a_i in w. The Parikh image of a language L is the set of vectors $\Psi(L) = \{\Psi(w) \mid w \in L\}$.

A set of the form $\{\alpha_0 + n_1\alpha_1 + \cdots + n_m\alpha_m \mid n_j \geq 0 \text{ for } j = 1, 2, \dots, m\}$, where $\alpha_0, \alpha_1, \dots, \alpha_m$ are vectors of non-negative integers, is said to be a *linear* set. A *semilinear* set is a finite union of linear sets. It is well-known that the Parikh images of regular languages coincide with semilinear sets.

The aim of this section is to analyse the Parikh images of union-free languages. We show that the Parikh images of union-free languages are somewhere between linear and semilinear sets. Then we define the so-called conditional-linear sets, and prove that such sets coincide with the Parikh image of union-free languages.

Lemma 1. *Every linear set is the Parikh image of a union-free language.*

Proof. Let a linear set be given by the vectors α_j for $j = 0, 1, \dots, m$. For each vector α_j, consider a string w_j with Parikh image α_j. First, construct an nfa accepting the singleton set $\{w_0\}$. Now, in the final state, add m disjoint cycles labelled by strings w_j for $j = 1, \dots, m$. This results in a one-cycle-free-path nfa accepting a union-free language with desired Parikh image. □

We have seen that the set of Parikh images of union-free languages is a superset of linear sets, now we give 'upper and lower bounds':

Lemma 2. *There exists a union-free language whose Parikh image is not linear.*

Proof. Consider the language given by the regular expression $a(bb(aab)^*ba)^*a$. Its Parikh image is $\{(2,0)+n(1,3) \mid n \in \mathbb{N}\} \cup \{(3,3)+m(1,3)+k(2,1) \mid m, k \in \mathbb{N}\}$. This set cannot be described by only one linear set. Moreover, this language is even deterministic union-free. □

Lemma 3. *There exists a semilinear set W such that there is no union-free language L with $\Psi(L) = W$.*

Proof. Consider the semilinear set $W = \{(3,1),(4,0)\}$. Every language with Parikh image W is finite and contains at least two shortest strings. □

Lemma 4. *Let a language be accepted by a one-cycle-free-path nfa that consists of a backbone and several, up to starting state, disjoint cycles starting and ending in a state of the backbone; each cycle contains just one state of the backbone. Then the Parikh image of the language is a linear set.*

Proof. The Parikh image of the language is a linear set given by the Parikh vector of the backbone, and the Parikh vectors of the labels of all the cycles. □

The previous result can be stated as follows: The Parikh image of a union-free language with star-height 1 is linear. Now we define conditional-linear sets and show that they coincide exactly with Parikh images of union-free languages.

Definition 1. *A set of vectors W is conditional-linear if every vector α is in W if and only if it can be written in the form*

$$\alpha = \alpha_0 + \delta_1 n_1 \alpha_1 + \delta_2 n_2 \alpha_i + \cdots + \delta_m n_m \alpha_m,$$

where n_j are non-negative integers and α_j are fixed vectors of non-negative integers, and δ_i are conditional coefficients defined in the following way: $\delta_1 = 1$, and if $i > 1$, then δ_i is either without any condition and equals 1, or depends on the coefficient of some α_j with $j < i$, and in such a case it is equal to 1 if $\delta_j n_j > 0$ and to 0 if $\delta_j n_j = 0$:

$$\delta_i = 1, \qquad \textit{if there is no condition for } \alpha_i,$$

$$\delta_i = \begin{cases} 1, & \textit{if } \delta_j n_j > 0, \\ 0, & \textit{if } \delta_j n_j = 0, \end{cases} \qquad \textit{if } \alpha_i \textit{ depends on the coefficient of } \alpha_j.$$

Having $\delta_i = 1$ for all i without any conditions, the linear sets can be obtained. Thus conditional-linear sets are a kind of extension of linear sets. Moreover, all conditional linear sets are semilinear; however not every semilinear set is conditional linear. In a conditional-linear set some vectors α_i have conditions, i.e., they can be present only if another vector with a smaller index is present.

Theorem 1. *Conditional-linear sets coincide with the Parikh images of union-free languages.*

Proof. Let a union-free language be given by a regular expression

$$x_1(r_1)^* x_2 \cdots x_n(r_n)^* x_{n+1},$$

where the parts x_i are star-free; it is allowed that $x_i = \varepsilon$ for some i. Let α_0 be the Parikh vector of $x_1 x_2 \cdots x_{n+1}$, that is, of the backbone string; these symbols are not under any Kleene-star in the tree for the expression. Next, let α_i be the Parikh image of the string obtained from the child-subexpression r_i by substituting ε for the parts under Kleene-star inside. These symbols are below exactly one Kleene-star in the tree. The obtained vectors α_i are without conditions, because their mother expression is the original expression, so let $\delta_1 = \cdots = \delta_n = 1$.

Now, for any subexpression r^* which has not been considered yet, using the order of subexpressions as they are in the tree form of the regular expression starting from the top, let the next vector α_i be the Parikh image of the symbols under this Kleene-star using ε for its child-subexpressions of the form p^*. Let the conditional coefficient δ_i depend on the coefficient of the vector α_j where j is the index of the mother subexpression. Since the union-free expression contains finitely many stars this procedure terminates after finitely many steps.

Next, let a conditional-linear set W be given by its vectors and δ's. Construct a one-cycle-free-path nfa with Parikh image W as follows. Let A be the automaton accepting only a backbone string with Parikh vector α_0. Consider every vector α_i which is without condition, that is, $\delta_i = 1$ independently on any other values. Add a cycle labelled by a string with Parikh vector α_i starting and ending at the same state of the backbone of A.

Then consider each other vector α_i in order of increasing i. Let the condition for this vector depend on the coefficient of a vector α_j. The cycle for vector α_j is already included in the automaton, since $j < i$. Add new states to form a subcycle labelled by a string with Parikh vector α_i in a state of the cycle for α_j; it always can be done by adding new states and ε-transitions to the cycle for α_j. Finally, after adding subcycles for all the vectors, we get a one-cycle-free-path nfa that accepts a language with Parikh image W. □

4 Representation of Deterministic Union-Free Languages

Let us turn our attention to deterministic union-free languages defined as languages accepted by deterministic one-cycle-free-path dfa's [8].

First we fix further terms used in the paper. A state p of a 1cfp automaton is a *branching state* if at least two transitions are defined from state p, thus if $\delta(p, a) = q_1$ and $\delta(p, b) = q_2$, where q_1, q_2 are states of the automaton and a, b are symbols in the input alphabet such that $a \neq b$. Since the automaton is a 1cfp automaton, we must have $q_1 \neq q_2$. The accepting state is a *branching state* if there is at least one transition defined from it. If p is not the accepting state, then exactly one transition from p lies on the cycle-free accepting path from state p. All the other transition defined in state p start different cycles. All the transitions in the accepting state start new cycles.

Let a path paq contain a transition (p, a, q) starting a cycle. In the case $p = q$, the cycle has length 1. Otherwise, let us consider the cycle-free accepting path from state q. This path must contain state p, thus the path is of the form qz_1pz_2, where $z_1 \in \Sigma(Q\Sigma)^*$ and $z_2 \in (\Sigma Q)^*$. The cycle $paqz_1p$ is a *starting cycle* at state p. As an example, consider the cycles $4a6b4$ and $4c2a3a4$ that are starting cycles at state 4 in Fig. 2 (left).

Cycles may only start at branching states. The state where a cycle contains a previously (surely) visited state is the *returning state* of the cycle. A cycle may return to a previously (surely) visited state in various ways: A cycle may return at the same state as it starts like the cycle $4a6b4$ in Fig. 2 (left), or a

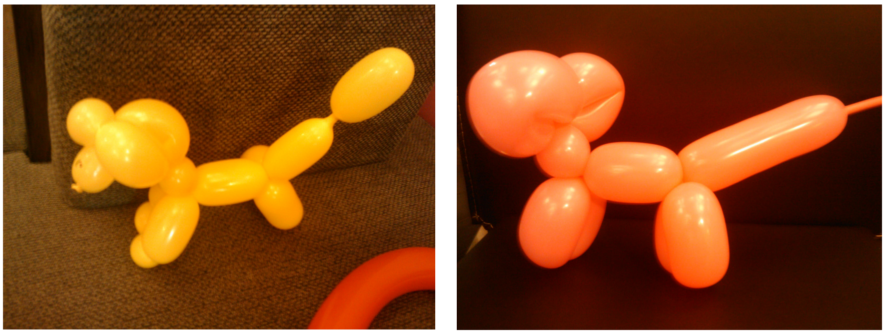


Fig. 1. A toydog from long balloon

cycle may return to another state like the cycle $2b0a1a2$ in Fig. 2; here state 0 is the returning state.

Since a one-cycle-free-path dfa over a unary alphabet cannot have a branching state, except the accepting state, the automaton may only have one cycle starting at the accepting state. Consequently, a deterministic union-free language over a unary alphabet either contains at most one string or is of the form $a^m(a^n)^*$ for some positive integers m, n. The other direction also holds: All these languages are accepted by 1cfp dfa's. Therefore, the unary case seems to be not so interesting. In what follows, we always assume that an alphabet has at least two symbols.

There can be several one-cycle-free-path dfa's for the same language. One of them plays an important role for us. We call it a *balloon* automaton. This name comes from the toys made by clowns for children from long balloons; for example, toydog (see Fig. 1).

Now we give the definition of a balloon automaton. We use balloon automata later to get regular expressions for deterministic union-free languages.

Definition 2 (Balloon DFA). *A backbone 1cfp dfa, that is, a dfa consisting of states of the backbone connected through the symbols of the backbone string is a balloon dfa. If A is a balloon automaton, then any extension of A obtained in the following way is a balloon automaton:*

– *pick a state p of A and a string $a_1 \cdots a_k$ of length k with $k \geq 1$ such that there is no transition on a_1 from p in A;*
– *add $k-1$ new states $p_1, \ldots, p_{k-1}$ to A connected through transitions*

$$p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} p_{k-1} \xrightarrow{a_k} p.$$

Thus a deterministic one-cycle-free-path automaton is called a balloon dfa, if every starting cycle returns at the same (branching) state as it starts. Fig. 2 shows a one-cycle-free-path dfa and the corresponding balloon automaton.

The following result helps us to characterize deterministic union-free languages by regular expressions.

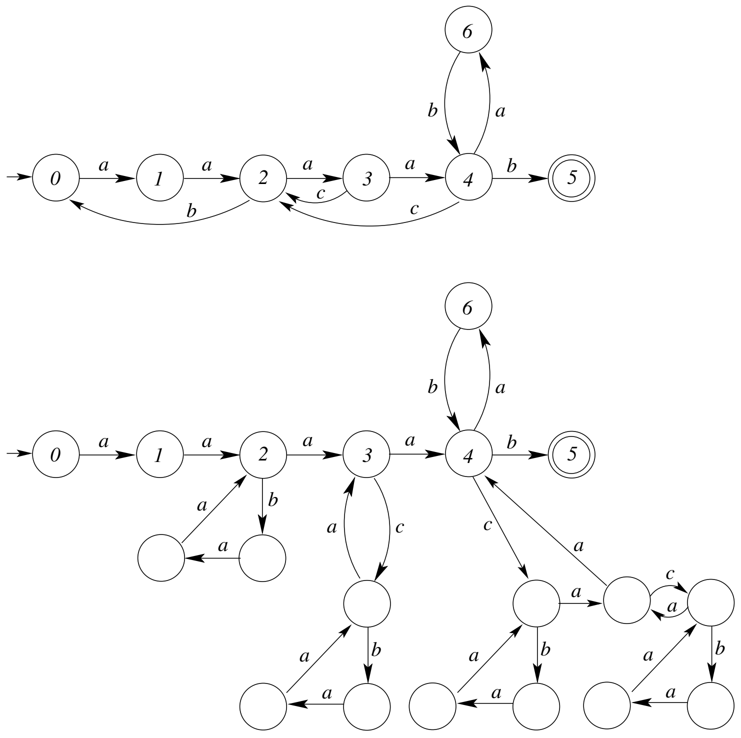


Fig. 2. A 1cfp dfa (up) and the corresponding balloon automaton (down)

Theorem 2. *Every deterministic union-free language is accepted by a balloon dfa.*

Proof. Let a deterministic union-free language L be given by a 1cfp dfa A. We are going to construct a regular expression and a balloon automaton for L in parallel from 1cfp dfa A. We use mixed-form expressions that contain symbols and names of the states of the automaton alternatively. We start with the expression $r_0 = sx_1q_1x_2 \cdots q_{m-1}x_mf$ that contains the symbols of the backbone string $x_1x_2 \cdots x_m$ and the names of the states $s, q_1, \ldots, q_{m-1}, f$ that occur on the backbone. All the states in expression r_0 are pairwise distinct.

As an example, consider the 1cfp dfa shown in Fig. 2. In this example, we have $r_0 = 0a1a2a3a4b5$.

The construction of the balloon automaton starts with the deterministic automaton A_0 that accepts only the backbone string: accordingly, it has states $s', q'_1, \ldots, q'_{m-1}, f'$.

We continue recursively while the automaton has branching states, that is, states that go to at least two distinct states by some symbols.

If the initial automaton A has no branching state, then it only accepts the backbone string, and it is a balloon automaton.

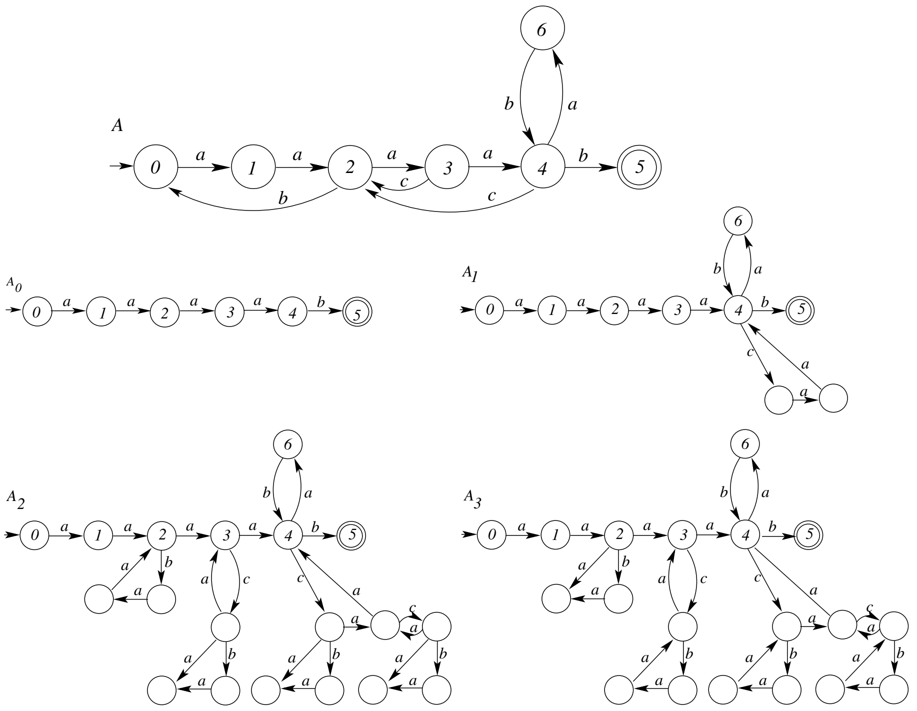


Fig. 3. The construction of a balloon automaton

Assume that our current expression is r_i, and our current automaton is A_i. Construct expression r_{i+1} by modifying expression r_i as follows. Choose the last branching state p that has not been considered yet, that is, the branching state that has the last occurrence in the mixed form r_i among the not yet considered mixed states. Put a pair of brackets into expression r_i to the points immediately after each occurrence of p. Put a star after the brackets, that is, use the following form:

$$u_1p()^*u_2\cdots u_mp()^*u_{m+1},$$

where p has m occurrences in r_i, and u_j's do not contain any p. Then put into these brackets as many sequences $()^*$ as many starting cycles exist at state p. After this, write into these brackets the mixed form expressions for the starting cycles using all the starting branches at state p without symbol p. The resulting expression is r_{i+1}. If only one cycle starts from a branching state, then the form $((z)^*)^*$ can be simplified to $(z)^*$, where z denotes the mixed form representing the cycle. In our example, we have

$$r_1 = 0a1a2a3a4((a6b)^*(c2a3a)^*)^*b5,$$
$$r_2 = 0a1a2a3(c2a)^*a4((a6b)^*(c2a3(c2a)^*a)^*)^*b5,$$
$$r_3 = 0a1a2(b0a1a)^*a3(c2(b0a1a)^*a)^*a4((a6b)^*(c2(b0a1a)^*a3(c2(b0a1a)^*a)^*a)^*)^*b5.$$

Notice that after each step, the number of occurrences of p remains m; and all the states which occur in, say, k cycles starting from p have km new occurrences. Moreover, since the automaton is a 1cfp dfa, there are no states appearing in the new brackets which have already been considered.

Now construct automaton A_{i+1} corresponding to expression r_{i+1} by modifying automaton A_i. Extend automaton A_i by cycles corresponding to the new subexpressions $(z)^*$; the ones that are not in r_i, but which are in r_{i+1}. For each of these new subexpressions, add a new cycle to every copy of state p' starting and ending in this state (in a similar way as we draw the backbone). In this step, we use various copies of the original states. Fig. 3 shows automata A_i in our example.

Repeat the above procedure until no more branching states exist. Since there are only finitely many branching states in the given automaton, this procedure terminates after finitely many steps. Let r_n be the resulting expression. After deleting the names of the states in r_n, we get a union-free expression which describes language L. The resulting automaton A_n accepts language L. Moreover, after each step, automaton A_i is a balloon automaton. □

The proof of the previous theorem gives also a regular expression for every one-cycle-free-path dfa. By this construction, we can characterize the deterministic union-free languages by regular expressions as follows. Every deterministic union-free language can be expressed by a regular expression of the following form:

(i) there is at most 1 symbol that continues the expression "in the same level";
(ii) the other symbols may enter for brackets: if there is one such symbol, then we use b: $(b\cdots)^*$; if there are two or more symbols, then we use $b_1, b_2, \ldots, b_\ell$: $((b_1\cdots)^*(b_2\cdots)^*\cdots(b_\ell\cdots)^*)^*$.

On the other hand, every regular expression that satisfies (i) and (ii) describes a deterministic union-free language since a 1cfp dfa for such a language can be constructed by our proof.

We conjecture that the balloon dfa has maximal number of states among all the incomplete dfa's that accept the language and for each of its state there is exactly one accepting cycle free path.

A rough idea for the proof of the conjecture could be the following: By the construction of the balloon automata for a 1cfp dfa, the number of its states is not smaller than the number of the states of the original automaton. Moreover, the balloon automaton cannot be extended further by states. Since it is deterministic, by adding a new state, some new transitions are needed. This modifies the backbone or a cycle, or creates a new cycle; thus modifies the accepted language.

One of the most important complexity measures of regular languages is the star height that is connected to the number of nested stars in the regular expressions [11]. To measure the complexity of balloon automata, we define the following concept.

Definition 3 (Cycle Depth of Balloon Automata). *The cycle depth of a balloon automaton is the maximal number of its nested cycles.*

The cycle depth of a balloon automaton is 0 for cycle-free automata. For example, the depth of the automaton A_0 in Figure 3 is 0. Such automata only accept singleton languages. The cycle depth of A_1, A_2, A_3 is 1, 2 and 3, respectively.

The cycle depth of the balloon automata and the (nested) star-height of the obtained regular expressions have a strong relation: Actually, if expressions of the form $((r_1)^*(r_2)^*)^*$ are rewritten of the form $(r_1 + r_2)^*$, then this new expression has the same star height as the cycle depth of the original balloon automaton.

A cycle pzp of a 1cfp dfa, where $z \in \Sigma(Q\Sigma)^*$, is called an *inner cycle* if there is no branching state in z. For example, the cycle $2b0a1a2$ in Figure 2 is an inner cycle. In balloon automata, these cycles are at the deepest level. For example, the cycles of length 2 of A_2 in Figure 3 are inner cycles.

We have some important observations about graphs of 1cfp deterministic automata, in particular, we formulate them for balloon automata.

Lemma 5. *Let $A = (Q, \Sigma, \delta, s, f)$ be a 1cfp dfa accepting a non-empty language. Then*

1. *The branching factor at each state of A is at most $|\Sigma|$.*
2. *If the cycle depth of a balloon automaton is zero, then there is a state, in which at most one symbol in Σ defines a transition; the transition on the other symbols are undefined.*
3. *If the cycle depth of a balloon automaton is at least one, and the length of the/an inner cycle is at least two, then there is a state where only one of the symbols defines a transition.*

Proof. 1. In every state of a dfa, at most $|\Sigma|$ transitions may be defined.

2. Balloon automata with cycle depth zero accept exactly one string, and all the transitions in the final state f are undefined.

3. If the cycle depth of a balloon automaton is one, then there is a cycle starting from a state on the backbone with, by our assumption, length at least two. This means that after the first transition of this cycle, there is a state that is outside of the backbone. In this inner state, exactly one transition is defined.

Now assume that the cycle depth of a balloon automaton is at least two. The argument is quite similar to the previous case. In a balloon automaton, one transition could go to the direction of the final state; the first symbol of the unique cycle-free accepting path gives this transition. All the other symbols may start a new cycle going more deeply in the cycle depth of the automaton. Consider the/an inner cycle; its length is at least two by the assumption of the lemma. In this cycle, there is no new starting cycle. Therefore, in the/an inner state, there exists only one transition, which starts the cycle-free accepting path from that state. □

We conclude this section with the following result showing that the class of deterministic union-free languages is not closed under basic regular operations.

Theorem 3 (Closure properties). *The class of deterministic union-free languages is not closed under boolean operations, concatenation, square, star, reversal, cyclic shift, homomorphism, and inverse morphism.*

Proof. For each operation, we present deterministic union-free languages such that the language resulting from the operation is not deterministic union-free. Recall that if a language has at least two shortest strings, or if its minimal dfa has at least two final states, then the language is not deterministic union-free.

Complement:	$\{\varepsilon\}^c$	$= \Sigma^+$,
Union:	$\{a\} \cup \{b\}$	$= \{a, b\}$,
Intersection:	$b^*ab^* \cap a^*ba^*$	$\subseteq \{ab, ba\} \cup \{a, b\}^{\geq 3}$
Symmetric difference:	$\{a\} \oplus \{b\}$	$= \{a, b\}$,
Cyclic shift:	$\text{Shift}(\{ab\})$	$= \{ab, ba\}$,
Shuffle:	$\{a\} \sqcup\!\sqcup \{b\}$	$= \{ab, ba\}$,
Inverse morphism:	$h^{-1}(\{aa\})$	$= \{aa, ab, ba, bb\}$
$h(a) = h(b) = a$		

For square and concatenation, consider the deterministic union-free language L_1 accepted by the 1cfp dfa shown in Fig. 4 (left). For reversal, consider the deterministic union-free language L_2 accepted by the 1cfp dfa shown in Fig. 4 (middle), and for star, the deterministic union-free language L_3 accepted by the 1cfp dfa shown in Fig. 4 (right). The minimal dfa's for L_1^2, L_2^R, and L_3^* have two final states, and therefore the resulting languages are not deterministic union-free.

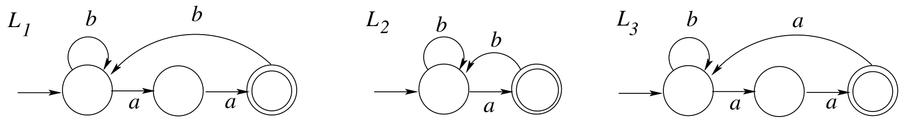


Fig. 4. The 1cfp dfa languages L_1, L_2, L_3 such that L_1^2, L_2^R, and L_3^* are not deterministic union-free

For homomorphisms, consider the deterministic union-free language ab^*ac^* and homomorphism $h(a) = a, h(b) = ab, h(c) = c$. Then $h(ab^*ac^*) = a(ab)^*ac^*$, the minimal dfa for which has two final states. □

5 Finite Union of Deterministic Union-Free Languages

Every regular language can be expressed as the union of a finite number of union-free languages [9]. This is one of the most important results on union-free languages. We can ask whether or not a similar result also holds for *deterministic* union-free languages. The next theorem provides a negative answer, and states one of the main results of this paper.

Theorem 4. *The language L described by the regular expression $((a+b)(a+b))^*$ cannot be expressed as a union of a finite number of deterministic union-free languages.*

Proof. The language L contains exactly the strings over $\{a,b\}$ of an even length. Therefore, each string over $\{a,b\}$ is a prefix of infinitely many strings in L.

Assume for the contradiction that L is given as a finite union of some deterministic union-free languages. Let us consider the balloon automata for these languages. None of these automata has a cycle of length one because otherwise this loop would allow to pump some strings of the accepted language symbol by symbol which would lead to the acceptance of strings of an odd length.

Now let us order all the balloon automata in a list $A_1, A_2, \ldots, A_\ell$. By Lemma 5, each of these automata contains at least one state, in which at most one symbol defines a transition. Moreover, such a state is reachable from every state of a balloon automaton: If there is no cycle at the final state, then the final state is such a state. If there is a cycle starting at the final state, then there is an inner cycle here with at least one such state. And since this state is reachable from the final state, it can be reached from each state of the automaton. Let q_i refer to such a state in automaton A_i.

Let us construct a string w in the following way: Let the prefix of w be the string w_1 that leads to the state q_1 in automaton A_1. Then let the next symbol of w be the one for which there is no transition from q_1. Now let $i = 2$, and hence the next automaton in the list is considered. Let us continue the construction of w by appending w_i to it, where w_i is defined as follows. If the already constructed initial part of w cannot be processed by the automaton A_i, then w_i is the empty string. Elsewhere let q_i' be the state reached by automaton A_i after reading w. Then w_i is the string that leads automaton A_i from state q_i' to state q_i. The next symbol of w is the/a symbol, for which there is no transition from q_i in A_i. Then, we increase i and consider the next automaton in the list until all of them are considered.

Finally, we add one or two symbols to the end of w, depending on length of w; for example, we add a if the length of w is odd, and aa in the other case. The constructed string has an even length, and therefore it is in L. However, our construction proves that no automaton in our list accepts the constructed string, which is a contradiction to our assumption that L is expressed as the union of languages $L(A_i)$. The theorem is proved. □

The proof also works for any language L that has the the following properties:

- the minimal complete dfa for the language has no dead state, that is, each string w in Σ^* is a prefix of a string in L;
- L contains only strings of an even/odd length.

To conclude the paper, let us consider the finite unions of deterministic union-free languages.

Definition 4. *For every positive integer n, we define dU_n as the family of languages that can be expressed as a union of n deterministic union-free languages. Furthermore, let*

$$dU_* = \bigcup_{i=1}^{\infty} dU_i.$$

The following result shows that the classes dU_n define a proper hierarchy.

Theorem 5. *For every n, there exists a language L_n such that $L_n \in dU_n \setminus dU_{n-1}$.*

Proof. Let $L_n = \{a^i b^{n-i} \mid i = 1, 2, \ldots, n\}$ be a language consisting of n strings over $\{a, b\}$ of length n. Then L_n is accepted by the union of n backbone automata. On the other hand, this language cannot be accepted by the union of any $n-1$ deterministic one-cycle-free-path automata because otherwise one of them would accept a language containing at least two shortest strings. □

Now we give some non-closure properties of the language classes dU_n.

Theorem 6. *The classes dU_n are not closed under union, concatenation, square, cyclic shift. If $n \geq 4$, then the class dU_n is not closed under star.*

Proof. For $n = 1$, that is, for the class of deterministic union-free languages, we have already proved these non-closure properties. Otherwise, consider the languages $K = \{b^i \mid 1 \leq i \leq n\}$ and $L = \{a^i b \mid 1 \leq i \leq n\}$ in dU_n. The union of these languages, as well as their concatenation, is a finite language, however, it contains more than n strings. The square of the language L, as well as its cyclic shift, is a finite language containing more than n strings.

The star of the language $aa + ab + ba + bb$ is not in dU_* by Theorem 4, and therefore the last statement of the theorem holds. □

The next theorem gives some closure properties of the class dU_*.

Theorem 7. *The class dU_* is closed under union, and it is not closed under star, intersection, and complement.*

Proof. For star, consider the language $\{\varepsilon, aa, ab, ba, bb\}$ which is in dU_5. After applying the star operation, we get the language $((a+b)(a+b))^*$. By Theorem 4, this language is not in dU_*.

For intersection, consider the languages $K = ((b^*a)^2)^*b^*$ and $L = ((a^*b)^2)^*a^*$, the first of which contains the strings with an even number of a's, while the second one consists of strings with an even number of b's. The proof of Theorem 4 works for the language $K \cap L$ as well since every string in $\{a, b\}$ is a prefix of a string in $K \cap L$, and $K \cap L$ contains only strings of an even length.

Hence the class dU_* is not closed under complement because otherwise, since it is closed under union, it would be closed also under intersection. □

We leave as an open problem whether or not the class dU_* is closed under other regular operations.

6 Conclusions

We examined in detail the classes of union-free and deterministic union-free languages. First we studied the Parikh images of union-free languages, and we proved that they coincide with so-called conditional-linear sets.

Then we defined balloon automata for deterministic union-free languages, and we used them to get regular expressions for deterministic union-free languages, as well as to prove one of the main results of our paper. This result shows that the finite unions of deterministic union-free languages describe a proper subfamily of regular languages. We also investigated the properties of classes represented as a finite union of deterministic union-free languages.

Some closure properties remain open. The characterization of the subregular language class containing the regular languages that cannot be expressed as a finite union of deterministic union-free languages is an interesting further task. Providing minimal 1cfp dfa for deterministic union-free languages, and proving/disproving our conjecture about the maximality of the balloon automata seem to be interesting challenges as well.

References

1. http://www.funstuffordogs.com/Qstore/Qstore.cgi?CMD=011&PROD=1296626935&PNAME=Balloon+Animal+Dog+Toy+-+Dog
2. http://www.facebook.com/pages/Black-Balloon-Shop-baguio-Only/183737488331381
3. Afonin, S., Golomazov, D.: Minimal Union-Free Decompositions of Regular Languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 83–92. Springer, Heidelberg (2009)
4. Dassow, J.: Contextual Grammars with Subregular Choice. Fundamenta Informaticae 64, 109–118 (2005)
5. Dassow, J., Manea, F., Truthe, B.: Networks of Evolutionary Processors with Subregular Filters. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 262–273. Springer, Heidelberg (2011)
6. Bordihn, H., Holzer, M., Kutrib, M.: Henning Bordihn, Markus Holzer, Martin Kutrib: Determination of finite automata accepting subregular languages. Theoretical Computer Science 410(35), 3209–3222 (2009)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley Publishing Company, Reading MA (1979)
8. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. International Journal of Foundations of Computer Science 22, 1639–1653 (2011)
9. Nagy, B.: A normal form for regular expressions. Calude, C.S., Calude, E., Dinnen, M.J. (eds.) Supplemental Papers for DLT 2004, CDMTCS, Auckland (2004)
10. Nagy, B.: Union-free regular languages and 1-cycle-free-path-automata. Publ. Math. Debrecen 68, 183–197 (2006)
11. Yu, S.: Chapter 2: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer, Heidelberg (1997)

A Characterisation of Languages on Infinite Alphabets with Nominal Regular Expressions

Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

Abstract. We give a characterisation of languages on infinite alphabets in a variant of nominal regular expressions with permutations (p-NREs). We also introduce automata with fresh name generations and permutations (fp-automata), inspired by history-dependent automata (HDAs) and fresh-register automata. Noteworthy, permutations require to deal with dynamic context-dependent expressions. Finally, we give a Kleene theorem for p-NREs and fp-automata to formally characterise languages on infinite alphabets.

1 Introduction

The study of languages on infinite alphabets has been pushed by the need of formalising data structures built on top of infinite domains of values [16]. In this context, it is natural to appeal to the theory of automata [11,19,3,22] operating on a countably infinite alphabet $\mathcal{N}$ of *names* to express languages of interest such as $\mathcal{L}_1$ below, see [22],

$$\mathcal{L}_1 = \{n_1 \dots n_k \in \mathcal{N}^* \;\big|\; \forall i \,.\, 1 \le i < k \,.\, n_i \ne n_{i+1}\}$$

In [13], we extended this line of investigations to languages where words do not only consist of names but also of binders (e.g. lambda-calculus terms). In particular, [13] studies a notion of regular expression for words with binders and the associated notion of finite automata, aiming at applications to the design and analysis of programming languages (as in [20] or [18]) or to verification and testing.

Here, we move back to languages of words without binders and apply the techniques of [13] in order to obtain a novel notion of regular expression for languages on infinite alphabets. While being built from concatenation, sum, and Kleene star in the usual way, the nominal regular expressions introduced in this paper may also contain a binder

$$\langle_n \mathsf{ne} \rangle_n^m \tag{1}$$

Intuitively, $\langle_n \mathsf{ne}$ allocates a fresh resource within ne, whereas $\rangle_n^m$ deallocates it. The crucial new ingredient, which allows us to capture for example the language $\mathcal{L}_1$ above, is that $\rangle_n^m$ permutes n and m before deallocating n. For example, we can read

$$\langle_m \; (\langle_n n \rangle_n^m)^* \; \rangle_m^m \tag{2}$$

as follows. First $\langle_m$ allocates a name m, then $\langle_n$ allocates a fresh (i.e. different) name n. Now the permutation specified by $\rangle_n^m$ makes sure that it is the first name m that is deallocated and the second name n that is retained, before looping back to the beginning as specified by the Kleene star *.

Note that permutations have a slightly subtle effect on concatenation. For example, in $\langle_m \langle_n n\rangle_n^m \circ \mathsf{ne}\rangle_m^m$, due to the permutation specified by $\rangle_n^m$, the name m in ne semantically represents n. So it will take some care, in § 3, to describe how nominal regular expressions give rise to nominal regular languages. In § 4 we introduce the corresponding notion of automata with fresh-name generation and permutations (fp-automata). Like HD-automata they have allocation transitions, corresponding to $\langle_n$, but unlike HD-automata permutations only appear in special deallocation transitions corresponding to $\rangle_n^m$. In § 5 we prove a Kleene-type theorem stating that the languages accepted by fp-automata are precisely the nominal regular languages.

Finally, we argue that the work nominal languages with binders also sheds new light on well-established work on languages on infinite alphabets.

Related Work. Apart from languages on infinite alphabets, our research draws on HD-automata [17,14] and nominal sets [9], as well as on the recent confluence of these three areas in [6,22,3,8,4,13]. Moreover, the way we deal with scope and binders is related to nested words [1] and automata working on lambda-calculus terms [21].

Although our paper does neither require nor use the theory of nominal sets, the work on nominal sets does suggest to view automata as (co)algebras in the category of nominal sets. As nominal sets are themselves permutation algebras one would expect regular expressions, as it is the case in our work, to contain permutations, see [15].

A form of regular expressions, called UB-expressions, for languages on infinite alphabets investigated in [10]. There it is shown that UB-expressions are as expressive as finite-state unification based automata (FSUBA), which are somewhat weaker than the finite-memory automata (FMA) of [11]. In a nutshell, FSUBA do not account for freshness (they can, for example, accept the first but not the second language of Example 6 in § 3.2). Moreover, UB-expressions do not have permutations, although their device of labelling the Kleene-star allows them to accept $\mathcal{L}_1$ in a similar fashion to ours.

2 Motivating Examples: Languages on Infinite Alphabets

Languages on infinite alphabets can suitably formalise data structures on infinite domains (take data words in [16,5] as an example). In this context, typical languages consist of finite words — that is finite sequences of symbols — whose symbols are possibly drawn from an infinite set. This is illustrated in the next example.

Example 1 ([22]). Assume that $a, a', a_1, \ldots$ range on an infinite set A. The language

$$\mathcal{L}_1 \overset{\text{def}}{=} \{a_1 \cdots a_k \mid k \geq 0 \wedge \forall i \in \{1, \ldots, k-1\}. a_i \neq a_{i+1}\}$$

consists of finite sequences on A in which all two consecutive letters are different. ◇

In general, the infinite alphabet has a very simple structure that permits to test just for equality or inequality of symbols (see e.g., [16]) as in Example 1 (in recent work this is being reconsidered, see § 6 for a discussion). For practical reasons, it is sometimes convenient to consider words whose letters can be taken either from an infinite alphabet or from a disjoint finite set, as in Example 2.

The next example shows how languages on infinite alphabets could be used to represent computations where resources have to be acquired and released before their usage.

Example 2. Assume that $a, a', a_1, \dots$ range on an infinite set A while α (for "acquire") and ρ ("for release") are two distinguished symbols not in A. The language

$$\mathcal{L}_{\alpha\rho} \stackrel{\text{def}}{=} \alpha A^* \rho \cup \bigcup_{a \neq a'} \{\alpha \underbrace{a \cdots a}_{i \geq 0 \text{ times}} \alpha a_1 \cdots a_j \rho \underbrace{a' \cdots a'}_{h \geq 0 \text{ times}} \rho \mid j \geq 0 \land \forall 1 \leq r \leq j. a_r \in \{a, a'\}\}$$

represents the executions of a process that acquires, uses, and then releases either one or two resources. ◇

3 Nominal Regular Expressions with Permutations

In order to characterise languages on infinite alphabets, we extend regular expressions with name-abstraction and permutation. Let $\mathcal{N}$ be an infinite set of names and $\mathcal{S}$ a finite set of letters. We assume that $\mathcal{N}$ and $\mathcal{S}$ are disjoint. A *language on* $\mathcal{N} \cup \mathcal{S}$ is a subset of $(\mathcal{N} \cup \mathcal{S})^*$, namely a collection of *words on infinite alphabets*.

The nominal regular expressions with permutations (*p-NREs*) are given by

$$\mathsf{ne} ::= \mathit{1} \mid \mathit{0} \mid n \mid s \mid \mathsf{ne} + \mathsf{ne} \mid \mathsf{ne} \circ \mathsf{ne} \mid \mathsf{ne}^* \mid \langle_n \mathsf{ne} \rangle_n^m$$

where $\mathit{1}$ denotes the singleton with the empty word, $\mathit{0}$ denotes the empty language, n ranges over $\mathcal{N}$, s ranges over $\mathcal{S}$; the operators $+$, $\circ$, and $\_^*$ are as the classical operators of regular expressions, while $\langle_n \mathsf{ne} \rangle_n^m$ is a binder. Note that the superscript m on the closing bracket is not bound unless it is the same as the subscript. Closed and bound occurrences of names are defined in the natural way and we call *closed* any p-NRE with no occurrence of free names.

Intuitively, $\langle_n \mathsf{ne}$ allocates a *fresh resource* within ne; as in nominal calculi, this is rendered by declaring a *fresh local name n*. Novel is here that $\rangle_n^m$ specifies a permutation when disposing the resource denoted by n, to the effect that to the right of $\rangle_n^m$ every syntactic occurrence of m is semantically read as n. One can think of n and m as registers whose contents are swapped when n is deallocated. To make this work, we assume that the superscript m on $\rangle_n^m$ is in the syntactic scope of some $\langle_m$. For example, the p-NRE $\langle_n \langle_m nm \rangle_m^n n \rangle_n^n$ is acceptable while $\langle_n \langle_m nm \rangle_m^l m \rangle_n^n$ is not, because the subexpression $\langle_m nm \rangle_m^l$ is not within a scope of a $\langle_l$, and the rightmost m is outside of the scope of $\langle_m$.

We aim to contribute to a foundational theory of interactions based on nominal calculi. This requires to consider interactions with an *environment* that can be seen as a resource handler for requiring and releasing resources. In this respect, the role of the execution environment can be suitably represented using *contexts*. In our theory, contexts capture two fundamental notions: one notion is *the fresh name generation* (read the environment) and the other is *the permutation action* (change the environment). To generate a fresh local name, we have to know which names are already in the environment. And, to leave the permutation result, we must update the environment which could be different from the original environment.

Our contexts are finite lists of names in $\mathcal{N}$. For a p-NRE ne, we call a triple $L \ddagger \mathsf{ne} \ddagger R$ an *expression-in-contexts*, where L and R are respectively called *pre-* and *post-context*.

Intuitively, $L \ddagger \mathsf{ne} \ddagger R$ is an expression ne that is interpreted in the pre-context L and modifies it to the post-context R.

3.1 Preliminaries

Given a function f, the update $f_{[a \mapsto b]}$ extends $dom(f)$ to $dom(f) \cup \{a\}$ with $f(a) = b$; $\perp$ is the empty map. Let L, M, ... range over lists of $\mathcal{N}$ and let $lth(L)$ be the length of the list L. The empty list is denoted as $[]$. For $n \in \mathcal{N}$, write $n\#L$ (read "n is fresh for L,") when $n \neq l$ for any element l in L and write $L@n$ to be the list appending n to the tail of L. We consider only lists with no repeated elements. Given a list L we may abuse the notation and denote its underlying set by L.

The transposition of n and m, denoted by $(m\,n)$, is the permutation that swaps m and n and is the identity on any other names. Given two lists of the same length k, say $N = [n_1, \dots, n_k]$ and $M = [m_1, \dots, m_k]$, let $N \rhd M$ be the bijection from N to M such that

$$N \rhd M : n_i \mapsto m_i \qquad \text{for each } i \in \{1, \dots, k\}$$

and define the bijection $\pi_{[N \rhd M]}$ on $\mathcal{N}$ as

$$\pi_{[N \rhd M]}(x) \overset{\text{def}}{=} \begin{cases} N \rhd M(x), & \text{if } x \in N \\ N \lessdot M(x), & \text{if } x \in M \setminus N \\ x, & \text{if otherwise} \end{cases}$$

where $N \lessdot M(x)$ is a function from $M \setminus N$ to $N \setminus M$ recursively defined as follows:

$$N \lessdot M(m_i) \overset{\text{def}}{=} \begin{cases} (N \rhd M)^{-1}(m_i), & \text{if } (N \rhd M)^{-1}(m_i) \notin N \setminus M \\ N \lessdot M(m_j), & \text{if } (N \rhd M)^{-1}(m_i) = m_j \text{ for some} j \neq i \end{cases}$$

For example, if $M = [b, c, d]$ and $N = [a, b, c]$, we have $\pi_{[N \rhd M]}$ as follows:

$$\begin{tikzcd}
\mathcal{N} \arrow[d, "\pi_{[N \rhd M]}"] & \cdots & x \arrow[d, mapsto] & \cdots & a \arrow[d, mapsto] & b \arrow[d, mapsto] \arrow[dl, dashed, no head] & c \arrow[d, mapsto] \arrow[dl, dashed, no head] & d \arrow[d, mapsto] \\
\mathcal{N} & \cdots & x & \cdots & b & c & d & a
\end{tikzcd}$$

where the target of d (defined by $N \lessdot M$) is traced by going backwards along $\longmapsto$ and the dashed lines.

We define the action of a permutation π on p-NREs and on lists as follows. For a p-NRE ne, the permutation action of π on a p-NRE ne, denoted as $\pi \cdot \mathsf{ne}$, is

1. $\pi \cdot \mathit{1} = \mathit{1}$; $\quad \pi \cdot \mathit{0} = \mathit{0}$; $\quad \pi \cdot n = \pi(n)$; $\quad \pi \cdot s = s$
2. $\pi \cdot (\mathsf{ne}_1 + \mathsf{ne}_2) = (\pi \cdot \mathsf{ne}_1) + (\pi \cdot \mathsf{ne}_2)$
3. $\pi \cdot (\mathsf{ne}_1 \circ \mathsf{ne}_2) = (\pi \cdot \mathsf{ne}_1) \circ (\pi \cdot \mathsf{ne}_2)$
4. $\pi \cdot (\mathsf{ne}^*) = (\pi \cdot \mathsf{ne})^*$
5. $\pi \cdot (\langle_n \mathsf{ne} \rangle_n^m) = \langle_{\pi(n)} (\pi \cdot \mathsf{ne}) \rangle_{\pi(n)}^{\pi(m)}$

while, the permutation action of π on $L = [l_1, \dots, l_k]$ is $\pi \cdot L = [\pi(l_1), \dots, \pi(l_k)]$.

3.2 From p-NREs to Languages on Infinite Alphabets

The interpretations of p-NREs depend on pre- and post-contexts, therefore we introduce the set of rules in Fig. 1.

$$\frac{L \ddagger \mathsf{ne}_1 + \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_1 \ddagger R}\,(\hat{+}_1) \qquad \frac{L \ddagger \mathsf{ne}_1 + \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_2 \ddagger R}\,(\hat{+}_2)$$

$$\frac{L \ddagger \mathsf{ne}_1 \circ \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_1 \ddagger L \qquad L \ddagger \mathsf{ne}_2 \ddagger R}\,(\hat{\circ}) \qquad \frac{L \ddagger \mathsf{ne}^* \ddagger R}{L \ddagger \underbrace{\mathsf{ne} \circ \cdots \circ \mathsf{ne}}_{k \text{ times}} \ddagger R}\,(\hat{*})$$

$$\frac{L \ddagger \langle_n \mathsf{ne} \rangle_n^m \ddagger R \qquad m \neq n}{(L@\star) \ddagger (n\ \star) \cdot \mathsf{ne} \ddagger (((m\ \star) \cdot R)\ @m)}\,(\hat{\Diamond}_{\neq}) \qquad \frac{L \ddagger \langle_n \mathsf{ne} \rangle_n^m \ddagger R \qquad m = n}{(L@\star) \ddagger (n\ \star) \cdot \mathsf{ne} \ddagger (R@\star)}\,(\hat{\Diamond}_{=})$$

Fig. 1. Rules computing expressions-in-contexts

Given a closed p-NRE ne, we start applying the rules to the *expression-in-contexts* $[] \ddagger \mathsf{ne} \ddagger []$. In $(\hat{*})$ in Fig. 1, k is a natural number; if $k = 0$, the conclusion of the rule is $L \ddagger \mathit{1} \ddagger R$. Also, in $(\hat{\Diamond}_{=})$ and $(\hat{\Diamond}_{\neq})$, $\star$ denotes a name fresh for L and for R.

Fact 1. *For any derivation of* $L \ddagger \mathsf{ne}' \ddagger R$ *from an expression-in-contexts* $[] \ddagger \mathsf{ne} \ddagger []$ *using the rules in Fig. 1 we have that*

- *there is a permutation* π *such that* $R = \pi \cdot L$ *(and hence* $lth(L) = lth(R)$*)*
- *names in L are pairwise disjoint (and similarly for R)*

Example 3. Application of the rules in Fig. 1 to $\langle_m \langle_n m \rangle_n^m \langle_n nm \rangle_n^n \rangle_m^m$ gives:

$$\begin{array}{c}
\dfrac{[] \ddagger \langle_m \langle_n m \rangle_n^m \langle_n nm \rangle_n^n \rangle_m^m \ddagger []}{[a] \ddagger \langle_n a \rangle_n^a \langle_n na \rangle_n^n \ddagger [a]}\,(\hat{\Diamond}_{=}) \\
\hline
(\hat{\Diamond}_{\neq})\,\dfrac{[a] \ddagger \langle_n a \rangle_n^a \ddagger [a]}{[a,b] \ddagger a \ddagger [b,a]} \qquad \dfrac{\dfrac{[a] \ddagger \langle_n na \rangle_n^n \ddagger [a]}{[a,c] \ddagger ca \ddagger [a,c]}\,(\hat{\Diamond}_{=})}{[a,c] \ddagger c \ddagger [a,c] \qquad [a,c] \ddagger a \ddagger [a,c]}\,(\hat{\circ})
\end{array}\,(\hat{\circ})$$

the derivation tree is read from top to bottom

Note that b and c are distinct from a, but b may be the same as c (Fact 1). Also, in the first step of the derivation, we can take n instead of a as a fresh name, yielding $[n] \ddagger \langle_m n \rangle_m^n \langle_m mn \rangle_m^m \ddagger [n]$ as the conclusion. $\diamond$

By the rules of Fig. 1, there may be more than one *derivation tree for* $[] \ddagger \mathsf{ne} \ddagger []$ (one can choose either of the branches in a sum or unfold any number of times a Kleene-star). We associate a language to each derivation tree T. This is done by applying the rules in Fig. 2, starting from the leaves of T and going upwards to the root. Finally, we define *the language of* ne to be the union of the languages of all derivation trees for ne, and call such languages *nominal regular*.

To define the rules in Fig. 2, we extend the notation of expressions-in-contexts to languages and write e.g., $L \ddagger \mathbf{L}(\mathsf{ne}) \ddagger R$. Rules $(\mathit{1})$, $(\mathit{0})$, (n) and (s) yield the natural interpretation for basic expressions. Rule $(\check{\circ})$ deals with the concatenation of languages; note that, since permutations may change the post-context, it is necessary to rename

$$\dfrac{L \ddagger \mathit{1} \ddagger R}{L \ddagger \{\varepsilon\} \ddagger R}\,(\mathit{1}) \qquad \dfrac{L \ddagger \mathit{0} \ddagger R}{L \ddagger \emptyset \ddagger R}\,(\mathit{0}) \qquad \dfrac{L \ddagger n \ddagger R}{L \ddagger \{n\} \ddagger R}\,(n) \qquad \dfrac{L \ddagger s \ddagger R}{L \ddagger \{s\} \ddagger R}\,(s)$$

$$\dfrac{L \ddagger \mathbf{L}(\mathsf{ne}_1) \ddagger M \qquad M' \ddagger \mathbf{L}(\mathsf{ne}_2) \ddagger R}{L \ddagger \{w \circ \left(\pi_{[M' \rhd M]} \cdot v\right) \mid w \in \mathbf{L}(\mathsf{ne}_1), v \in \mathbf{L}(\mathsf{ne}_2)\} \ddagger \left(\pi_{[M' \rhd M]} \cdot R\right)}\,(\check{\circ})$$

$$\dfrac{(L@n) \ddagger \mathbf{L}(\mathsf{ne}) \ddagger (R@m)}{L \ddagger \{(n\,\star) \cdot w \mid \star \in \mathcal{N}, w \in \mathbf{L}(\mathsf{ne}) \text{ and } \star \# L\} \ddagger ((n\,\star) \cdot R)}\,(\check{\Diamond})$$

Fig. 2. Rules computing languages

everything by a permutation $\pi_{[M' \rhd M]}$ before combining them (recall $\pi_{[M' \rhd M]}$ from § 3.1). Also, the fact that the rule is applied on proof trees obtained by rules in Fig. 1 implies that $lth(M') = lth(M)$.

Rule $(\check{\Diamond})$ deallocates n. If $n \in R$, then $(n\,\star) \cdot R$ remembers the fresh name $\star$ in the new post-context. This rule maintains the invariant that the set of names in the pre-context is in bijection to the set of names in the post-context.

For simplicity, in Fig. 2 we do not explicitly consider e.g., rules for the $+$ operator; when such kind of nodes are reached, the computed language is just the language of the branch and similarly for the Kleene-star (cf. Example 5 below).

Example 4. Starting from the tree in Example 3, we calculate the language of the expression $\langle_m \langle_n m\rangle_n^m \langle_n nm\rangle_n^n \rangle_m^m$

$$\dfrac{\dfrac{\dfrac{[a,b] \ddagger a \ddagger [b,a]}{[a,b] \ddagger \{a\} \ddagger [b,a]}\,(a)}{\begin{array}{c}[a] \ddagger \{(b\,\star_1) \cdot a \mid \star_1 \in \mathcal{N}, \star_1 \neq a\} \ddagger [\star_1] \\ \hdashline [a] \ddagger \{a \mid \star_1 \in \mathcal{N}, \star_1 \neq a\} \ddagger [\star_1]\end{array}}\,(\check{\Diamond}) \qquad \dfrac{\dfrac{\dfrac{[a,c] \ddagger c \ddagger [a,c]}{[a,c] \ddagger \{c\} \ddagger [a,c]}\,(c) \quad \dfrac{[a,c] \ddagger a \ddagger [a,c]}{[a,c] \ddagger \{a\} \ddagger [a,c]}\,(a)}{[a,c] \ddagger \{ca\} \ddagger [a,c]}\,(\check{\circ})}{\begin{array}{c}[a] \ddagger \{(c\,\star_2) \cdot ca \mid \star_2 \in \mathcal{N}, \star_2 \neq a\} \ddagger [a] \\ \hdashline [a] \ddagger \{\star_2 a \mid \star_2 \in \mathcal{N}, \star_2 \neq a\} \ddagger [a]\end{array}}\,(\check{\Diamond})}{\dfrac{\begin{array}{c}[a] \ddagger \{a \circ ((a\,\star_1) \cdot \star_2 a) \mid \star_1, \star_2 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq a\} \ddagger [\star_1] \\ \hdashline [a] \ddagger \{a \star_2 \star_1 \mid \star_1, \star_2 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq \star_1\} \ddagger [\star_1]\end{array}}{\begin{array}{c}[] \ddagger \{(a\,\star_3) \cdot a \star_2 \star_1 \mid \star_1, \star_2, \star_3 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq \star_1\} \ddagger [] \\ \hdashline [] \ddagger \{\star_3 \star_2 \star_1 \mid \star_1, \star_2, \star_3 \in \mathcal{N}, \star_1 \neq \star_3, \star_2 \neq \star_1\} \ddagger []\end{array}}\,(\check{\Diamond})}\,(\check{\circ})$$

(where the dashed lines are just simplifications of expressions) and we obtain that $\mathbf{L}(\langle_m \langle_n m\rangle_n^m \langle_n nm\rangle_n^n \rangle_m^m) \overset{\text{def}}{=} \{acb \mid a,b,c \in \mathcal{N}. b \neq a \text{ and } c \neq b\}$.

Interestingly, the application of rule $(\check{\Diamond})$ in the left branch of the above derivation corresponds to the deallocation of m (which yields the name a) and the contextual renaming of the content of n with the new name chosen for n (that is $\star_1$) due to the permutation dictated by the subexpression $\langle_n m\rangle_n^m$. Instead, the application of $(\check{\Diamond})$ in the right branch, simply cuts out the last names of both the left and right contexts because the subexpression $\langle_n nm\rangle_n^n$ does not involve any permutation. $\diamond$

Example 5 below shows that the language $\mathcal{L}_1$ in Example 1 is nominal regular.

Example 5. To show how the rules in Fig. 2 apply when the expressions contain a Kleene-star, we consider $\langle_m (\langle_n n\rangle_n^m)^* \rangle_m^m$ with the derivation tree corresponding to a threefold unfolding of the Kleene-star.

$$\dfrac{\dfrac{\dfrac{[] \ddagger \langle_m(\langle_n n\rangle^m_n)^*\rangle^m_m \ddagger []}{[a] \ddagger (\langle_n n\rangle^a_n)^* \ddagger [a]}(\hat{\Diamond}_=)}{[a] \ddagger \langle_n n\rangle^a_n\langle_n n\rangle^a_n\langle_n n\rangle^a_n \ddagger [a]}(\hat{*})}{\dfrac{[a] \ddagger \langle_n n\rangle^a_n \ddagger [a]}{[a,b] \ddagger b \ddagger [b,a]}(\hat{\Diamond}_{\neq}) \qquad \dfrac{[a] \ddagger \langle_n n\rangle^a_n\langle_n n\rangle^a_n \ddagger [a]}{\dfrac{[a] \ddagger \langle_n n\rangle^a_n \ddagger [a]}{[a,c] \ddagger c \ddagger [c,a]}(\hat{\Diamond}_{\neq}) \quad \dfrac{[a] \ddagger \langle_n n\rangle^a_n \ddagger [a]}{[a,d] \ddagger d \ddagger [d,a]}(\hat{\Diamond}_{\neq})}(\hat{\circ})}(\hat{\circ})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[a,b] \ddagger b \ddagger [b,a]}{[a,b] \ddagger \{b\} \ddagger [b,a]}(b)}{[a] \ddagger \{\star_1 \mid \star_1 \neq a\} \ddagger [\star_1]}(\check{\Diamond}) \quad \dfrac{\dfrac{\dfrac{[a,c] \ddagger c \ddagger [c,a]}{[a,c] \ddagger \{c\} \ddagger [c,a]}(c)}{[a] \ddagger \{\star_2 \mid \star_2 \neq a\} \ddagger [\star_2]}(\check{\Diamond}) \quad \dfrac{\dfrac{[a,d] \ddagger d \ddagger [d,a]}{[a,d] \ddagger \{d\} \ddagger [d,a]}(d)}{[a] \ddagger \{\star_3 \mid \star_3 \neq a\} \ddagger [\star_3]}(\check{\Diamond})}{[a] \ddagger \{\star_2\star_3 \mid \star_2 \neq a, \star_3 \neq \star_2\} \ddagger [\star_3]}(\check{\circ})}{[a] \ddagger \{\star_1\star_2\star_3 \mid \star_1 \neq a, \star_2 \neq \star_1, \star_3 \neq \star_2\} \ddagger [\star_3]}(\check{\circ})}{[] \ddagger \{\star_1\star_2\star_3 \mid \star_1 \neq \star_0, \star_2 \neq \star_1, \star_3 \neq \star_2\} \ddagger []}(\check{\Diamond})}{[] \ddagger \{\star_1\star_2\star_3 \mid \star_2 \neq \star_1, \star_3 \neq \star_2\} \ddagger []}$$

Generalising to a k-fold unfolding, we have the language $L^k = \{a_1 \cdots a_k \mid a_1 \neq a_0, \ldots, a_k \neq a_{k-1}\}$ which yields

$$\mathbf{L}(\langle_m(\langle_n n\rangle^m_n)^*\rangle^m_m) \overset{\text{def}}{=} \bigcup_{k\in\mathbb{N}} L^k = \{a_1 \cdots a_k \mid \forall k \in \mathbb{N}, \forall i \in \{1,\ldots,k-1\}. a_i \neq a_{i+1}\} \qquad \diamond$$

We can also express the following languages taken from [11]. Note that the second is not quasi-regular in the sense of [11], that is, it cannot be accepted by FMAs.

Example 6. Define $\mathsf{N} \overset{\text{def}}{=} \langle_n n\rangle^n_n$. Note that $\mathbf{L}(\mathsf{N}) = \mathcal{N}$. We have

$$\mathbf{L}(\langle_n \mathsf{N}^* n \mathsf{N}^* n\rangle^n_n \mathsf{N}^*) = \{a_1 \cdots a_k \mid \exists i,j.\, 1 \leq i < j \leq k \;\&\; a_i = a_j\},$$
$$\mathbf{L}(\langle_n \mathsf{N}^* n\rangle^n_n) = \{a_1 \cdots a_k \mid \forall i.\, 1 \leq i < k \;\Rightarrow\; a_i \neq a_k\}. \qquad \diamond$$

The language $\mathcal{L}_{\alpha\rho}$ in Example 2 is nominal regular as discussed in the next example.

Example 7. Calculations similar to those in Examples 4 and 5 can be done to show that the p-NRE $\alpha\langle_m((m^*\langle_n(m+n)\rho\rangle^m_n m^*)+m^*)\rho\rangle^m_m$ (where $\alpha,\rho \in \mathcal{S}$) defines $\mathcal{L}_{\alpha\rho}$. $\diamond$

α-conversion on p-NREs. Permutations can be used to define α-conversion on p-NREs. Let S be a finite set of $\mathcal{N}$ and let π_S be a bijection on $\mathcal{N}$ which fixes each element in S. We say that $L \ddagger \pi_L \cdot \mathsf{ne} \ddagger R$ is an α-conversion of an expression-in-contexts $L \ddagger \mathsf{ne} \ddagger R$. For a p-NRE ne, ne' is an α-conversion of ne, if ne' is obtained by α-converting a subexpression of ne which appears in a derivation tree from $[] \ddagger \mathsf{ne} \ddagger []$ using the rules Fig. 1 (taking care of unfolding Kleene-stars only once and of avoiding any renamings in name abstractions). For example, a p-NRE $\langle_m\langle_n m\rangle^m_n\langle_n nm\rangle^n_n\rangle^m_m$ is α-converted to $\langle_l\langle_m l\rangle^l_m\langle_n nl\rangle^n_n\rangle^l_l$.

$$\dfrac{\dfrac{[] \ddagger \langle_a\langle_m a\rangle^a_m\langle_n na\rangle^n_n\rangle^a_a \ddagger []}{\dfrac{[a] \ddagger \langle_m a\rangle^a_m\langle_n na\rangle^n_n \ddagger [a]}{(m\ n)\;\dfrac{[a] \ddagger \langle_n a\rangle^a_n \ddagger [a]}{[a] \ddagger \langle_m a\rangle^a_m \ddagger [a]} \qquad [a] \ddagger \langle_n na\rangle^n_n \ddagger [a]}(\hat{\circ})}(\check{\Diamond})}{[] \ddagger \langle_l\langle_m l\rangle^l_m\langle_n nl\rangle^n_n\rangle^l_l \ddagger []}(l\ a)$$

Proposition 1. *α-conversion on p-NREs is an equivalence relation on p-NREs.*

Theorem 1. *For p-NREs* ne_1 *and* ne_2*, if they are α-equivalent, they define the same nominal regular languages, i.e.* $\mathbf{L}(\mathsf{ne}_1) = \mathbf{L}(\mathsf{ne}_2)$.

The above results are due to the fact that a completely fresh name is always available when applying the rules $(\hat{\Diamond}_{\neq})$ and $(\hat{\Diamond}_{=})$ in Fig. 1.

4 Automata with Fresh-Name Generations and Permutations

To handle *binders (fresh names)* and *permutations*, we extend *automata on binders over $\mathcal{S}$ and $\mathcal{N}_{fin}$* in [13]. Denote the set of natural numbers with $\mathbb{N}$ and define $\underline{i}$ as $\{1,\ldots,i\}$ for each $i \in \mathbb{N}$.

The automata in Definition 1 below have a set (of states) Q and a map $\|\_\| : Q \to \mathbb{N}$ which yield the *local registers of* $q \in Q$ as $\underline{\|q\|}$. Also, given $q \in Q$,

$$\mathfrak{L}(q) \overset{\text{def}}{=} \mathcal{S} \cup \underline{\|q\|} \cup \{\star\} \cup \{\circlearrowleft_i \mid i \in \underline{\|q\|}\},$$

is the set of possible labels of q.

Definition 1. *An* automaton with fresh-name generation and permutations over $\mathcal{S}$*, an* fp-automaton *for short, is a tuple $\mathcal{H} = \langle Q, q_0, F, tr \rangle$ such that*

- *Q is a finite set (of states) equipped with a map $\|\_\| : Q \to \mathbb{N}$*
- *$q_0 \in Q$ is* the initial state *and $\|q_0\| = 0$*
- *$F \subseteq Q$ is the set of* final states *and $\|q\| = 0$ for each $q \in F$*
- *for each $q \in Q$ and $\alpha \in \mathfrak{L}(q) \cup \{\varepsilon\}$, the set $tr(q,\alpha) \subseteq Q$ contains the* successor states *of q; for all $q' \in tr(q,\alpha)$, the following conditions must hold:*

$$\begin{aligned} \alpha = \star \quad &\Longrightarrow \quad \|q'\| = \|q\| + 1 \\ \alpha = \circlearrowleft_i \ \textit{for}\ i \in \underline{\|q\|} \quad &\Longrightarrow \quad \|q'\| = \|q\| - 1 \\ \alpha \in \mathcal{S} \cup \underline{\|q\|} \ \textit{or}\ \alpha = \varepsilon \quad &\Longrightarrow \quad \|q'\| = \|q\| \end{aligned}$$

An fp-automaton is deterministic*, if for each $q \in Q$ and each label $\alpha \in \mathfrak{L}(q)$*

$$\begin{cases} |tr(q,\varepsilon)| = 0, & \\ |tr(q,\alpha)| = 1, & \textit{otherwise.} \end{cases}$$

Finally, the i-th layer of $\mathcal{H}$ *is the subset $Q^i \overset{\text{def}}{=} \{q \in Q \mid \|q\| = i\}$ of Q.*

In an fp-automaton, the i-th layer is connected only by $\star$ to the $(i+1)$-th layer, and only by $\{\circlearrowleft_j \mid j \in \underline{i}\}$ to the $(i-1)$-th layer. Note that the i-th layer forms an automaton over $\mathcal{S} \cup \underline{i} \cup \{\varepsilon\}$ in the classical sense. Note that each state on the 0-th layer cannot have any $\circlearrowleft_\_$ transition, by definition; similarly, states in the highest layer cannot have $\star$-transitions. For a technical reason, we assume every fp-automaton is accessible in the usual sense.

Hereafter we fix an fp-automaton as $\mathcal{H} = \langle Q, q_0, F, tr \rangle$. A *configuration* of $\mathcal{H}$ is a triple $\langle q, w, \sigma \rangle$ consisting of a state q, a map $\sigma \colon \underline{\|q\|} \to \mathcal{N}$ assigning names to the local registers in q and a word w. The following definition is almost the same as the one in [13]. The only exceptions are $\star$-transitions and $\circlearrowleft_i$-transitions.

Definition 2. *Given $q, q' \in Q$ and two configurations $t = \langle q, w, \sigma\rangle$ and $t' = \langle q', w', \sigma'\rangle$, an fp-automaton $\mathcal{H}$* moves from *t* to *t' (written $t \overset{\mathcal{H}}{\to} t'$) if there is $\alpha \in \mathfrak{L}(q) \cup \{\varepsilon\}$ such that $q' \in tr(q, \alpha)$ and*

$$\begin{cases} \alpha \in \underline{\|q\|}, & w = \sigma(\alpha)\ w' \text{ and } \sigma' = \sigma \\ \alpha \in \mathcal{S}, & w = \alpha\, w' \text{ and } \sigma' = \sigma \\ \alpha = \varepsilon, & w = w' \text{ and } \sigma' = \sigma \\ \alpha = \star, & w = w', n \in \mathcal{N} \setminus Im(\sigma) \text{ and } \sigma' = \sigma_{[\|q'\| \mapsto n]} \\ \alpha = \circlearrowleft_i, & w = w' \text{ and } \sigma' = (\sigma \cdot (\|q\|\ i))_{|\underline{\|q'\|}} \end{cases}$$

where $(\sigma \cdot (\|q\|\ i))_{|\underline{\|q'\|}}$ is the restriction on $\underline{\|q'\|}$ of the function $\sigma \cdot (\|q\|\ i)$, i.e. σ permuted by $(\|q\|\ i)$. A configuration $\langle q, w, \sigma\rangle$ is initial *if $q = q_0$, w is a word and $\sigma = \perp$, and it is* accepting *if $q \in F$, $w = \varepsilon$ and $\sigma = \perp$.*

The set $reach_{\mathcal{H}}(t)$ of states reached by $\mathcal{H}$ from the configuration t is defined as

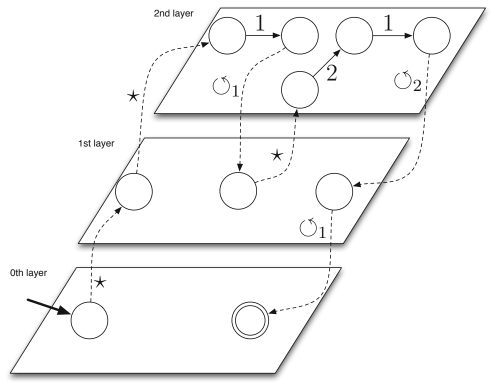
$$reach_{\mathcal{H}}(t) \overset{\text{def}}{=} \begin{cases} \{q\} & \text{if } t = \langle q, \varepsilon, \sigma\rangle \\ \bigcup_{t \overset{\mathcal{H}}{\to} t'} reach_{\mathcal{H}}(t') & \text{otherwise} \end{cases}$$

A run of $\mathcal{H}$ on a word *w is a sequence of moves of $\mathcal{H}$ from $\langle q_0, w, \perp\rangle$.*

Intuitively, $\star$ means "generate a *fresh* name" and store it in the highest register. Transitions labelled by $\circlearrowleft_i$ are meant to permute the value in highest register with the one in the i-th register and dispose the highest register. The $\star$ and $\circlearrowleft_i$ transitions are performed independently of the word w and introduce some non-determinism even to deterministic fp-automata.

Definition 3. *The fp-automaton $\mathcal{H}$* accepts*, or* recognises*, a word w on $\mathcal{S} \cup \mathcal{N}$ when $F \cap reach_{\mathcal{H}}(\langle q_0, w, \perp\rangle) \neq \emptyset$. The language of $\mathcal{H}$ is the set $\mathcal{L}_{\mathcal{H}}$ of words accepted by $\mathcal{H}$.*

Example 8. The fp-automaton below



accepts the language $\mathbf{L}(\langle_m \langle_n m\rangle_n^m \langle_n nm\rangle_n^n\rangle_m^m)$, see Examples 3 and 4. $\diamond$

5 A Kleene Theorem

We show a Kleene theorem for nominal regular languages: Every nominal regular language is recognised by an fp-automaton (Theorem 2) and, vice versa, every language accepted by an fp-automaton is nominal regular (Theorems 3 and 4).

The interpretation of p-NREs via the rules of Fig. 1 has to be extended to expressions-in-contexts and to *languages-in-contexts*. For example, for $[a] \ddagger \langle_n n\rangle_n^a \langle_n n\rangle_n^a \ddagger [a]$, the language-in-contexts is $[a] \ddagger \{cd \mid \forall c \neq a, \forall d \neq c\} \ddagger [a]$.

5.1 From p-NREs to fp-Automata

Given a p-NRE ne, we shall inductively construct an fp-automaton:

Theorem 2. *Given a p-NRE* ne*, there exists an fp-automaton* $\mathcal{H}$ *which accepts the nominal language* $\mathbf{L}(\mathsf{ne})$*, i.e.* $\mathbf{L}(\mathsf{ne}) = \mathcal{L}_{\mathcal{H}}$.

As seen in § 3, our expressions are *context-dependent* and the contexts are *dynamic*. Similarly, we construct automata-in-contexts $L \ddagger \mathcal{H} \ddagger R$, that is generalised fp-automata where initial and final states may have $lth(L) = lth(M)$ registers equipped with a function η mapping the h-th register of the initial state to the h-th name of L. Abusing notation, we let $\mathcal{H}$ to range over automata-in-contexts.

Base cases. Let $L \ddagger \mathsf{ne} \ddagger R$ be an expression-in-contexts. By Fact 1, we can assume that L and R have the same elements (hence $lth(L) = lth(R)$). Since L and R are in general non-empty, we equip fp-automata with a function η that maps the local registers of the initial state to names (in L).

When $\mathsf{ne} = \mathit{1}$ or $\mathsf{ne} = \mathit{0}$, we define

$\mathcal{H}_{(\!|\mathit{1}|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows

- $Q \overset{\text{def}}{=} \{q_0\}$ with $\|q_0\| = lth(L)$;
- $tr(q_0, \alpha) \overset{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_0)$;
- $F \overset{\text{def}}{=} \{q_0\}$;
- $\eta(k) \overset{\text{def}}{=} l_k$, for each $k \in \{1, \dots, lth(L)\}$.

$\mathcal{H}_{(\!|\mathit{0}|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

- $Q \overset{\text{def}}{=} \{q_0\}$ with $\|q_0\| = lth(L)$;
- $tr(q_0, \alpha) \overset{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_0)$;
- $F \overset{\text{def}}{=} \emptyset$;
- $\eta(k) \overset{\text{def}}{=} l_k$ for each $k \in \{1, \dots, lth(L)\}$.

When $\mathsf{ne} = n$, we let $\mathcal{H}_{(\!|n|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

- $Q \overset{\text{def}}{=} \{q_0, q_1\}$ with $\|q_0\| = lth(L)$ and $\|q_1\| = lth(L)$;
- $tr(q_0, \alpha) \overset{\text{def}}{=} \begin{cases} \{q_1\} & \alpha = k \text{ and } l_k = n \\ \emptyset & \text{otherwise} \end{cases}$ and $tr(q_1, \alpha) \overset{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_1)$;
- $F \overset{\text{def}}{=} \{q_1\}$; $\eta(k) \overset{\text{def}}{=} l_k$ for each $k \in \{1, \dots, lth(L)\}$.

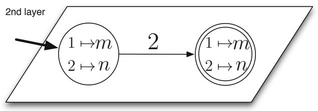
When $\mathsf{ne} = s$, we let $\mathcal{H}_{(\!|s|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

- $Q \overset{\text{def}}{=} \{q_0, q_1\}$ with $\|q_0\| = lth(L)$ and $\|q_1\| = lth(L)$;
- $tr(q_0, \alpha) \overset{\text{def}}{=} \begin{cases} \{q_1\} & \alpha = s \\ \emptyset & \text{otherwise} \end{cases}$ and $tr(q_1, \alpha) \overset{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_1)$;
- $F \overset{\text{def}}{=} \{q_1\}$; $\eta(k) \overset{\text{def}}{=} l_k$ for each $k \in \{1, \dots, lth(L)\}$.

Proposition 2. *The automata-in-contexts $L \ddagger \mathcal{H}_{(\!|1|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|0|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|s|\!)} \ddagger R$ accept the languages-in-contexts $L \ddagger \{\varepsilon\} \ddagger R$, $L \ddagger \emptyset \ddagger R$, $L \ddagger \{n\} \ddagger R$ and $L \ddagger \{s\} \ddagger R$, respectively. Furthermore, for every final state q in $L \ddagger \mathcal{H}_{(\!|1|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|0|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|s|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

For automata-in-contexts we consider configurations and runs as in Definition 2, with the exception that the η in the initial and final configurations $\langle q, w, \eta\rangle$ takes into account the names in the pre- and post-contexts.

Example 9. The automaton-in-contexts $[m,n] \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger [n,m]$ below



is constructed from $[m,n] \ddagger n \ddagger [n,m]$ ◇

Union. Let $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_2|\!)} \ddagger R$ be automata-in-contexts, where $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)} = \langle Q_1, q_{1,0}, tr_1, F_1, \eta_1\rangle$ and $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)} = \langle Q_2, q_{2,0}, tr_2, F_2, \eta_2\rangle$ for the corresponding expressions-in-contexts $L \ddagger \mathsf{ne}_1 \ddagger R$ and $L \ddagger \mathsf{ne}_2 \ddagger R$. Therefore, η_1 and η_2 are identical. Then, we let $\mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} = \langle Q^+, q_0^+, tr^+, F^+, \eta^+\rangle$ as follows:

– $Q^+ \stackrel{\text{def}}{=} \{q_0^+\} \uplus Q_1 \uplus Q_2$ with $\|q_0^+\| = lth(L)$;
– $tr^+(q_0^+, \alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_{1,0}, q_{2,0}\} & \alpha = \varepsilon \\ \emptyset & \text{otherwise} \end{cases}$ and the others are the same as before;
– $F^+ \stackrel{\text{def}}{=} F_1 \uplus F_2$; $\qquad \eta \stackrel{\text{def}}{=} \eta_1\ (= \eta_2)$.

Proposition 3. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} \ddagger R$ accepts the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}_1 + \mathsf{ne}_2) \ddagger R$. Furthermore, for each final state q in $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

Concatenation. By the context calculus of Fig. 1, the post-context of the first expression must be the same as the pre-context of the second expression. Let $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1|\!)} \ddagger L$ and $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_2|\!)} \ddagger R$ be automata-in-contexts with $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)} = \langle Q_1, q_{1,0}, tr_1, F_1, \eta_1\rangle$ and $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)} = \langle Q_2, q_{2,0}, tr_2, F_2, \eta_2\rangle$, and $L \ddagger \mathsf{ne}_1 \ddagger L$ and $L \ddagger \mathsf{ne}_2 \ddagger R$ the corresponding expressions-in-contexts. By the definition of the context calculus, the post-context of the first expression must be the same as the pre-context of the second expression. We let $\mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} = \langle Q^\circ, q_0^\circ, tr^\circ, F^\circ, \eta^\circ\rangle$ as follows:

– $Q^\circ \stackrel{\text{def}}{=} Q_1 \uplus Q_2$; $\quad q_0^\circ \stackrel{\text{def}}{=} q_{1,0}$; $\quad F^\circ \stackrel{\text{def}}{=} F_2$; $\quad \eta^\circ \stackrel{\text{def}}{=} \eta_1$;
– $tr^\circ(q, \alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q,\alpha) \cup \{q_{2,0}\} & q \in F_1 \text{ and } \alpha = \varepsilon \\ tr_1(q,\alpha) & q \in Q_1 \text{ and either } q \notin F_1 \text{ or } \alpha \neq \varepsilon\,. \\ tr_2(q,\alpha) & q \in Q_2 \end{cases}$

Proposition 4. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} \ddagger R$ accepts the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}_1 \circ \mathsf{ne}_2) \ddagger R$. Furthermore, for each final state q in $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

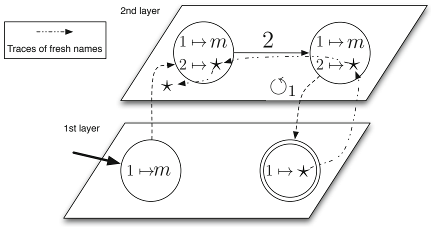
Name-abstraction. Let $(L@n) \ddagger \mathcal{H}_{(\!|\mathsf{ne}|\!)} \ddagger (R@m)$ be an automaton-in-contexts, where $\mathcal{H}_{(\!|\mathsf{ne}|\!)} = \langle Q, q_0, tr, F, \eta\rangle$, and $(L@n) \ddagger \mathsf{ne} \ddagger (R@m)$ the expression-in-contexts. We let $\mathcal{H}_{(\!|\langle_n \mathsf{ne}\rangle^m_n|\!)} = \langle Q^\diamondsuit, q_0^\diamondsuit, tr^\diamondsuit, F^\diamondsuit, \eta^\diamondsuit\rangle$ as follows:

- $Q^\diamondsuit \stackrel{\text{def}}{=} Q \uplus \{q_s, q_t\}$ with $\|q_s\| = lth(L)$ and $\|q_t\| = lth(L)$;
- $q_0^\diamondsuit \stackrel{\text{def}}{=} q_s$; $\quad F^\diamondsuit \stackrel{\text{def}}{=} \{q_t\}$; $\quad \eta^\diamondsuit \stackrel{\text{def}}{=} \eta$;
- $tr^\diamondsuit(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_0\} & q = q_s \text{ and } \alpha = \star \\ \emptyset & q = q_s \text{ and } \alpha \neq \star \\ \emptyset & q = q_t \\ \{q_t\} & q \in F \text{ and } \alpha = \circlearrowleft_k \text{ for } k \text{ with } r_k = n \\ \emptyset & q \in F \text{ and } \alpha = \circlearrowleft_k \text{ for } k \text{ with } r_k \neq n \\ tr(q,\alpha) & \text{otherwise} \end{cases}$;

where $(R@m) = [r_1, \ldots, r_{lth(R)+1}]$ (so $r_{lth(R)+1} = m$).

Proposition 5. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\langle_n \mathsf{ne}\rangle^m_n|\!)} \ddagger R$ recognises the language-in-contexts $L \ddagger \mathbf{L}(\langle_n \mathsf{ne}\rangle^m_n) \ddagger R$. Furthermore, for the final state q_t in $L \ddagger \mathcal{H}_{(\!|\langle_n \mathsf{ne}\rangle^m_n|\!)} \ddagger R$, we have $\|q_t\| = lth(R)$.*

Example 10. For $[m,n] \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger [n,m]$, the fp-automaton below



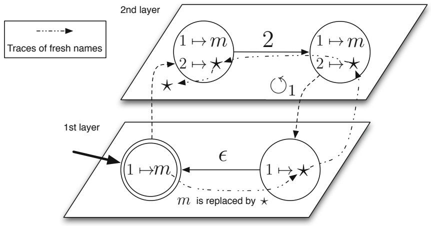
is constructed according to the name-abstraction in contexts $[m] \ddagger \mathcal{H}_{(\!|\langle_n n\rangle^m_n|\!)} \ddagger [m]$. $\diamond$

Kleene star. For an automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}|\!)} \ddagger R$ with $\mathcal{H}_{(\!|\mathsf{ne}|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ and the expression-in-contexts $L \ddagger \mathsf{ne} \ddagger R$, let $\mathcal{H}_{(\!|\mathsf{ne}^*|\!)} = \langle Q^*, q_0^*, tr^*, F^*, \eta^*\rangle$ as follows:

- $Q^* \stackrel{\text{def}}{=} Q$; $\quad q_0^* \stackrel{\text{def}}{=} q_0$; $\quad F^* \stackrel{\text{def}}{=} \{q_0^*\}$; $\quad \eta^* \stackrel{\text{def}}{=} \eta$;
- $tr^*(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q,\alpha) \cup \{q_0^*\} & q \in F \text{ and } \alpha = \varepsilon \\ tr(q,\alpha) & \text{otherwise} \end{cases}$.

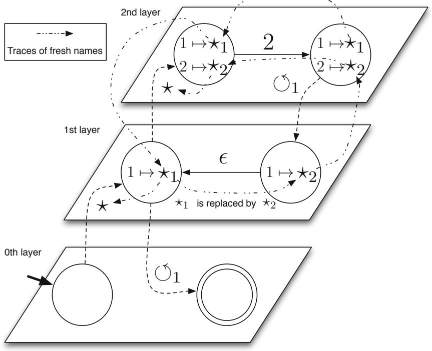
Proposition 6. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}^*|\!)} \ddagger R$ recognises the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}^*) \ddagger R$. Furthermore, for the final state q_0 in $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}^*|\!)} \ddagger R$, we have $\|q_0\| = lth(R)$.*

Example 11. For $[m] \ddagger \mathcal{H}_{(\!|\langle_n n\rangle^m_n|\!)} \ddagger [m]$, the fp-automaton



is the Kleene star construction for $[m] \ddagger \mathcal{H}_{(\!|(\langle_n n \rangle_n^m)^*|\!)} \ddagger [m]$. $\diamond$

From the fp-automaton in Example 11 we build an fp-automaton that accepts the language $\mathcal{L}_1$ in Example 1 by name-abstraction of $[m] \ddagger \mathcal{H}_{(\!|(\langle_n n \rangle_n^m)^*|\!)} \ddagger [m]$. This yields the following fp-automaton



5.2 From fp-Automata to p-NREs

Deterministic and non-deterministic fp-automata are equivalent.

Theorem 3. *Given an fp-automaton $\mathcal{H}$, there is a deterministic fp-automaton which accepts the same language as $\mathcal{L}_{\mathcal{H}}$.*

Proof (Sketch). The main proof technique is a "layer-wise" powerset construction. Since the i-th layer is basically a classical automaton over $\mathcal{S} \cup \underline{i} \cup \{\varepsilon\}$, the powerset construction allows us to make each layer deterministic. The only thing we have to care about is how to connect these deterministic layers by $\star$ and $\{\circlearrowleft_{i'} \mid i' \in \underline{i}\}$ in a deterministic way. This is shown below.

For each subset $\{q_1^i, \dots, q_k^i\}$ of the i-th layer Q^i, we let

$$tr(\{q_1^i, \dots, q_k^i\}, \star) \overset{\text{def}}{=} \{q^{i+1} \in Q^{i+1} \mid \exists j \in \underline{k}.\ q^{i+1} \in tr(q_j^i, \star)\}$$
$$tr(\{q_1^i, \dots, q_k^i\}, \circlearrowleft_{i'}) \overset{\text{def}}{=} \{q^{i-1} \in Q^{i-1} \mid \exists j \in \underline{k}.\ q^{i-1} \in tr(q_j^i, \circlearrowleft_{i'})\}$$

for each $i' \in \underline{i}$. Hence we obtain a deterministic automaton of $\mathcal{H}$. $\square$

Note that the powerset construction in the proof above has to be performed layer-wise due to the presence of local registers.

Theorem 4. *Any language accepted by a deterministic fp-automaton $\mathcal{H}$ is a nominal regular language. That is, there exists a p-NRE* ne *such that $\mathcal{L}_{\mathcal{H}} = \mathbf{L}(\mathsf{ne})$.*

Proof (Sketch). The states Q of a deterministic fp-automaton $\mathcal{H}$ can be decomposed into $h = \max_{q \in Q} \|q\|$ layers (where h is the highest layer of $\mathcal{H}$):

$$Q^0 = \{q_1^0, \dots, q_{m_0}^0\}, \qquad Q^1 = \{q_1^1, \dots, q_{m_1}^1\}, \qquad \cdots \qquad Q^h = \{q_1^h, \dots, q_{m_h}^h\} \tag{3}$$

Note that $q_0 \in Q^0$ (we assume $q_1^0 = q_0$) and $F \subseteq Q^0$. We fix an arbitrary order on states given by their index in (3), let ${}^sR^k_{i,j}$ denote the set of paths from q^s_i to q^s_j which visit only states on layers higher than s or states $q^s_r \in Q^s$ with $r \leq k$, and let $E_{i,j} \overset{\text{def}}{=} \emptyset$ if $i \neq j$ and $E_{i,i} \overset{\text{def}}{=} \{\varepsilon\}$. Then, ${}^sR^k_{i,j}$ is defined by

$$ {}^hR^0_{i,j} \overset{\text{def}}{=} \{\alpha \mid q^h_j \in tr(q^h_i, \alpha)\} \cup E_{i,j} \qquad {}^hR^k_{i,j} \overset{\text{def}}{=} {}^hR^{k-1}_{i,k} \left({}^hR^{k-1}_{k,k}\right)^* {}^hR^{k-1}_{k,j} \cup {}^hR^{k-1}_{i,j} $$

on the highest layer h. On the other layers ($s < h$), it is defined by

$$ \begin{aligned} {}^sR^0_{i,j} &\overset{\text{def}}{=} \{\alpha \mid q^s_j \in tr(q^s_i, \alpha)\} \cup \bigcup_{s' \in \underline{s+1}} \Big\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s,s'}_{i,j}} {}^{s+1}R^{m_{s+1}}_{i',j'} \Big\rangle^{s'}_{s+1} \cup E_{i,j} \\ {}^sR^k_{i,j} &\overset{\text{def}}{=} {}^sR^{k-1}_{i,k} \left({}^sR^{k-1}_{k,k}\right)^* {}^sR^{k-1}_{k,j} \cup {}^sR^{k-1}_{i,j} \cup \bigcup_{s' \in \underline{s+1}} \Big\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s}_{i,j}} {}^{s+1}R^{m_{s+1}}_{i',j'} \Big\rangle^{s'}_{s+1} \end{aligned} $$

where $\Gamma^{s,s'}_{i,j} \overset{\text{def}}{=} \{(i',j') \mid q^{s+1}_{i'} \in tr(q^s_i, \star) \ \&\ q^s_j \in tr(q^{s+1}_{j'}, \circlearrowleft_{s'})\}$ for each $s' \in \underline{s+1}$. Hence, $\bigcup_{s' \in \underline{s+1}} \Big\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s,s'}_{i,j}} {}^{s+1}R^{m_s}_{i',j'} \Big\rangle^{s'}_{s+1}$ is the collection of all paths from q^s_i to q^s_j visiting only states on the higher layers. Finally, we translate all paths from the initial state to final states into a nominal regular expression, but this is analogous to the classical theory. The only distinction is how to choose fresh names for binders. However, this is done by reserving names for fresh names as a distinct subset $\{n_1, \dots, n_h\}$ of $\mathcal{N}$, with the $\Big\langle_{s+1}$ and $\Big\rangle^{s'}_{s+1}$ indicating how to generate expressions for the binding construct. □

Therefore, by the above theorems, we conclude that every fp-automaton $\mathcal{H}$ has a p-NRE ne such that $\mathcal{L}_{\mathcal{H}} = \mathbf{L}(\mathsf{ne})$.

6 Conclusion

We have extended the nominal regular expressions and automata presented in [13] with permutations in order to provide a notion of regular expression for languages on infinite alphabets (without binders). Our main technical contribution is a Kleene theorem that establishes an equivalence between nominal regular expressions with permutations and fp-automata.

A novelty of our approach is how to handle the environments and how permutations change the local views of the environment. This is done with the help of the context calculus in Fig. 1, which represents the views on the environments by "contexts" similar to Hoare triples. The language construction of Fig. 2 then explains how this information flow generates nominal regular languages.

Yet another delicate aspect of our theory is the subtle non-deterministic behaviour present even in deterministic automata. As highlighted by the first language of Example 6, Definition 2 does not require the automaton to consume a letter if moving on an allocation or deallocation transition. These moves are non-deterministic in the sense that they are not controlled by the word to be recognised. This is crucial to the equivalence established in Theorem 4. Indeed, non-deterministic models are more expressive than deterministic ones when considering languages on infinite alphabets [11,16].

The natural next step to take in our research is to exploit the results presented here to compare the expressiveness of nominal regular expressions with other models featuring languages on infinite alphabets. We note that our nominal regular languages are closed under union, intersection, concatenation and Kleene-star, but not under complement. Whereas the regular languages of [13] are closed under resource-sensitive complement, this is no longer the case here, since allocation and deallocation transitions are no longer controlled by explicit binders in the words. This situation is similar to the FMA of [11] although FMA do not accept the second language of Example 6. A precise comparison with FMA and related models such as those of [16] or [22] is left for future work.

Further investigations should reveal the categorical and (co)algebraic nature of our automata. In particular, the fact that the automata work level-wise suggests a many-sorted approach via presheaves (see also the two-sorted coalgebras of [7]). It would also be interesting to combine the work of this paper with [13] along the lines suggested by [2], which investigates how the implicit scope of names in words without binders interacts with binders having explicit scope. In another direction, we plan to extend our approach towards Kleene algebras (with tests) [12] and possible applications to verification. Other interpretations of the binders in the style of the research programme devised in [4] will also be of interest.

Acknowledgements. We would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3) (2009)
2. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Hard life with weak binders. Electr. Notes Theor. Comput. Sci. 242(1), 49–72 (2009)
3. Bojanczyk, M.: Data monoids. In: STACS, pp. 105–116 (2011)
4. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: IEEE Symposium on Logic in Computer Science, pp. 355–364 (2011)
5. Bouyer, P., Petit, A., Thérien, D.: An algebraic approach to data languages and timed languages. Inf. Comput. 182(2), 137–162 (2003)
6. Ciancia, V., Tuosto, E.: A novel class of automata for languages on infinite alphabets. Technical Report CS-09-003, Leicester (2009)
7. Ciancia, V., Venema, Y.: Stream automata are coalgebras. In: 11th International Workshop on Coalgebraic Methods in Computer Science, CMCS 2012 (2012)
8. 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)
9. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: Symbolic on Logics in Comput Science, pp. 214–224 (1999)
10. Kaminski, M., Tan, T.: Regular expressions for languages over infinite alphabets. Fundam. Inform. 69(3), 301–318 (2006)

11. Kaminski, N., Francez, M.: Finite-memory automata. TCS 134(2), 329–363 (1994)
12. Kozen, D.: On Hoare logic and Kleene algebra with tests. ACM Trans. Comput. Log. 1(1), 60–76 (2000)
13. 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)
14. Montanari, U., Pistore, M.: π-Calculus, Structured Coalgebras and Minimal HD-Automata. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 569–578. Springer, Heidelberg (2000)
15. Myers, R.: Rational Coalgebraic Machines in Varieties: Languages, Completeness and Automatic Proofs. PhD thesis, Imperial College London (2011)
16. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic 5(3), 403–435 (2004)
17. Pistore, M.: History Dependent Automata. PhD thesis, Dipartimento di Informatica, Università di Pisa (1999)
18. Pouillard, N., Pottier, F.: A fresh look at programming with names and binders. In: Proceeding of the 15th ACM SIGPLAN International Conference on Functional Programming, pp. 217–228 (2010)
19. Segoufin, L.: Automata and Logics for Words and Trees over an Infinite Alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
20. Shinwell, M., Pitts, A., Gabbay, M.: Freshml: programming with binders made simple. In: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, pp. 263–274 (2003)
21. Stirling, C.: Dependency Tree Automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 92–106. Springer, Heidelberg (2009)
22. Tzevelekos, N.: Fresh-Register Automata. In: Symposium on Principles of Programming Languages, pp. 295–306. ACM (2011)

Formal Verification of Distributed Algorithms
From Pseudo Code to Checked Proofs

Philipp Küfner, Uwe Nestmann, and Christina Rickmann

Technische Universität Berlin

Abstract. We exhibit a methodology to develop mechanically-checkable parameterized proofs of the correctness of fault-tolerant round-based distributed algorithms in an asynchronous message-passing setting. Motivated by a number of case studies, we sketch how to replace often-used informal and incomplete pseudo code by mostly syntax-free formal and complete definitions of a global-state transition system. Special emphasis is put on the required deepening of the level of proof detail to be able to check them within an interactive theorem proving environment.

1 Introduction

Lamport, Shostak and Pease [LSP82] write about an argument concerning the *Byzantine Generals Problem*: "This argument may appear convincing, but we strongly advise the reader to be very suspicious of such nonrigorous reasoning. Although this result is indeed correct, we have seen equally plausible "proofs" of invalid results. We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm." Along these lines, our goal is to develop mechanically checked proofs about distributed algorithms. In this paper, we address "positive" results on the correctness of problem-solving algorithms, as opposed to "negative" results about the impossibility to solve a problem. We strive for experimental answers to:

– Which description techniques fit best to prove the desired properties of a distributed algorithm when using a theorem prover?
– How big is the gap between informal paper proofs and computer-checked proofs? Is the gap merely due to the level of detail that is hidden in words like "straightforward", or is it due to the incompleteness or inadequacy of the model and the used proof techniques?

Fault-Tolerant Distributed Algorithms. Any analysis of distributed algorithms is placed within a particular context provided by a system model in which both the algorithm and its specification are to be interpreted.

Basic Process Model. We assume a setting of finitely many processes that behave at independent speeds. The controlled global progress of a synchronous system, often paraphrased by means of a sequence of communication rounds, is not available. Moreover, in our model, processes communicate via asynchronous messaging, i.e., without any bound on the delay between emission and reception of messages. Processes typically perform atomic steps consisting of three kinds of actions: (1) input of a message, followed by (2) local computation, followed by (3) output of messages. Starting from some initial configuration of processes and the messaging mechanism, system runs are generated by subsequently performing steps, typically (though not necessarily) in an interleaving fashion. In our model, processes may crash, but do not recover. Here, a process is called *correct* (in a run) if it does not crash (in this run). In other models, where processes may recover, the notion of processes' correctness needs to be adapted.

Distributed Consensus [Lyn96] (or short: Consensus) is a well-known problem in the field of distributed algorithms: n processes with symmetric behavior, but possibly different initial data, are to commonly decide on one out of several possible values. The task is to find an algorithm satisfying the following three trace properties: (1) Validity: If a process decides a value v, then v was initially proposed by some process. (2) Agreement: No two correct processes decide differently. (3) Termination: Every correct process (eventually) decides some value. Fischer, Lynch and Paterson ([FLP85]) showed that Consensus is *not* solvable in a fault-prone environment like the above. Essentially, this is due to the fact that in an asynchronous setting, processes that wait for messages to arrive cannot know whether the sender has crashed or whether the message is just late. Often, this is phrased as a lack of synchrony in the communication infrastructure.

Failure Detection and Round-Based Solutions. To enhance the setting by some degree of partial synchrony, Chandra and Toueg [CT96] introduced the notion of *failure detectors* (FD). They allow to determine what is needed to make Consensus solvable in an asynchronous crash-failure environment. A failure detector can be regarded as a local instance at every process that monitors the crash status of all [other] processes. The information provided by the failure detector is usually imperfect, i.e., unreliable to a certain degree. In addition to an enhancement of the model, a common modeling idiom is to use *round-based algorithms*, which help to simulate the spirit of synchronized executions of the processes to some extent: every process keeps and controls a local round number in its own state. This round number can be attached to messages, which can thereby be uniquely identified. In asynchronous systems, this enables receiving processes to arrange the messages sent by other processes into their intended sequential order.

Proofs Revisited. Our set of case studies comprises several distributed algorithms that solve the problem of Consensus for various system models and with varying degree of complexity, taken from [CT96, Lam98, FH07]. For all of these algorithms, proofs of correctness are available, though of quite different level of detail and employing rather different techniques. The proofs often use an induction principle based on round numbers. This counters the fact that executions of distributed algorithms in an asynchronous model do usually *not* proceed globally round-by-round—which resembles the reasoning in a synchronous model—but rather locally step-by-step. More precisely, the proof does not follow the step-by-step structure of executions, but rather magically proceeds from round to round, while referring to ("the inspection of") individual statements in the pseudo code.

Own Work. Motivated by the lack of formality in typical published proofs of the properties of distributed algorithms, it is part of our own research programme to provide comprehensive formal models and develop formal—but still intuitive—proofs, best mechanically checked by (interactive) theorem provers.

Previous Work. In [FMN07, Fuz08], one of the algorithms of [CT96] and the related Paxos algorithm [Lam98] were presented with a high degree of detail and formality: (*i*) the previously implicit setting of the networked processes including the communication infrastructure is made explicit by global state structures; (*ii*) the previous pseudo code is replaced by a syntax-free description of actions that transform vectors of state variables; (*iii*) the behavior of these two parts is modeled as a labeled global transition system, on which the three trace properties required for Consensus are defined; (*iv*) all proof arguments are backed up by explict reference to this transition system.

Contributions. (1) We demonstrate a method based on the models of [FMN07, Fuz08] to represent distributed algorithms, including the underlying mechanisms for communication and failure detection, together with their respective fairness assumptions, within the theorem-proving environment Isabelle [NPW02]. It turns out that the representation of the transition system specification is quite similar to the syntactic representation of actions in Lamport's *Temporal Logic of Actions* (TLA) [Lam02], enriched by the notion of global configuration and system runs. (2) We explain how this representation, especially the first-class status of system runs, can be used to verify *both* safety *and*—as a novelty—liveness properties within a theorem prover. We argue that, to prove safety properties, it is more intuitive to replace the mere invariant-based style (as exemplified in Isabelle by [JM05]) by arguments that make explicit reference to system runs. Moreover, also to prove liveness properties, explicit system runs represent a convenient proof vehicle. (3) Accompanied by the concrete running example[1] inspired by [FH07], we present a formal framework for distributed algorithms and explain techniques for verifying different classes of properties of the algorithm. (4) We give a quick overview of our case studies and provide links to the mechanized proofs in Isabelle. To our knowledge, it is the first time that algorithms from [CT96] have been mechanically checked. In the case of the Paxos proof of [Fuz08], we were able to correct some bugs, while developing the formalization in Isabelle. In contrast, the case study drawn from [FH07] was previously subject to a process-algebraic proof; here, we just used it as a reasonably simple example to demonstrate the quick applicability of our method, small enough to be presented within the space limits of this paper.

Overview of the Paper. In §2, we introduce our formal model based on transition rules and explicit traces that is fundamental for this work. In §3, we present the respective formalization of the required properties. In §4, we discuss how the proof techniques in the respective settings differ, referring to round-based induction and history variables. In §5, we summarize the results of our case studies on doing mechanized proofs for Consensus algorithms. For the sake of readability,

[1] Full details at `http://www.mtv.tu-berlin.de/fileadmin/a3435/Isa/RotCoord.zip`

we only sketch essential details of the simplest algorithm to demonstrate the method and the shape of the model on which to carry out proofs.

2 Modeling the Algorithm

Distributed algorithms are often described in terms of state machines [Lam78, Sch90], which are used to capture the individual processes' behavior. As in automata theory, transitions from one state to the next may be given by abstractly named actions. In a more concrete version of this model, states can be considered as vectors of values of the relevant variables and transitions are directed manipulations of these vectors. An essential part of a distributed system is the interprocess communication (IPC) infrastructure, e.g., shared memory, point-to-point messaging, broadcast, or remote procedure calls ([Lyn96]). Thus, the global collection of local state machines for the individual processes must be accompanied by some appropriate representation of the IPC.

2.1 Pseudo Code for State Machines

Listing 1.1 exemplifies a common style to present distributed algorithms: a piece of pseudecode that indicates what is executed by each process locally. We use it as a running example to illustrate our experience with Isabelle-checked proofs.

The algorithm solves Consensus in the presence of a *strong* failure detector (see [CT96] and below). We use process ids (PIDs) p_1 to p_n to refer to the n processes. The algorithm proceeds in rounds. For every round, there is one process playing the role of a coordinator. For the given algorithm, coordination means to propose a single value to all processes, while the latter wait for this proposal. The function alive contacts the above-mentioned failure detector to detect the possible failure of the coordinator. The (only) assumption about the failure detector is that at least one correct process is not suspected to be crashed, i.e. the function alive does always return 'True' for some process that never crashes. Correct processes that are not suspected are called *trusted-immortal*.

Listing 1.1. The Rotating Coordinator Algorithm for participant p_i

```
x_i := Input
for r:= 1 to n do {
      if r = i then broadcast x_i;
      if alive(p_r) then x_i := input_from_broadcast;
}
output x_i;
```

As there is no formal semantics and no compiler for pseudo code, formal reasoning is impossible at this level of abstraction. Furthermore, the environment including the IPC-infrastructure (e.g., the various messages in transit) is usually not an explicit part of the pseudo code model. Nevertheless, we choose pseudo code as a starting point for our formalization, as many algorithms are given in

locale *Algorithm* =
  fixes
    $InitPred \quad :: \ 'configuration \Rightarrow bool$ **and**
    $ProcActionSet :: ('configuration \Rightarrow 'configuration \Rightarrow proc \Rightarrow bool)\ set$ **and**
    $ComActionSet :: ('configuration \Rightarrow 'configuration \Rightarrow bool)\ set$ **and**
    $ProcessState :: 'configuration \Rightarrow proc \Rightarrow 'process\text{-}state$
  assumes
    ActionSetsFin:
  finite ProcActionSet
  finite ComActionSet **and**
    StateInv:
  $\bigwedge A\ i.\ [\![A \in ProcActionSet;\ A\ c\ c'\ i;\ ProcessState\ c\ j \neq ProcessState\ c'\ j]\!] \Longrightarrow i = j$
  $\bigwedge A.\ [\![A \in ComActionSet;\ A\ c\ c'\ ]\!] \Longrightarrow ProcessState\ c\ j = ProcessState\ c'\ j$

Fig. 1. Locale for Distributed Algorithm

this style. Hence, the model is both informal and incomplete; it cannot be transferred into a theorem-proving environment without giving a formal semantics to the usually textual description of actions and not without providing a complete representation of the IPC-infrastructure. We provide a formal model for this example, mention more general modeling aspects and show the key techniques used for the respective correctness proofs.

2.2 State Machines in Isabelle

Fuzzati et al. [FMN07] define the algorithm in terms of transition rules where the transitions are computation steps between so-called configurations, which represent the global states of the distributed system. In [FMN07], a configuration at time t consists of three components: (1) an array of the local states of the processes, (2) the message history (a set of all point-to-point messages sent until t) and (3) the broadcast history (a set of all broadcast messages sent until t).

The concrete definition of configurations obviously depends on the respective algorithm. To get a general model for distributed algorithms we introduce an abstract type variable *'configuration*. We apply ideas from Merz et al. [CDM11] and model distributed algorithms as interpretations of the locale given in Fig. 1. Locales are used to introduce parameterized modules in Isabelle's theories (see [Bal03]). In the definition of the locale *Algorithm*, *InitPred* is a predicate that returns true if a given configuration is a valid initial configuration. *ProcActionSet* contains all valid actions that a process can execute: examples are local computations or the sending of messages by placing messages into the local outbox. *ComActionSet* contains (communication) actions that can be performed independently from a dedicated process (by the system or the communication infrastructure); as an example, imagine the loss of a message during transmission. There is no sense in assigning such an action to the sender or the receiver of the message. Both action sets have to be finite. *ProcessState* must be instantiated with a mapping that returns a process state for a given configuration and a process. Of course communication actions are not allowed to change the state

definition
$\mathit{LocalStep} :: {}'\mathit{configuration} \Rightarrow {}'\mathit{configuration} \Rightarrow \mathit{bool}$ **where**
$\mathit{LocalStep}\ c\ c' \equiv \exists\, i \in \mathit{procs}.\ \exists\, A \in \mathit{ProcActionSet}.\ A\ c\ c'\ i$
definition
$\mathit{ComStep} :: {}'\mathit{configuration} \Rightarrow {}'\mathit{configuration} \Rightarrow \mathit{bool}$ **where**
$\mathit{ComStep}\ c\ c' \equiv \exists\, A \in \mathit{ComActionSet}.\ A\ c\ c'$
definition
$\mathit{Step} :: {}'\mathit{configuration} \Rightarrow {}'\mathit{configuration} \Rightarrow \mathit{bool}$ (**infixl** $\rightarrow$ $\mathit{900}$) **where**
$\mathit{Step}\ c\ c' \equiv \mathit{LocalStep}\ c\ c' \lor \mathit{ComStep}\ c\ c'$
definition
$\mathit{deadlock} :: {}'\mathit{configuration} \Rightarrow \mathit{bool}$ **where**
$\mathit{deadlock}\ c \equiv \forall\, c'.\ \neg\ c \rightarrow c'$
definition
$\mathit{FinalStuttering} :: {}'\mathit{configuration} \Rightarrow {}'\mathit{configuration} \Rightarrow \mathit{bool}$ **where**
$\mathit{FinalStuttering}\ s\ s' \equiv (s = s') \land \mathit{deadlock}\ s$
definition
$\mathit{Run} :: (T \Rightarrow {}'\mathit{configuration}) \Rightarrow \mathit{bool}$ **where**
$\mathit{Run}\ R \equiv \mathit{InitPred}\ (R\ \mathit{0}) \land (\forall\, t{::}T.\ ((R\ t) \rightarrow (R\ (t{+}\mathit{1})))$
$\lor\ \mathit{FinalStuttering}\ (R\ t)\ (R\ (t{+}\mathit{1})))$

Fig. 2. Definition: Steps and Runs

of any process and performed by some process may not change the state of any other process. This is asserted by *StateInv*. This allows us to formulate some standard lemmas to be used for each interpretation of the locale. An action A from *ProcActionSet* is a predicate that takes two configurations c, c' and a proc i and returns true if and only if A is a valid step from c to c' executed by process i. Likewise, an action A from *ComActionSet* is a predicate over configurations.

We call *LocalStep* the execution of an action from *ProcActionSet*, and *ComStep* the execution of an action from *ComActionSet*. A step from c to c' happens if and only if there is a *LocalStep* or a *ComStep* from c to c' (see Fig. 2). To verify properties of an algorithm, all possible executions of the algorithm must be inspected. We use the term *Run* for the execution of the algorithm and define it as an infinite sequence of configurations where *InitPred* holds in the initial configuration and every configuration and its successor is in the step relation. Of course there might be configurations where no further step is possible. In this case the system is deadlocked. For these cases where the actual execution would be finite we allow the system to take *stuttering* steps, i.e. to repeat the last configuration until the end of time. (see Fig. 2).

IPC. The given example requires to implement a message passing mechanism to enable the communication between processes. In our model, a message will traverse three states on its way from the sender to the receiver:

– **outgoing**: When a sender wants to send a message (or a set of messages) it puts the message into its outgoing buffer. Messages in the outgoing buffer are still at the senders site, i.e. outgoing messages are lost if the sender crashes.
– **transit**: The message is on its way to the receiver.
A crash of the sender does no longer concern messages that are in transit.

```
record MsgStatus =          record content =
  outgoing :: nat             snd :: nat
  transit  :: nat             rcv :: nat
  received :: nat
```

definition *Msgs* :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme)\ set)$
where
$Msgs\ M \equiv \{m.\ outgoing\ (M\ m) > 0 \vee transit\ (M\ m) > 0 \vee received\ (M\ m) > 0\}$

definition *OutgoingMsgs* :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme)\ set)$ **where** $OutgoingMsgs\ M \equiv \{m.\ outgoing\ (M\ m) > 0\}$

definition *TransitMsgs* :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme)\ set)$ **where** $TransitMsgs\ M \equiv \{m.\ transit\ (M\ m) > 0\}$

definition *ReceivedMsgs* :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme)\ set)$ **where** $ReceivedMsgs\ M \equiv \{m.\ received\ (M\ m) > 0\}$

Fig. 3. Messages

- **received**: The message has already arrived on the receiver's site.
  It is now ready to be processed by the receiver.

We use a multiset-like structure to represent messages in our system, i.e. for every message the number of copies that are **outgoing**, (respectively **transit**, **received**) are stored by mapping messages to records of type *MsgStatus*, a record that can store a number for each option. For such a mapping we use the term message history. The message history is part of each configuration of our algorithm and represents the state of the message evolution. In our model, sender and receiver are stored for each message. The type *Message* will later be extended by the payload of the message (depending on the algorithm we want to model). To work with the set of messages (respectively the set of outgoing messages, of transit messages, of received messages) we define functions that return the respective set for a given message history M (see Fig. 3). Now we are ready to define relations between message histories that describe

- the placement of a message into the outgoing buffer
- the sending of a message, i.e. the change of status from outgoing to transit
- the receiving of a message

Our running example requires to put a set of messages in the outbox. Therefore, we define a directive *MultiSend*: it is a predicate that is true for two message histories M,M' and a set of messages *msgs* if and only if M and M' are equal, except for the **outgoing** values for messages in *msgs*, which are incremented by one in M'. The definitions for changing a message status from **outgoing** to **transit** (*Transmit*) and from **transit** to **received** (*Receive*) are for single

definition $MultiSend$ :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow ('a\ content\text{-}scheme)\ set \Rightarrow bool$ **where**
$MultiSend\ M\ M'\ msgs \equiv M' = (\lambda m.\ if\ (m \in msgs)\ then\ incOutgoing\ (M\ m)\ else\ M\ m)$

definition $Transmit$ :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow ('a\ content\text{-}scheme) \Rightarrow bool$ **where**
$Transmit\ M\ M'\ m \equiv m \in OutgoingMsgs\ M$
$\wedge\ M' = M\ (m := (\!|\ outgoing = (outgoing\ (M\ m)) - (1::nat),\ transit = Suc(transit\ (M\ m)),\ received = received\ (M\ m)|\!))$

definition $Receive$ :: $(('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow (('a\ content\text{-}scheme) \Rightarrow MsgStatus) \Rightarrow ('a\ content\text{-}scheme) \Rightarrow bool$ **where**
$Receive\ M\ M'\ m \equiv m \in TransitMsgs\ M$
$\wedge\ M' = M\ (m := (\!|\ outgoing = (outgoing\ (M\ m)),\ transit = transit\ (M\ m) - (1::nat),\ received = Suc(received\ (M\ m))|\!))$

Fig. 4. Message movements

messages only and straightforward (see Fig. 4). Note that their preconditions '$m \in OutgoingMsgs\ M$' ('$m \in TransitMsgs\ M$') are necessary to rule out a decrement of zero accidentally generating infinitely many messages.

Rotating Coordinator Model. For the concrete model of our algorithm, we need to define a data structure *configuration* that contains all relevant information of the system for one point in time. We already identified message histories as one important part of every configuration. Of course, processes can take steps and change their local state without changing the message history (for example, by doing local computations or processing received messages). Hence, another important component of a configuration must be the local states of processes. In our case, the state of a process p_i can be defined by five variables:

r	- The round of p_i.
phs	- The phase of p_i, i.e. a next-step indicator.
x	- The value of the variable x_i (cp. listing 1.1)
$crashed$	- A flag showing whether p_i has crashed.
$decision$	- A value that is set from $\perp$ to v value when p_i decides v.

We use the value *P1* in *phs* if the process is in line 3 and *P2* if it is in line 4 (c.p. listing 1.1). We do not need more values, as being in line 6 can be detected by testing if $r > n$ (where n is the number of processes) and the remaining lines are just initialization of variables that will be modeled by the respective *InitPred*.

As in [FMN07], we use the term *program counter* for the pair (r, phs). In a configuration, we have to store a process state entry for each process. Hence, we use a mapping from processes to process states as another component of configurations. As explained above, the message type must be extended by the respective payload. In our case, the content a process has to send is its current x

```
record process-state =                    record msg = Message +
  r          :: nat                         cnt-v :: Input
  phs        :: Phase
  x          :: Input                     record configuration =
  crashed    :: bool                        St :: proc ⇒ process-state
  decision   :: Input option                Me  :: msg ⇒ MsgStatus
```

Fig. 5. Definition: *configuration*

```
definition
  p1msgset :: proc ⇒ Input ⇒ msg set where
  p1msgset i v ≡ {m. ∃j ∈ procs.
        m = (|snd = i, rcv = j, cnt-v = v|) }

definition
  MsgGen :: configuration ⇒ configuration ⇒ proc ⇒ bool where
  MsgGen c c' i ≡ crashed (St c i) = False
  ∧ phs (St c i) = P1 ∧ r (St c i) ≤ N
  ∧ St c' = (St c) (i := (|
  r = r (St c i),
  phs = P2,
  x = x (St c i),
  crashed = False,
  decision = decision (St c i)  |))
  ∧ (if (r (St c i) = PID i) then
    MultiSend (Me c) (Me c') (p1msgset i (x (St c i)))
  else
    Me c' = Me c)
```

Fig. 6. Definition: Action *MsgGen*

value, here of type *Input*. Hence, the message type is extended by a field *cnt-v*. As a result of these considerations, we get the type for a configuration as a record consisting of an array of process states and a message history (see Fig. 5).

Next, we define the communication and process actions. Regarding Listing 1.1 after the initialization, a process p_i checks whether it is itself the coordinator of its current round. If so, then it sends a set of messages with $p_i i$ as the sender, the current value of x at p_i as the content, and all processes as the receivers. *p1msgset* in Fig. 6 constructs such a set for the respective arguments p_i and x.

Based on this definition, for example, the definition of action *MsgGen* is read as follows: A step from c to c' is a *MsgGen* step taken by p_i if and only if

– p_i is not crashed in c
– *phs* of p_i is *P1* in c and *P2* in c'
– r of p_i is less or equal than N and is not changed to c'

- x, *crashed* and *decision* of p_i do not change from c to c'
- states of all other processes do not change
- a multisend of the respective *p1msgset* happens from c to c' if the current round of the process equals its process id (PID).

We use the Isabelle function update syntax to implement the desired behaviour: $St\ c' = (St\ c)\ (i := (\!|\ X\ |\!)\ )$ denotes the update in $(St\ c)$ at i to X to yield $(St\ c')$. Thanks to currying in Isabelle, *crashed* $(St\ c\ i)$ denotes the *crashed* variable of process p_i's state in configuration c. *MsgRcvTrust* trusts an awaited sender and receives its message, while *MsgRcvSuspect* suspects a sender by no longer waiting for its message. *Finish* implements the decision step of a process and *Crash* disables a process by setting its *crashed* variable. There are also the two communication actions *MsgSend* and *MsgDeliver* which push messages one step further (from **outgoing** to **transit**, respectively from **transit** to **received**).

For runs, *initial* configurations are those where, for all processes p_i, (1) (*r,phs*) is (*1,P1*), (2) x is the input of p_i, (3) (*crashed,decision*) is (false,$\bot$) and (4) for all messages **outgoing**, **transit** and **received** are set to 0.

We define two sets *ProcActions* and *Networkactions* and write down the interpretation of the introduced locale *Algorithm* as *RotCoord* with *St* as the required mapping from configuration and processes to process states (given in Fig. 7). Note that proofs for the finiteness of *ProcActions* and *NetworkActions* and for the assertions about process states are required. Two lemmas *StateInv1* and *StateInv2* show that the locale assumption *StateInv* is satisfied.

```
type-synonym procAction = configuration ⇒ configuration ⇒ proc ⇒ bool

definition
  ProcActions :: procAction set where
  ProcActions ≡ {MsgGen, MsgRcvTrust, MsgRcvSuspect, Crash, Finish}

type-synonym networkAction = configuration ⇒ configuration ⇒ bool

definition
  NetworkActions :: networkAction set where
  NetworkActions ≡ {MsgSend, MsgDeliver}

interpretation RotCoord: Algorithm
  Init
  ProcActions
  NetworkActions
  St
  by (unfold-locales,
    auto simp add: ProcActions-def NetworkActions-def StateInv1 StateInv2)
```

Fig. 7. Interpretation *RotCoord*

3 Requirement Specification

Validity and Agreement (see the Introduction) are safety properties; as pure invariants of the algorithm they can be formulated as state predicates. Hence, for Validity and Agreement, it would be sufficient to reason about individual states, and whether the properties are preserved by every transition. Termination is a liveness property; it requires us to consider full runs as first-class entities.

We do not include a formulation of Validity in this paper. Instead, next to the formalizations of Agreement and Termination (see Fig. 8), we explicitly mention Irrevocability for decisions, i.e, that decisions cannot be undone or overwritten.

lemma *Irrevocability*: **assumes** R: *Run R* **and**
  d: *decision* $(St\ (R\ t)\ i) \neq$ *None* **and** z: $z \geq t$
  shows
  the(*decision*$(St\ (R\ z)\ i)) =$ *the* (*decision*$(St\ (R\ t)\ i))$
  decision $(St\ (R\ z)\ i) \neq$ *None*

theorem *Agreement*: **assumes** R: *Run R* **and**
  di: *decision*$(St\ (R\ t)\ i) \neq$ *None* **and**
  dj: *decision*$(St\ (R\ t)\ j) \neq$ *None*
  shows *the*(*decision*$(St\ (R\ t)\ i)) =$ *the* (*decision*$(St\ (R\ t)\ j))$

theorem *Termination*: **assumes** R: *Run R* **and** i: $i \in$ *Correct R*
  shows $\exists\, t.$ *decision* $(St\ (R\ t)\ i) \neq$ *None*

Fig. 8. Irrevocability, Agreement and Termination

4 Proof Techniques

'Inspection of the Code'. To show that an algorithm exhibits certain properties, we need to refer to its 'code'. Since pseudo code has no formal semantics, this kind of reference cannot be formal. The reader has to believe that certain basic assertions are implied by single lines of the code; e.g., if line 27 states $x := 5$ then, after line 27 is executed by p_i, variable x will indeed have value 5. Reasoning is done by 'inspection of the code'. For distributed algorithms, this kind of local reasoning is of course error-prone, because one might assume $x = 5$ when executing line 28, which might be wrong if x is shared and another process changes x while p_i moves from line 27 to 28. In [FMN07], the reference to local pseudo code is replaced by the reference to formally-defined global transition rules. Then, if some rule A is provably the only one that changes the variable x of process p_i and process p_i's variable has changed from time t_x to t then, obviously "by inspection of" the rules, we can infer that A was executed between t_x and t. Such a setup is a useful basis for its application within a theorem prover.

Invariant-Based Reasoning. This well-known technique boils down to the preservation of properties from one configuration to another during computation steps: essentially, it requires a proof by case analysis for all possible actions in such a step. Here, the formal version of 'inspection of the code' is pertinent. Finding a proof that an invariant property holds in some initial configuration leads to the standard proof technique of induction over time t, i.e., along the configurations of a run, which we evidently use a lot in our examples.

History-Based Reasoning. Reasoning along the timeline gets more difficult if also assertions about the past are made. Showing that p_i received a message m on its way to configuration c would require to inspect every possible prefix of a run, unless there is some kind of bookkeeping implemented in the model. In [FMN07], this problem is solved by the introduction of history variables [Cli73, Cla78, Cli81] that keep track of events during the execution of the algorithm. For verification purposes of concurrent programs, history variables are common (see [GL00, Owi76]). Technically, we make history variables an explicit part of our model that also serves for the needed IPC. This provides access to the entire communication history. Every sent message is stored in the history and will not be deleted during a run. Hence, when inspecting a configuration $(R\ t)$, all messages sent before t are accessible. Therefore, the above-mentioned assertion can be reduced to the simple check that m is in the message history of c.

Application of Proof Techniques. As Validity is an invariant, the technique for invariant-based reasoning is applied. For the used induction principle, it is to show that Validity holds in the initial state of every run and every step of the algorithm preserves it. Hence the main part of the proof is a classical example for the most-used proof technique mentioned above: fix a run R and a time t, then perform induction on t. As a consequence, the remaining proof goals are:

– show that for every initial configuration $R(0) \in \mathbf{Init}$ $P(R(0))$ holds.
– show that $P(R(t))$ implies $P(R(t{+}1))$.

Mostly, the first goal is implied by the definition of the initial states **Init**. The second goal requires that every defined transition rule preserves P; the induction hypothesis can be strengthened by the knowledge that the step $R(t) \rightarrow R(t{+}1)$ is derived by exactly one application of one of the defined transition rules (distinction of cases). Hence, it remains to show that every application of some rule leads to a successor configuration $R(t{+}1)$ with $P(R(t{+}1))$.

The main argument for Agreement in our running example is that processes cannot skip messages of trusted-immortal processes. Let ti be a trusted-immortal process. We sketch the proof that every process has the same value stored in x before entering a round higher than *PID ti* (where *PID* is a function that returns a unique process id from $1..N$ for every process): Every process j that decides a value must traverse every round number between 1 and n and, therefore, also the round number *PID ti* where ti is the coordinator. Since j cannot skip the message of a trusted-immortal process, j has to assign the value v_{ti} of ti to its state variable x before entering round *(PID ti) + 1*. Therefore, afterwards, all processes j in higher rounds than *PID ti* will send the value v_{ti} or nothing (if they crash before) and, hence, processes can only apply value v_{ti} in such rounds. Formally, this is expressed by the Lemma *uniformRndsAfterTI2* asserting that two processes in rounds higher than the process id of some trusted-immortal process must have the same x value. This implies agreement since every process that decides, decides for its x value and must be in round $n{+}1$ and $n{+}1$ is greater than every process id. Many more invariants must be derived to prove this lemma and both introduced proof techniques are applied in multiple steps.

The proof for Termination appears to be, at first sight, quite obvious: Processes can only block while waiting for messages of trusted-immortal processes. We sketch the proof how mutual waiting is ruled out. Let ti_1 and ti_2 be two trusted-immortal processes waiting for each other's messages. Without loss of generality, let $PID\ ti_1 \leq PID\ ti_2$. Since ti_1 waits for the message of ti_2, it must be in round $PID\ ti_2$ and therefore in a round greater than or equal to its own round. Thus, ti_1 already must have sent the message for round $PID\ ti_1$ and therefore ti_2 must eventually receive this message.

In our formal model, this proof is quite more difficult. The suggested proof relies on the implicit fairness assumption that every possible process step and every possible message transition from **outgoing** to **transit** and from **transit** to **received** will eventually happen, which is implicitly implied by the model.

Nevertheless, in the given example, it is possible to give a formal proof without introducing further fairness assumptions about the execution of *ProcActions*: since we allow *FinalStuttering* only if no further defined actions are possible, in runs with *FinalStuttering*, there can be no infinitely enabled actions. Hence, one can prove that every action that is enabled either gets disabled later or is executed later on. The proof of Termination therefore relies on proving two assertions: every run of the algorithm has *FinalStuttering* and, at the beginning of the *FinalStuttering*, a correct process is in round number $n{+}1$ and therefore has decided (otherwise the action *Finish* would still be enabled).

5 More Case Studies

Our much more complicated case studies are two widely known Consensus algorithms: (1) one by Chandra and Toueg [CT96] (thus, from now on referenced as CT) that uses the failure detector $\diamond S$—known as the weakest that allows to solve Consensus—and (2) Paxos, by Lamport [Lam98]. The latter does not satisfy Termination, but it does not need failure detectors.

A formal review of both algorithms is found in [Fuz08]. Compared to the running example (Listing 1.1), both cases are much more complex caused by the weaker assumptions on the asynchrony of the environment. For each algorithm, we required approximately 10k LOC in Isabelle/HOL[2]. The basic model and the proof techniques are essentially the same except for a few mentionable details.

[2] Full models and proofs can be found at
`http://www.mtv.tu-berlin.de/fileadmin/a3435/Isa/CT.zip`
`http://www.mtv.tu-berlin.de/fileadmin/a3435/Isa/Paxos.zip`

In the Rotating Coordinator algorithm all n processes decide in the same round (in round $n+1$), while in Paxos and CT processes might decide in different rounds; also, there is no upper bound for the traversed round numbers. Moreover, processes can decide values broadcast in different rounds. Hence, we need some kind of global view on the system, i.e. to consider whole configurations. For this purpose, already [CT96] introduce the notion of *Locked Values*. A value is locked for a round r if more than the half of all processes acknowledged the value sent by the coordinator of r. A central lemma for both algorithms states that if v_1 and v_2 are locked values in rounds r_1 and r_2 then $v_1 = v_2$. Inspired by the proof sketches in [CT96], Fuzzati et al. [FMN07] use induction on the round number to prove this lemma: To prove a proposition P holds for all rounds r' with $r' \geq r$ the first step is to show P holds for $r = r'$. In the inductive step, P is shown for round k under the assumption that P holds for all r' with $r \leq r' < k$.

Regarding the timeline, this approach dissents from standard temporal reasoning techniques, as the 'global' round number does not proceed consistently with the global clock. In fact, the round number might be different in all local states of the processes and can evolve independently from the global progress as long as it is monotonically increasing; it is possible that a round number r_i of process p_i is greater than the round number r_j of a process j and later in time $r_i < r_j$ holds. Therefore, proofs done by this technique are intricate, hard to follow by a reader, and not preferred for doing formal proofs. This is documented by errors that we found in [FMN07, Fuz08] (see below). Making such errors within a theorem proving environment is not possible and, hence, we were forced to correct them.

Another difference due to the complexity of Paxos and CT is that there are different types of messages within single rounds; hence, we get dependencies of messages. For example, if message m_2 depends on the prior reception of message m_1, we can deduce that, if p_i sent m_2 to p_i, it must have received m_1. Moreover, the sender of m_1 must indeed have sent m_1. Thus, new proof patterns arise for dependencies between and also concerning their contents.

One important contribution of the mechanizing proofs is the awareness that even proofs at such a formal level as [Fuz08] can exhibit severe faults without being noticed. During our work, we found several problems both in the model and the proofs. The major problems we found in the proof for Paxos of [Fuz08] are:

- There was an error in the broadcast mechanism that circumvented a delivery of broadcasts to all processes except for its sender and therefore would render executions, where only the minimal majority of processes are alive when the first process decides, nonterminating. Moreover an assumption about the mechanism claimed that every broadcast will eventually be received by all correct processes. Due to the error mentioned before this is in contradiction to the transition rules. Of course from this contradiction one could derive any property needed.
- Another problem concerned the basic orderings that are introduced for the reasoning on process states. It turned out that the ordering does not fullfill

the required monotoncity in time that was assumed. Since many proofs for the following lemmas relied on this ordering, this problem is serious.
- The proof for one of the central lemmas (Locking Agreement) is wrong. It uses another lemma (Proposal Creation), but its assumptions are not fullfilled. Therefore, we had to find an adequate version of this lemma with weaker assumptions and redo both proofs (a similar error occurs in [FMN07]).

6 Conclusion

Exemplified with Consensus algorithms, we show how to represent their widespread informal and incomplete pseudo-code descriptions instead in a formal and complete way that can be processed within a theorem prover. It is not our intention to suggest algorithm designers shall start with pseudo code; we rather show how given pseudo descriptions can be formalized. Furthermore, we may thus point out alternative algorithm representations that *can* be formalized in theorem proving environments. By intention, our approach (continuing our previous 'pencil-and-paper' work [FMN07]) is close to the well-known abstract state machines from Gurevich[Gur93], and also actions in the TLA-format, as it is our goal to achieve formalizations of algorithms and their proofs that are reasonably close to the intuitions of typical researchers in the field. The formalization usually requires to add details to the pseudo code so it hardly ever correponds one-to-one. However, this can also be seen as an advantage as it forces to clarify potential sources of misinterpretation.

When mechanizing the proofs (or rather: previous proof sketches), we tried to stick to the intuitive arguments and proof techniques as much as possible. Hence automatic tools like Sledgehammer or Quickcheck were not used. But, mechanization requires us to write out *all* the details; thereby, it proves that the intuitive reasoning (also in our own previous work) is often enough too sloppy.

We report on three case studies within this paper. (1) Our running example is very simple, as it is based on strong assumptions about the system model. We chose it just as a convenient representative for this paper, as it is impossible to show the more interesting case studies within the space constraints. Still, most of the method can be exemplified with it. We found this example in a process-algebraic setting [FH07], and also wanted to be able to roughly compare the amount of work needed in the two completely different settings. We now believe that the approach of the current paper is more intuitive—and mechanized!—and thus leads to quicker proofs. (2) The CT-algorithm has now, to our knowledge the first mechanically-checked proofs, including Termination. The latter is only possible, as our formal model includes an explicit representation of runs. (3) The Paxos algorithm can, modulo some changes to the model, be seen as a variant of CT. As mentioned before, the work on Disk Paxos in [JM05] is quite similar to our work for safety properties. The main difference is based on the different model that allows us to comfortably prove liveness properties in the case of CT.

References

Bal03. Ballarin, C.: Locales and Locale Expressions in Isabelle/Isar. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 34–50. Springer, Heidelberg (2004)

CDM11. Charron-Bost, B., Debrat, H., Merz, S.: Formal Verification of Consensus Algorithms Tolerating Malicious Faults. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 120–134. Springer, Heidelberg (2011)

Cla78. Clarke, E.M.: Proving the correctness of coroutines without history variables. In: ACM-SE 16, pp. 160–167. ACM, New York (1978)

Cli73. Clint, M.: Program proving: Coroutines. Acta Informatica 2, 50–63 (1973)

Cli81. Clint, M.: On the use of history variables. Acta Informatica 16, 15–30 (1981)

CT96. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43, 225–267 (1996)

FH07. Francalanza, A., Hennessy, M.: A Fault Tolerance Bisimulation Proof for Consensus (Extended Abstract). In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 395–410. Springer, Heidelberg (2007)

FLP85. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (1985)

FMN07. Fuzzati, R., Merro, M., Nestmann, U.: Distributed Consensus, Revisited. Acta Informatica 44(6), 377–425 (2007)

Fuz08. Fuzzati, R.: A Formal Approach to Fault Tolerant Distributed Consensus. PhD thesis, EPFL, Lausanne (2008)

GL00. Gafni, E., Lamport, L.: Disk Paxos. In: Herlihy, M.P. (ed.) DISC 2000. LNCS, vol. 1914, pp. 330–344. Springer, Heidelberg (2000)

Gur93. Gurevich, Y.: Evolving algebras: An attempt to discover semantics (1993)

JM05. Jaskelioff, M., Merz, S.: Proving the correctness of disk paxos. In: The Archive of Formal Proofs (2005), http://afp.sf.net/entries/DiskPaxos.shtml

Lam78. Lamport, L.: The implementation of reliable distributed multiprocess systems. Computer Networks 2(2), 95–114 (1976)

Lam98. Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems 16, 133–169 (1998)

Lam02. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002)

LSP82. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM ToPLaS 4(3), 382–401 (1982)

Lyn96. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Pub. (1996)

NPW02. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)

Owi76. Owicki, S.: A consistent and complete deductive system for the verification of parallel programs. In: Proceedings of STOC 1976, pp. 73–86. ACM (1976)

Sch90. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys 22, 299–319 (1990)

A Temporal Logic for Multi-threaded Programs[⋆]

Salvatore La Torre and Margherita Napoli

Dipartimento di Informatica, Università degli Studi di Salerno, Italy

Abstract. Temporal logics for nested words are a specification formalism for procedural programs, since they express requirements about matching calls and returns. We extend this formalism to multiply nested words, which are natural models of the computations of concurrent programs. We study both the satisfiability and the model-checking problems, when the multiply nested words are runs of multi-stack pushdown systems (Mpds). In particular, through a tableau-based construction, we define a Büchi Mpds for the models of a given formula. As expected both problems are undecidable, thus we consider some meaningful restrictions on the Mpds, and show decidability for the considered problems.

1 Introduction

Temporal logic is a standard specification language in program verification. Traditional linear time temporal logic (Ltl) [14] allows to express ω-regular properties and recent research has enriched this formalism with temporal operators that allow to take into account the call-return structure of the control flow in sequential programs with recursive procedure calls, such as CaRet [2] and Nwtl [1]. In these *call-return* temporal logics, the key intuition is to look at the program computations not simply as a word but as a *nested word*, which is essentially a graph with two kinds of edges: the *linear* edges capturing the sequential structure of computations (total ordering among the program states), and the *call-return* edges, that connect a call to its matching return in the computation (defining a matching relation). The Ltl operators refer only to the total ordering given by the linear edges. The mentioned call-return logics instead present versions of the standard Ltl operators which refer to the call-return edges, and thus properties such as "a procedure A is always invoked through calls of a procedure B" and "a write to a variable x should be followed by a read of x in the same procedure invocation", become expressible.

In concurrent programs communicating through a shared memory and with recursive procedure calls, each thread has its own control flow structured into procedure calls. A suitable model for the computations of such programs are the *multiply nested words*, that is graphs with linear edges and different kinds of call-return edges (one for each thread and each kind defining a different matching relation). Along the line of the call-return temporal logics, we can define extensions of temporal operators that refer to the different types of call-return edges.

⋆ This work was partially funded by the MIUR grants FARB 2010-2011 Università degli Studi di Salerno (Italy).

In this paper, we consider the logic MultiCaRet that extends CaRet to multiply nested words, and study both the satisfiability and the model-checking problems. We model the programs as multi-stack pushdown systems (Mpds). The computations of Mpds naturally define multiply nested words, and thus, the model checking question we ask is: given a MultiCaRet formula φ and a Mpds M, do all the nested words generated by M satisfy φ?

We face these problems using the automata-theoretic approach. In particular, for a formula φ over n-nested words (i.e., multiply nested words with n matching relations), we give a tableau-based construction that defines a Büchi Mpds M_φ which generates all the n-nested words satisfying φ. We show that the size of M_φ is linear in n and exponential in the size of φ. Defining with $\mathcal{L}(M)$ the language of multiply nested words defined by M, satisfiability is reduced to checking the non-emptiness of $\mathcal{L}(M_\varphi)$ and model-checking reduces to checking the emptiness of $\mathcal{L}(M) \cap \mathcal{L}(M_{\neg\varphi})$. Observe that, since the push and pop transitions of a Mpds are visible on a generated multiply nested word, by a standard cross product synchronized on the labels, we can construct a Büchi Mpds that generates the intersection language. Therefore, both the considered problems reduce to solving the emptiness problems for Büchi Mpds.

Unfortunately, the emptiness problem for Mpds is known to be undecidable already with two stacks (two stacks are sufficient to encode the computations of a Turing machine), and this can be used to show that indeed also the MultiCaRet model-checking problem is undecidable. Moreover, we prove that also MultiCaRet satisfiability is undecidable with a simple reduction from PCP. We thus consider some known restrictions that have been recently studied to obtain decidable models of Mpds that are meaningful when dealing with non-terminating computations. In particular, we look at multiply nested words that correspond to Mpds computations where each symbol that is popped has been pushed into a stack during the last k contexts (*scope-bounded multiply nested words*) [11] or where a symbol is popped from stack i if all stacks $j < i$ are empty (*ordered multiply nested words*) [8]. Observe that, both the restrictions do not limit the number of execution contexts where we can do push or pop transitions of any stack and thus are suitable to take into account non-trivial infinite interactions among the different threads of a program.

It is known that the emptiness problem for Büchi Mpds restricted to ordered multiply nested words is decidable in time doubly exponential in the number of stacks and polynomial in the number of states, and is 2Etime-complete [3] (we recall that 2Etime denotes the class of all the decision problems which are solvable by a deterministic Turing machine in time $2^{2^{O(n)}}$). Combining this result with our construction we get that the MultiCaRet satisfiability and model checking problems on ordered multiply nested words are 2Etime-complete.

A further contribution of our paper is to solve the emptiness problem for Büchi Mpds restricted to scope-bounded multiply nested words. We reduce this problem to checking the emptiness of a standard Büchi automaton of size exponential in the number of stacks and the bound k on the scope of the matching relations. The solution relies on defining for each stack a pushdown automaton whose reachable states correspond to *thread interfaces* of dimension at most k (tuples of pairs summarizing the control states when context-switching into and out of a thread along with the information if in a context an accepting state has been seen). The entire contribution of a thread to a k-scoped computation of the MPDS can be summarized with an infinite thread interface which is the composition of thread interfaces of dimension at most k (see [12]). Thus, we define an automaton that nondeterministically selects a thread interface of dimension k (*k-thread-interface*) for each thread and simulates a computation modifying the current state by applying the next pair of one such thread interface. Once a k-thread-interface is completely used, a new k-thread-interface is nondeterministically selected for that thread. In this simulation, we make accepting all the states that are introduced by a pair corresponding to a context where an accepting state has been visited. By the above described reduction, we thus have that the MULTICARET satisfiability and model-checking problems are both decidable in time exponential in the size of the formula, the number of stacks, and the bound k. Since this problems are already EXPTIME-hard for CARET [2], we get EXPTIME-completeness.

As a final contribution, we show that the logic MULTICARET can be expressed in FO, and since MSO is decidable on all classes of MSO-definable multiply nested words of bounded tree-width [13], we also get that MULTICARET is decidable for all such classes. We recall that the class of ordered multiply nested words defined by a MPDS have bounded tree-witdh and are MSO-definable [13], and so do the class of scope-bounded multiply nested words [12], therefore the decidability of MULTICARET satisfiability and model-checking for these classes can also be obtained using these arguments.

Related Work. Besides the already cited research there are few other papers that are related to ours.

A general temporal logic for concurrent programs which subsumes MULTICARET is introduced in [6]. However, the decidability results there are obtained by restricting the computations to a bounded number of *phases* [10], where in each phase pop transitions are all from the same stack. The phase restriction looks quite limiting when considering infinite computations: the last phase is infinite and there is no unbound alternation between popping different stacks, i.e., from some point there is only one stack that is really used. Besides, it is orthogonal to the scope-bounded restriction and does not allow to express more behaviors than placing an ordering on the matching relations [4]. It is not known a relation between these two last restrictions. We also note that the approach followed here is completely different from [6].

Another concurrent extension of CARET is considered in [7]. This logic differ from MULTICARET both in the syntax and the semantics. The model checking question for LTL formulas with respect to ordered MPDS is settled in [3]. The scope-bounded restriction strictly extends the notion of bounded-context switch that have been successfully used in finding bugs in concurrent programs [15].

The scope-bounded restriction on matching relation defined here is more permissive than that introduced in [11]. Here, we do not bind this definition to a round-based organization of computations (a round is a sequence of contexts where each stack is active exactly once). Therefore, while there the number of context-switches between a push and its matching pop is always bounded (nk where n is the number of stacks and k is the bound on the scope of the matching relations), here we can have unboundedly many context switches. As an example, consider a sequence $a_1(a_2a_3b_2b_3)^m b_1$ where a_i is a push and b_i is a pop of the i-th stack. For each m, this satisfies the 2-scoped restriction given in this paper, while it does not satisfy the $2m$-scoped restriction given in [11].

Simultaneously and independently of our research, in [5] the model-checking of linear-time temporal properties (LTL) for MPDS under the scope-bounded restriction has been shown to be EXPTIME-complete. This result can be obtained as a corollary of our results since our logic MULTICARET syntactically subsumes CARET and thus LTL.

2 A Temporal Logic over Multi-nested Words

Given two positive integers i and j, $i \leq j$, we denote with $[i, j]$ the set of integers k with $i \leq k \leq j$, and with $[j]$ the set $[1, j]$.

Multiply Nested Words. In this section, we recall the concept of multiply nested word which a natural formalism for expressing the computations of multi-stack pushdown systems, which in turns carefully capture the flow of control in concurrent programs with shared memory and recursive procedure calls.

Fix an alphabet Σ, an *infinite word* over Σ is a mapping that assigns to each *position* $i \in \mathbb{N}$ a symbol $\sigma_i \in \Sigma$, and is denoted as $\{\sigma_i\}_{i\in\mathbb{N}}$ or equivalently $\sigma_1, \ldots, \sigma_n \ldots$. Each infinite word defines a linear ordering among its positions which corresponds to the ordering induced by the relation $<$ over $\mathbb{N}$. In this paper, we make use only of the infinite words therefore we will refer to them also simply as words. A *multiply nested word* is a word equipped with one or more matching relations. For a word $\{\sigma_i\}_{i\in\mathbb{N}}$, a matching relation expresses a relation between two disjoint sets of its positions, the calls and the returns, such that each call i is matched to at most one return j that follows i in the linear ordering (i.e., $i < j$) and each return i is matched to at most one call j that precedes i in the linear ordering (i.e., $j < i$), and matched calls and returns are well-nested. Formally, a *matching relation* over $\mathbb{N}$ is a triple (μ, C, R) where $C, R \subseteq \mathbb{N}$ (respectively, the set of calls and the set of returns of the relation), $C \cap R = \emptyset$, and $\mu \subseteq C \times R$ is such that for all $i, j, h \in N$:

– $\mu(i, j)$ implies $i < j$ (respects the linear ordering of w);
– $\mu(i, j)$ and $\mu(i, h)$ implies $j = h$ (each call matches at most one return);
– $\mu(i, j)$ and $\mu(h, j)$ implies $i = h$ (each return matches at most one call);
– $i \leq j$, $i \in C$, and $j \in R$ implies that there is a $i \leq k \leq j$ such that either $\mu(i, k)$ or $\mu(k, j)$.

An *n-nested word* w is $(\{\sigma_i\}_{i\in\mathbb{N}}, \{\langle\mu_i, C_i, R_i\rangle \mid i \in [n]\})$ where $C_1, R_1, \ldots, C_n, R_n$ are pairwise disjoint and for $i \in [n]$, μ_i is a matching relation with set of calls

C_i and set of returns R_i. A *multiply nested word* is an n-nested word for some $n \in \mathbb{N}$. A 1-nested word is also called a *nested word* [1].

It is simple to see that a multiply nested word $w = (\{\sigma_i\}_{i\in\mathbb{N}}, \{\langle\mu_i, C_i, R_i\rangle \mid i \in [n]\})$ can be graphically represented as a labeled directed graph $G_{nw} = (\mathbb{N}, E, \lambda)$ where: $\mathbb{N}$ is the set of vertices, E is the set of edges and is defined as the union of all μ_i along with the set $\{(i, i+1) \mid i \in \mathbb{N}\}$ denoting the linear ordering induced by $<$, and the labeling function $\lambda : \mathbb{N} \to \Sigma$ that maps each vertex i to σ_i.

In Fig. 1, we report the initial fragment of a 2-nested word where μ_1 matches each occurrence of a's with an occurrence of b, μ_2 matches each occurrence of c with an occurrence of d, and the occurrences of e stay unmatched. The calls and the returns are the positions of respectively the a'a and the b's for μ_1, and the c's and the d's for μ_2. We use subscripts to stress the position of each symbol. The relation μ_1 is denoted with the full curved edges and the relation μ_2 with dashed curved edges.

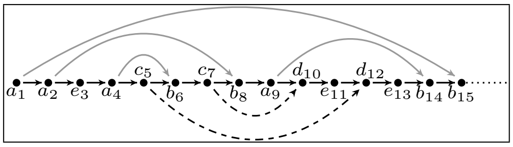


Fig. 1. A fragment of a 2-nested word

Paths in Multiply Nested Words. Different kinds of paths can be defined in multiply nested words depending on the notion of successor which is used. In this section we define the different notions of successors that will be used to give the semantics of the temporal logic we introduce in the next section. These successors are the adaptation to multiply nested words of those defined on nested words for defining the logic CaRet [2].

Fix a multiply nested word $w = (\{\sigma_i\}_{i\in\mathbb{N}}, \{\langle\mu_i, C_i, R_i\rangle \mid i \in [n]\})$. The first kind of successor we consider is the *linear successor* that is defined by the linear order induced by $<$ on $\mathbb{N}$. The linear successor of a position $j \in \mathbb{N}$ is simply $j+1$.

The *abstract successor* over the i-th matching relation of a position j, denoted $next_i^a(j)$, is defined as the abstract successor in the nested word which is obtained by ignoring all the matching relations but the i-th one. In particular, it is the matching return for matched calls, is not defined on unmatched calls and on linear predecessors of matched returns, and is the linear successor in all the other cases (i.e. for positions that are not calls and whose linear successors are not returns). For each position $j \in \mathbb{N}$, $next_i^a(j)$ is: h if $\mu_i(j, h)$ holds; otherwise, $\perp$ (*undefined*) if either $j+1 \in R_i$ and $\mu_i(h, j+1)$ holds for some $h < j$, or $j \in C_i$ and $\mu_i(j, h)$ does not hold for each $h > j$; and $j+1$ in all the remaining cases.

Analogously to the abstract successors, the *call successor* over the i-th matching relation of a position j, denoted $next_i^-(j)$, is defined as the call successor in the nested word which is obtained by ignoring all the matching relations but the i-th one. In particular, it is the largest call h that precedes j and is not matched up to j, if any, and is undefined otherwise. Formally, for each position $j \in \mathbb{N}$, $next_i^-(j)$ is the largest $h < j$ such that $h \in C_i$ and either $\{k \mid \mu_i(h, k)\} = \emptyset$ (h is unmatched) or $\mu_i(h, k)$ holds for some $k > j$ (the call is not matched yet), if any, and $\perp$ otherwise.

The Temporal Logic MultiCaRet. Multiply nested words naturally arise as models of the computations of concurrent threads communicating through shared memory. In this interpretation, the nesting relations capture the call-return relation of the recursive procedure calls within each thread. We will make this more precise in the next section where we will model such computations as multi-stack pushdown systems.

We use multiply nested words to interpret the formulas of the logic MultiCaRet which extends the logic CaRet [2] to express properties of multi-threaded programs. The logic MultiCaRet (CaRet for multiply nested words) has the usual linear temporal logic modalities according to the linear ordering and CaRet modalities indexed over the different matching relations. For each $i \in [n]$, we use the atomic propositions $call_i$ and ret_i to identify the respectively a call and a return of the i-th matching relation of an n-nested word.

Formally, we fix a finite sets of *atomic propositions* AP and $\{call_i, ret_i \mid i \in [n]\}$, for $n \in \mathbb{N}$. The formulas of MultiCaRet are inductively defined as follows:

$$\varphi ::= p \mid call_i \mid ret_i \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi \mid \bigcirc^b \varphi \mid \varphi \mathcal{U}^b \varphi$$

where $i \in [n]$, $b \in \{a_i, -_i \mid i \in [n]\}$ and $p \in AP$.

The constants true and false are defined as abbreviations: $\top \equiv \varphi \vee \neg\varphi$ and $\bot \equiv \varphi \wedge \neg\varphi$. Other common abbreviations are $\Diamond^b\varphi \equiv \top\mathcal{U}^b \varphi$ and $\Box^b\varphi \equiv \neg\Diamond^b\neg\varphi$. The semantics of the logic operators is given as usual. Each of the introduced temporal operators correspond to one of the given notions of successor. In particular, the *global* modalities $\bigcirc$ and $\mathcal{U}$ refer to the linear successor, the *abstract* modalities $\bigcirc^{a_i}$ and $\mathcal{U}^{a_i}$ to the abstract successor, and the *call* modalities $\bigcirc^{a_i}$ and $\mathcal{U}^{a_i}$ to the call successor. Formally, fix an n-nested word $w = (\{\sigma_i\}_{i \in \mathbb{N}}, \{\langle \mu_i, C_i, R_i\rangle \mid i \in [n]\})$ over the alphabet 2^{AP}. The truth value of a formula w.r.t. a position $i \in \mathbb{N}$ in w is defined as follows:

- $(w, i) \models p$ iff $p \in \sigma_i$ (where $p \in AP$);
- $(w, i) \models call_j$ (resp. ret_j)iff $i \in C_j$ (resp. $i \in R_j$);
- $(w, i) \models \neg\varphi$ iff $(w, i) \models \varphi$ does not hold;
- $(w, i) \models \varphi_1 \vee \varphi_2$ iff either $(w, i) \models \varphi_1$ or $(w, i) \models \varphi_2$:
- $(w, i) \models \bigcirc\varphi$ iff $(w, i+1) \models \varphi$;
- $(w, i) \models \bigcirc^{b_j}\varphi$ (with $b \in \{a, -\}$) iff $next^b_j(i) \neq \bot$ and $(w, next^b_j(i)) \models \varphi$;
- $(w, i) \models \varphi_1\mathcal{U}\varphi_2$ iff there exists a $h \geq i$ such that $(w, h) \models \varphi_2$ and $(w, k) \models \varphi_1$ for all $k \in [i, h-1]$;
- $(w, i) \models \varphi_1\mathcal{U}^{b_j}\varphi_2$ (with $b \in \{a, -\}$) iff there exists a $h \in \mathbb{N}$ such that $(w, h) \models \varphi_2$, $i = x_1, x_2, \ldots, x_m = h$ where $x_{k+1} = next^b_j(x_k)$ for $k \in [m-1]$, and $(w, x_k) \models \varphi_1$ for $k \in [m-1]$.

We say that a multiply nested word w *satisfies* a formula φ, written $w \models \varphi$, if $(w, 1) \models \varphi$.

Satisfiability. The satisfiability problem for MultiCaRet formulas is defined as the problem of determining if given a MultiCaRet formula φ there exists a multiply nested word w such that $w \models \varphi$. This problem turns out to be undecidable already for formulas using only two matching relations. A proof of this can be obtained by reducing the*Post's Correspondence Problem* (PCP). Given a set of pairs (u_i, v_i), $i \in [m]$, where $u_i, v_i \in \Sigma^*$ for a finite alphabet Σ, the PCP consists of determining if there is a sequence of indices $i_1, \ldots, i_h$ such that $u_{i_1} \ldots u_{i_h} = v_{i_1} \ldots v_{i_h}$.

The reduction consists of writing a formula that is satisfied only on multiply nested words $(\{\sigma_i\}_{i\in\mathbb{N}}, \{\langle \mu_i, C_i, R_i\rangle \mid i \in [2]\})$ where denoting the word $\{\sigma_i\}_{i\in\mathbb{N}}$ as the concatenation $\alpha\beta\gamma$: $\alpha \in (\bigcup_{i\in[m]}(i.u_i.v_i))^*$ and $\beta \in \{\sigma.\sigma \mid \sigma \in \Sigma\}^*$; each position of the α part is in C_1, if it is labeled with a piece of a u_i, and is in C_2, if it is labeled with a piece of v_i; the positions in β are alternately in R_1 and R_2; none of the positions of the γ part is either a call or a return; there are no unmatched calls and returns; for $i \in [2]$, if $\mu_i(x, y)$ holds then $\sigma_x = \sigma_y$. It is simple to write a MultiCaRet formula φ_{PCP} that checks all the above properties. By a simple proof one can prove that the considered PCP instance admits a solution iff φ_{PCP} is satisfiable. Moreover, the formula is parameterized on the PCP instance, therefore, this construction reduces the PCP to MultiCaRet satisfiability using only two matching relations:

Theorem 1. *The* MultiCaRet *satisfiability problem is undecidable already with two matching relations.*

Expressing Properties of Multi-threaded Programs. The main motivation for MultiCaRet is to introduce a suitable temporal logic for multi-threaded programs. MultiCaRet is the natural extension of CaRet with abstract and caller modalities over many matching relations, and thus can also capture the usual linear time temporal logic properties. A typical correctness requirement that can be expressed in MultiCaRet consists of the pairs of pre- and post-conditions that must be fulfilled by procedure invocations within each computation. For instance, we can require that a procedure A must satisfy a pre-condition p_A upon invocation and a post-condition q_A on returning from a call (note that as an additional correctness requirement this also implies that it must return from each call), and this has to hold for each thread. We can express this as: $\Box \bigwedge_i[(call_i \wedge p_A) \to \bigcirc^{a_i} q_A]$. Variations of such property requiring different pre/post-conditions for different threads, or limiting the request only to some threads, or admitting that some call may be not returned, can be easily designed.

Additional correctness properties can be required when two procedures are simultaneously invoked in different threads:

$$\Box \bigwedge_{i\neq j}[(\bigcirc^{-i} p_A \wedge \bigcirc^{-j} p_B) \to (\bigcirc^{-i} \bigcirc^{a_i} q_A \wedge \bigcirc^{-j} \bigcirc^{a_j} q_B)].$$

The temporal modalities based on the call successors allows to express properties on the contents of the stacks, which can be used to specify a variety of security properties. For instance, the requirement that a procedure A should be invoked only when in each thread i a call to a procedure B_i is still pending and no overriding call to procedure C_i is happening can be expressed by the formula $\Box[(\bigvee_i call_i \wedge p_A) \to \bigwedge_i(\neg p_{C_i} \mathcal{U}^{-i} p_{B_i})]$.

3 MultiCaRet Model-Checking

In this section we first recall the definition of multi-stack pushdown systems and show how their runs define multiply nested words. Then we consider the MultiCaRet model checking problem.

Multistack Pushdown Systems. A multi-stack pushdown system consists of a finite control along with one or more stacks and is equipped with a labelling function of its states. The system can push a symbol on any of its stacks, or pop a symbol from any of them, or just change its control location by maintaining unchanged the stack contents. Thus there are several push/pop functions (one for each stack) and one internal action function. We also allow pop transitions on empty stack to take into account the unmatched returns. This is modeled with a bottom-of-the-stack symbol $\gamma^\perp$ which is never removed from the stack.

Let $n \in \mathbb{N}$. A *n-stack pushdown system* (n-Mpds) M is a tuple $(Q, Q_0, \Gamma \cup \{\gamma^\perp\}, \Sigma, \lambda, \delta^{int}, \{(\delta_i^{push}, \delta_i^{pop})\}_{i\in[n]})$ where Q is a finite set of control states, $Q_0 \subseteq Q$ is the set of initial states, Γ is a finite stack alphabet, $\gamma^\perp$ is the bottom-of-the-stack symbol, Σ is the alphabet of the state labels, $\lambda : Q \to \Sigma$ is a labelling function, $\delta_{int} \subseteq (Q \times Q)$ is a set of internal transitions and, for every $i \in [n]$, $\delta_i^{push}, \subseteq (Q \times \Gamma \times Q)$ and $\delta_i^{pop} \subseteq (Q \times \Gamma \cup \{\gamma^\perp\} \times Q)$ are respectively push and pop transitions involving the i'th stack. A Pds is a n-Mpds with $n = 1$.

A *configuration* of M is a tuple $\mathcal{C} = \langle q, \{w_i\}_{i\in[n]}\rangle$, where $q \in Q$ is the state of the configuration and each $w_i \in \Gamma^*.\{\gamma^\perp\}$ is the content of the i'th stack. Moreover, $\mathcal{C}$ is *initial* if $q \in Q_0$ and $w_i = \gamma^\perp$ for every $i \in [n]$. Let $Act = \bigcup_{i\in[n]}\{push_i, pop_i\} \cup \{int\}$ be the set of all *actions* of M. A transition between two configurations over an action $act \in Act$ is defined as follows:

$\langle q, \{w_i\}_{i\in[n]}\rangle \xrightarrow{act}_M \langle q', \{w'_i\}_{i\in[n]}\rangle$ if one of the following holds for some $i \in [n]$

[Internal] $act = int$, $(q, q') \in \delta_{int}$, and $w'_h = w_h$ for every $h \in [n]$.
[Push] $act = push_i$, $(q, \gamma, q') \in \delta_i^{push}$, $w'_i = \gamma.w_i$, and $w'_h = w_h$ for $h \in ([n]\setminus\{i\})$.
[Pop] $act = pop_i$, $(q, \gamma, q') \in \delta_i^{pop}$, $w'_h = w_h$ for $h \in ([n] \setminus \{i\})$, and either $w_i = \gamma.w'_i$ or $w_i = w'_i = \gamma = \gamma^\perp$.

A *run* ρ of M is a possibly empty sequence of transitions $\mathcal{C}_0 \xrightarrow{act_1} \mathcal{C}_1 \xrightarrow{act_2} \ldots$. Furthermore, ρ is a *computation* of M if $\mathcal{C}_0$ is initial.

MultiCaRet *Model-Checking.* Each run $\rho = \mathcal{C}_0 \xrightarrow{act_1} \mathcal{C}_1 \xrightarrow{act_2} \ldots$ of M defines a multiply nested word $\langle\langle\rho\rangle\rangle = (\{\sigma_s\}_{s\in\mathbb{N}}, \{\langle\mu_i^\rho, C_i^\rho, R_i^\rho\rangle \mid i \in [n]\})$ with $\sigma_s = \lambda(q_s)$, where q_s is the state of the configuration $\mathcal{C}_s$ in ρ, $C_i = \{s \mid act_s = push_i\}$, $R_i = \{s \mid act_s = pop_i\}$ and $\mu_i^\rho(s,t)$ holding true if the t'th transition of ρ pops the symbol pushed on stack i at the s'th transition. With $\mathcal{L}(M)$ we denote the language $\{\langle\langle\rho\rangle\rangle \mid \rho \text{ is a computation of } M\}$. Moreover, we say that M satisfies a MultiCaRet formula φ, written $M \models \varphi$, if $w \models \varphi$ holds for each $w \in \mathcal{L}(M)$. Thus, the *model-checking* problem for Mpds and MultiCaRet formulas is:

Given a Mpds *M and a* MultiCaRet *formula φ, does $M \models \varphi$?*

From the undecidability of reachability of Turing machines and the fact that a 2-Mpds can simulate a Turing machine, the following theorem holds.

Theorem 2. *The model-checking problem for* 2-Mpds *and* MultiCaRet *formulas is undecidable.*

Büchi Mpds. For a Mpds M, a *Büchi condition* is a subset $\mathcal{F}$ of the set of states of M. A Büchi Mpds is a Mpds along with a Büchi condition. Denoting with $\mathcal{F}^{\mathcal{C}}$ the set configurations of M of the form $\mathcal{C} = \langle q, \{w_i\}_{i\in[n]}\rangle$ where $q \in \mathcal{F}$, we say that a run $\rho = \mathcal{C}_0 \xrightarrow{act_1} \mathcal{C}_1 \xrightarrow{act_2} \ldots$ is accepted by the Büchi Mpds $(M, \mathcal{F})$ (or equivalently satisfies a Büchi condition $\mathcal{F}$) if for infinitely many s, $\mathcal{C}_s \in \mathcal{F}^{\mathcal{C}}$. If M is a Büchi Mpds, we extend the notation $\mathcal{L}(M)$ by requiring that the nested words in $\mathcal{L}(M)$ also satisfy the Büchi condition.

The problem of determining the existence of an accepting run for a given Büchi Mpds (*emptiness problem*) is in general undecidable, again from the undecidability of reachability problem for Turing machines.

4 Büchi Mpds for MultiCaRet Formulas

In this section, we give a tableau-based construction of a Büchi Mpds which generates the multiply nested words satisfying a given MultiCaRet formula. We fix a formula φ over the set of atomic propositions $AP \cup \{call_i, ret_i \mid i \in [n]\}$, for $n \in \mathbb{N}$, and denote with top_i, for $i \in [n]$, a new atomic proposition. The closure of φ, denoted cl_φ, is the smallest set of formulas that contains φ, $\neg\varphi$, top_i, $call_i$ and ret_i for $i \in [n]$, and satisfies the properties described in Fig. 2 (where $\neg\neg\psi$ is identified with ψ and $b \in \{a_i, -_i \mid i \in [n]\}$).

1. $\neg\psi \in cl_\varphi$ if and only if $\psi \in cl_\varphi$
2. if $\psi_1 \vee \psi_2 \in A$ if and only if $\psi_1 \in A$ or $\psi_2 \in A$
3. if $\bigcirc\psi \in cl_\varphi$ or $\bigcirc^b\psi \in cl_\varphi$ then $\psi \in cl_\varphi$
4. if $\psi_1\,\mathcal{U}\,\psi_2 \in cl_\varphi$ then $\psi_1, \psi_2, \bigcirc(\psi_1\,\mathcal{U}\,\psi_2) \in cl_\varphi$
5. if $\psi_1\,\mathcal{U}^b\,\psi_2 \in cl_\varphi$ then $\psi_1, \psi 2, \bigcirc^b(\psi_1\,\mathcal{U}^b\,\psi_2) \in cl_\varphi$

Fig. 2. Properties defining cl_φ

An atom A of φ is a maximal and logically consistent subset of cl_φ (see Fig. 3). We denote the set of all atoms of φ as $Atoms_\varphi$, and the set of all atoms that contain a formula of the form $\bigcirc^{a_i}\psi$ or $\psi_1\mathcal{U}^{a_i}\psi_2$, for some $i \in [n]$, as $Atoms^a_\varphi$. Calls and returns of the i-th matching relation are identified by atoms containing respectively $call_i$ and ret_i.

1. for each $\psi \in cl_\varphi$, either $\neg\psi \in A$ or $\psi \in A$
2. A contains at most one among $\{call_i, ret_i \mid i \in [n]\}$
3. $\psi_1 \vee \psi_2 \in A$ iff $\psi_1 \in A$ or $\psi_2 \in A$
4. $\psi_1\,\mathcal{U}\,\psi_2 \in A$ iff $\psi_2 \in A$ or $\psi_1, \bigcirc(\psi_1\,\mathcal{U}\,\psi_2) \in A$
5. $\psi_1\,\mathcal{U}^b\,\psi_2 \in A$ iff $\psi_2 \in A$ or $\psi_1, \bigcirc^b(\psi_1\,\mathcal{U}^b\,\psi_2) \in A$
6. if $\bigcirc^{a_i}\psi \in A$ then $call_i \in A$.

Fig. 3. An atom A of φ

The main idea in the construction of an Mpds M_φ which defines the set of models of φ, is to mimic the labeling of each position of a multiply nested word with the atom of φ that contains exactly all the formulas of cl_φ that are fulfilled from there. Therefore, the states of M_φ are exactly the atoms of φ. The state labelling function of M_φ labels each atom A of φ with $A \cap AP$ (the set of atomic propositions contained in A).

The set of initial states $Atoms^0_\varphi$ contains all the atoms A of φ such that $\varphi \in A$, and for all $i \in [n]$, $top_i \in A$ and no formula of the form $\bigcirc^{-i}\psi$ belongs to A. Actually, each atomic proposition top_i is used to mark the top positions for the

i-th matching relation, i.e., the positions that are not in between a call of the i-th relation and its matching return. (Observe that all the top_i positions form an infinite sequence of linearly ordered positions that are related by the abstract successor $next_i^a$, and among all the maximal such sequences -paths- this is the only one which is infinite.)

Besides the bottom-of-the-stack symbol $\gamma^\perp$, for the stack symbols, we take only the atoms that contain sub-formulas of the form $\bigcirc^{a_i}\psi$ or $\psi_1\mathcal{U}^{a_i}\psi_2$ and a new symbol ∂. This symbol is never popped from a stack and is used as a placeholder for unmatched calls.

The transition functions are defined such that states and stack contents are consistently updated to ensure the correct propagation of the next modalities and the correct labeling with the top_i propositions.

- Each internal transition (A, A') is such that: $\bigcirc\psi \in A$ iff $\psi \in A'$ (global formulas propagation) and for each $i \in [n]$: $call_i, ret_i \notin A$ (internal moves are not from calls or returns), $top_i \in A$ iff $top_i \in A'$ (top_i status is preserved), $\bigcirc^{a_i}\psi \in A$ iff $\psi \in A'$ (abstract formulas propagation), and A and A' contain the same formulas of the form $\bigcirc^{-_i}\psi$ (call formulas propagation). Moreover, if $ret_i \in A'$ and $top_i \notin A$, then A must not contain any formula of the form $\bigcirc^{a_i}\psi$ (undefined i-abstract successor of A).
- Each push transition (A, B, A') of stack $i \in [n]$ satisfies the following. $call_i \in A$ (push transitions are from calls), $\bigcirc\psi \in A$ iff $\psi \in A'$ holds (global formulas propagation) and for $j \neq i$: $top_j \in A$ iff $top_j \in A'$ (top_j status is preserved on i push transitions), A and A' contain the same formulas of the form $\bigcirc^{-_j}\psi$ (same j-call successor), $\bigcirc^{a_j}\psi \in A$ iff $\psi \in A'$ (propagation of j-abstract formulas). Moreovoer, $\bigcirc^{-_i}\psi \in A'$ iff $\psi \in A$ (i-call formulas update), and if $B \neq \partial$ (i.e. the call is matched), then: $top_i \in A$ iff $top_i \in B$ and $\neg top_i \in A'$, A and B contain the same formulas of the form $\bigcirc^{-_i}\psi$, and $\bigcirc^{a_i}\psi \in A$ iff $\psi \in B$. Otherwise, i.e., $B = \partial$ (the call is not matched), $top_i \in A$ and $top_i \in A'$ (unmatched calls are all at top positions), and A does not contain formulas of the form $\bigcirc^{a_i}\psi$.
- Each pop transition (A, B, A') of stack $i \in [n]$ is such that: $ret_i \in A$ (pop transitions are from returns); $B \neq \partial$ (∂ cannot be popped out of any stack); $\bigcirc\psi \in A$ iff $\psi \in A'$ (global formulas are propagated from A);
  for $j \neq i$: $top_j \in A$ iff $top_j \in A'$ (top_j status is preserved on i pop transitions), A and A' contain the same formulas of the form $\bigcirc^{-_j}\psi$ (same j-call successor), $\bigcirc^{a_j}\psi \in A$ iff $\psi \in A'$ (propagation of j-abstract formulas);
  if $B \neq \gamma^\perp$ (A is a matched return), then: $top_i \in B$ iff $top_i \in A'$ (A' gets the top_i status of its matching call), $\bigcirc^{a_i}\psi \in B$ iff $\psi \in A'$ (i-abstract formulas are propagated from matching call), and B and A' contain the same formulas of the form $\bigcirc^{-_i}\psi$ (matching call and return have the same i-call successor);
  if $B = \gamma^\perp$ (A is an unmatched return), then: $top_i \in A$ and $top_i \in A'$ (unmatched returns are top), $\bigcirc^{a_i}\psi \in A$ iff $\psi \in A'$ (i-abstract formulas are propagated from A), and A and A' do not contain formulas of the form $\bigcirc^{-_i}\psi$.

The fulfillment of formulas of the form $\psi_1\mathcal{U}\psi_2$ and, only on the top_i positions, of formulas of the form $\psi_1\mathcal{U}^{a_i}\psi_2$ is ensured with the addition of a Büchi condition.

In particular, for each formula of the form $\psi_1 \mathcal{U} \psi_2 \in cl_\varphi$, we define an acceptance set with all the atoms containing either ψ_2 or $\neg(\psi_1 \mathcal{U} \psi_2)$, and for each formula of the form $\psi_1 \mathcal{U}^{a_i} \psi_2$ we define an acceptance set with all the atoms containing top_i along with either ψ_2 or $\neg(\psi_1 \mathcal{U}^{a_i} \psi_2)$.

For each $i \in [n]$, a Büchi acceptance condition with all the atoms containing top_i is also needed if there are no formulas of the form $\psi_1 \mathcal{U}^{a_i} \psi_2$ in cl_φ. This ensures that each accepting run visits infinitely often top_i-atoms, and thus by the transition rules, each call that is declared matched (by pushing a $B \neq \partial$ onto a stack) is effectively matched in any accepting run.

Note that in this construction we actually use a generalized Büchi acceptance condition, that is a set of Büchi acceptance conditions that have to be all fulfilled in order to accept. Moving from a generalized Büchi acceptance condition with m acceptance sets to a standard Büchi condition by using a modulo $m+1$ counter is a well known technique and thus we omit further details on this.

The size of cl_φ is linear in the size of φ, thus the number of states, stack symbols and transitions of M_φ is $2^{O(|\varphi|)}$. The translation from generalized to standard Büchi conditions increases the size only by a $O(|\varphi|)$ factor (number of until formulas in cl_φ plus n). Therefore, we get:

Theorem 3. *Given a* MultiCaRet *formula φ over n-nested words, it is possible to construct a Büchi n-*Mpds *M_φ such that for each w: $w \models \varphi$ iff $\langle\langle \rho, \lambda_\varphi \rangle\rangle = w$ for some computation ρ of M_φ. Moreover, the size of M_φ is $2^{O(|\varphi|)}$.*

5 Büchi Mpds with Scope-Bounded Matching Relations

In this section, we show that emptiness of Büchi Mpds restricted to computations with scope-bounded matching relations is decidable in exponential time.

Scoped Runs. We restrict Mpds to runs where a symbol can be popped from a stack i only if it has been pushed within one of the last k execution contexts of this stack, where a *context* is a run such that all the pop and push transitions are over the same stack. We formally define this restriction on the multiply nested words and then export it to corresponding runs.

A multiply nested word $w = (\{\sigma_i\}_{i \in \mathbb{N}}, \{\langle \mu_i, C_i, R_i \rangle \mid i \in [n]\})$ is *k-scoped* if for each $i, j \in \mathbb{N}$ for which $\mu_h(i,j)$, $h \in [n]$, holds then there are at most $2d-3$ positions $x_1, \ldots, x_{d-1} \in \bigcup_{h' \neq h}(C_{h'} \cup R_{h'})$ and $y_1, \ldots, y_{d-2} \in C_h \cup R_h$ such that $i < x_1 < y_1 < \ldots < x_{d-2} < y_{d-2} < x_{d-1} < j$ and $d \leq k$. A run ρ is *k-scoped* if $\langle\langle \rho \rangle\rangle$ is k-scoped.

A context where the only active stack is the h-th is also called a h-context. For a finite h-context from $\mathcal{C}_0$ to $\mathcal{C}_r$, we write $(q, w) \rightsquigarrow_h (q', w')$ if $\mathcal{C}_0 = \langle q, \{w_i\}_{i \in [n]} \rangle$ with $w_h = w$ and $\mathcal{C}_r = \langle q', \{w'_i\}_{i \in [n]} \rangle$ with $w'_h = w'$.

Decision Procedure. We reduce the emptiness problem for Büchi Mpds to the same problem for standard Büchi automata.

Given M, we first define a *PDS* M_h, for $h \in [n]$, obtained by ignoring all the actions $push_i$ and pop_i, for $i \neq h$. M_h collects, in its states, pairs of states of

M, which are the beginning and the end of a context involving stack h, along with a bit storing the information whether a state from $\mathcal{F}$ has been entered in that context or not. The idea exploited here is similar to that in [12] where the concept of *thread-interface* is used to summarize the executions of a thread in consecutive rounds within a fixed-point algorithm to solve the scope-bounded reachability problem for MPDS.

Fix $M = (Q, q_0, \Gamma \cup \{\gamma^{\perp}\}, \Sigma, \lambda, \delta^{int}, \{(\delta_i^{push}, \delta_i^{pop})\}_{i \in [n]})$ be an n-MPDS, $h \in [n]$ and $\mathcal{F} \subseteq Q$.

Formally, the PDS M_h is $(Q', Q_0, \Gamma, \Sigma, \lambda, \delta'^{int}, (\delta'^{push}, \delta'^{pop}))$ where:

– the set of states is $Q' = \bigcup_{m \in [k]} (Q \times Q \times \{0,1\})^m$;
– the set of iniatial states is $Q_0 = \{(q, q, 0) | q \in Q \setminus \mathcal{F}\} \cup \{(q, q, 1) | q \in \mathcal{F}\}$;
– the transition functions are as follows (set X be either a state of Q', with length $m < k$, or the empty word)
  - $(X(q, p, f), \gamma, X(q, p', f')) \in \delta'^{push}$ if $(p, \gamma, p') \in \delta_h^{push}$, $f' = 1$ if $p' \in \mathcal{F}$, and $f' = f$, otherwise.
  - $(X(q, p, f), \gamma, X(q, p', f')) \in \delta'^{pop}$, if $(p, \gamma, p') \in \delta_h^{pop}$, $f' = 1$ if $p' \in \mathcal{F}$, and $f' = f$, otherwise.
  - $(X(q, p, f), X(q, p', f')) \in \delta'^{int}$, if $(p, p') \in \delta^{int}$, $f' = 1$ if $p' \in \mathcal{F}$, and $f' = f$, otherwise.
  - $(X, X(q, q, f)) \in \delta'^{int}$ (a *jump*), for every state $q \in Q$ and $f = 1$ if $q \in \mathcal{F}$, and $f = 0$, otherwise.

A triple $(q, p, f) \in Q \times Q \times \{0, 1\}$ is called a *summary*. The PDS M_h collects summaries in its states: it starts from a state (q, q, f) and modifies the second and the third component, following a run of M, then, by nondeterministic jumps, it adds a new summary, forming lists of at most k summaries. To obtain longer, possibly infinite, lists of summaries, we use a sequential composition which simply appends a list of summary after another list. Let $\mathcal{R}_h$ be the set of reachable states of M_h and let *Closure*$(\mathcal{R}_h)$ contain the finite and infinite lists of summaries obtained by sequential composition, starting from elements of $\mathcal{R}_h$. The the following lemma follows, using induction, from the definition of M_h.

Lemma 1. *Let $X = \{(q_i, p_i, f_i)\}_{i \in \mathbb{N}}$. If $X \in$ Closure$(\mathcal{R}_h)$ then there exist h-contexts ρ_i and words $w_i \in \Gamma^*$, for $i \in \mathbb{N}$, such that:*

1. *$w_1 = \epsilon$ and $(q_i, w_i) \rightsquigarrow_h (p_i, w_{i+1})$,*
2. *$f_i = 1$ if and only if a state from $\mathcal{F}$ occurs in ρ_i.*

To obtain also a reverse implication of Lemma 1 for k-scoped runs, we show that a list X of summaries can be associated to a k-scope bounded run ρ in such a way that the summaries in X are associated to the h-contexts, and X is obtained just by sequentially composing sequences, each having at most k summaries. The idea is similar to that used in [12], by taking into account the non terminating computations and the Büchi condition.

Lemma 2. *Let ρ be a k-scoped run of M. There exist h-contexts $\{\rho_i\}_{i \in \mathbb{N}}$ in ρ such that, called q_i and p_i the first and the last states of ρ_i, $\{(q_i, p_i, f_i)\}_{i \in \mathbb{N}} \in$ Closure$(\mathcal{R}_h)$, where $f_i = 1$ if and only if a state from $\mathcal{F}$ occurs in ρ_i.*

Now, we define a Büchi automaton $B_M = (Q_B, \{\sigma\}, \delta_B, Q_0^M, \mathcal{F}_\mathcal{B})$ which puts together summaries to simulate a run of M. For this, it saves in its states the current state of M, a reachable state X_h of M_h, for $h \in [n]$, and a bit for the acceptance condition. At each step, B_M consumes a summary. When the summaries of an X_h has been exhausted then it chooses another X'_h in the same set $\mathcal{R}_h$. Thus the states of B_M are $(p, X_1, \cdots, X_n, f)$, where $p \in Q$, $X_h \in \mathcal{R}_h$ and $f \in \{0, 1\}$ and the initial states are $(q, X_1, \cdots, X_n, f)$ such that $q \in Q_0$ and $f = 1$ if and only if $q \in \mathcal{F}$. The transition function δ_B contains $((p, X_1, \cdots, X_n, f), \sigma, (p', X'_1, \cdots, X'_n, f'))$ if there exists $h \in [n]$ such that $X'_i = X_i$, for $i \neq h$, $X_h = (p, p', f_h)Y$, $f' = f_h$, and $X'_h = Y$ if $Y \neq \epsilon$ and $X'_h \in \mathcal{R}_h$, otherwise. Finally, the acceptance condition $\mathcal{F}_\mathcal{B}$ is $\{(p, X_1, \cdots, X_n, f) \mid f = 1\}$.

It is easy to see that B_M accepts a word if and only if $(M, \mathcal{F})$ accepts a k-scoped computation. Moreover, the size of B_M is exponential in the number of the stacks and in the bound k. Thus, since the reachable states of a pushdown system can be efficiently computed (e.g., [9]), we can state the theorem:

Theorem 4. *The problem of deciding whether there exists a k-scope run of an n-*Mpds *M satisfying a Büchi condition $\mathcal{F}$ is decidable in $|M|^{O(nk)}$ time.*

6 Decidability Results for MultiCaRet

In this section, we show that the satisfiability and model-checking problems for MultiCaRet become decidable by restricting the models to meaningful subclasses of multiply nested words. Let us first show that these problems reduce to the emptiness problem for Büchi Mpds.

Automata-theoretic approach to MultiCaRet *model-checking.* For $i \in [2]$, let $M_i = (Q_i, Q_i^0, \Gamma_i \cup \{\gamma^\perp\}, \Sigma, \lambda_i, \delta_i^{int}, \{(\delta_i^{push_j}, \delta_i^{pop_j})\}_{j \in [n]}, F_i)$ be a n-Mpds. The synchronized cross product $M_1 \otimes M_2$, is the n-Mpds M such that: $M = (Q, Q_0, (\Gamma_1 \times \Gamma_2) \cup \{\gamma^\perp\}, \Sigma, \lambda, \delta^{int}, \{(\delta_j^{push}, \delta_j^{pop})\}_{j \in [n]}, F)$ where $Q = \{(q_1, q_2) \mid \lambda_1(q_1) = \lambda_2(q_2)\}$, $Q_0 = Q \cap (Q_1^0 \times Q_2^0)$, λ is such that $\lambda(q_1, q_2) = \lambda_1(q_1)$, the transition functions are such that: $((q_1, q_2), (q'_1, q'_2)) \in \delta^{int}$ iff $(q_i, q'_i) \in \delta_i^{int}$ for $i \in [2]$, and for $j \in [n]$: $((q_1, q_2), (\gamma_1, \gamma_2), (q'_1, q'_2)) \in \delta_j^{push}$ iff $(q_i, \gamma_i, q'_i) \in \delta_i^{push_j}$ for $i \in [2]$, and $((q_1, q_2), (\gamma_1, \gamma_2), (q'_1, q'_2)) \in \delta_j^{pop}$ iff $(q_i, \gamma_i, q'_i) \in \delta_i^{pop_j}$ for $i \in [2]$ (where we have identified $(\gamma^\perp, \gamma^\perp)$ with $\gamma^\perp$).

If one or both the Mpds are Büchi Mpds the above construction can be adapted in the usual way to suit the Büchi condition(s). A salient property of this construction is that the resulting Mpds defines a language of multiply nested words that is the intersection of the languages of the starting Mpds.

Lemma 3. *For (Büchi)* Mpds *M_i with $i \in [2]$, $M = M_1 \otimes M_2$ is a (Büchi)* Mpds *and $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ holds.*

In section 4, we have shown that given a formula φ over n-nested words, we can construct a Büchi Mpds M_φ that captures all the n-nested words that satisfy φ. Thus, a given model checking instance formed by a Mpds M and a formula φ, reduces to checking that $\mathcal{L}(M) \cap \mathcal{L}(M_{\neg\varphi})$ is empty. Therefore, by the above lemma and Theorem 3, we get:

Theorem 5. *For a* MPDS M *and a* MULTICARET *formula* φ, $M \models \varphi$ *iff* $\mathcal{L}(M) \cap \mathcal{L}(M_{\neg\varphi}) = \emptyset$.

6.1 Scope-Bounded Multiply Nested Words

Let us restrict to k-scoped multiply nested words. To capture the set of all k-scoped multiply nested words satisfying a MULTICARET formula, it suffices to place the same limitation on the runs of the Büchi MPDS M_φ from Section 4. Therefore, following the automata-theoretic approach described above, by Theorems 3, 4 and 5, we have:

Theorem 6. *The* MULTICARET *satisfiability and model-checking problems restricted to k-scoped multiply nested works are* EXPTIME*-complete.*

6.2 Multiply Nested Words with Ordered Matching Relations

We recall that in ordered MPDS a symbol can be popped out from a stack h provided that all stacks of lower indices (from 1 through $h-1$) are empty [8]. We define ordered multiply nested words by imposing the same restriction. A multiply nested word $(\{\sigma_i\}_{i\in\mathbb{N}}, \{\langle\mu_i, C_i, R_i\rangle \mid i \in [n]\})$ is *ordered* if for every $i, j \in \mathbb{N}$ for which $\mu_h(i,j)$ holds for some $h \in [n]$: if there is a $x < j$ such that $x \in C_{h'}$, $h' < h$, then there is a $y < j$ such that $\mu_{h'}(x,y)$ holds (all calls of lower-index relations preceding j are already matched at j).

Checking the emptiness of Büchi ordered MPDS is known to be 2ETIME-complete [3,4], and can be solved in time $|M|^{2^{O(n)}}$ where $|M|$ denotes the size of the input MPDS [3]. Therefore, by Theorems 3 and 5, we have:

Theorem 7. *The* MULTICARET *satisfiability and model-checking problems restricted to ordered multiply nested works are* 2ETIME*-complete.*

6.3 Multiply Nested Words of Bounded Tree-Width

The expressiveness of MULTICARET does not go beyond first-order logic interpreted over multiply nested words. We define FO_μ as the first-order logic over multiply nested words which has in its signature relations that capture the matching relations. Namely, the logic contains the usual binary predicate $<$ (the ordering relation over integers) along with a binary predicate μ_i and unary predicates C_i, R_i for $i \in [n]$ such that (μ_i, C_i, R_i) define a matching relation. Also, we use the unary predicates $P_\sigma(x)$ meaning that x is labeled with symbol σ, and fix a countable set of first-order variables $x, y, \ldots$. The logic FO_μ is defined as:

$$\varphi := P_\sigma(x) | x < y | \mu_i(x,y) | \neg\varphi | \varphi \vee \varphi | \exists x \varphi \quad \text{(where } i \in [n])$$

With similar constructions as those used in [1] to show that CARET formulas are FO definable, we can show the following theorem.

Theorem 8. *Given a* MULTICARET *formula φ it is possible to construct effectively a sentence ψ of FO_μ such that $|\psi| = O(|\varphi|)$ and $w \models \varphi$ if and only if w satisfies ψ.*

This result allows us to extend the decidability of MultiCaRet satisfiability and model-checking to all the MSO-definable classes of multiply nested words of bounded tree-width. In fact, for each class of MSO-definable graphs of bounded tree width, the satisfiability of MSO sentences is decidable [13]. Thus, by Theorem 8 we get:

Theorem 9. *Restricting the models to any MSO-definable class of multiply nested words, the satisfiability and model-checking problems of* MultiCaRet *formulas are decidable.*

Acknowledgments. We thank Gennaro Parlato for helpful discussions.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Log. Meth. Comp. Sci. 4(4) (2008)
2. Alur, R., Etessami, K., Madhusudan, P.: A Temporal Logic of Nested Calls and Returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
3. Atig, M.F.: Global model checking of ordered multi-pushdown systems. In: FSTTCS, pp. 216–227 (2010)
4. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of Multi-pushdown Automata Is 2ETIME-Complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
5. Atig, M.F., Bouajjani, A., Narayan Kumar, K., Saivasan, P.: Linear-time model-checking for multithreaded programs under scope-bounding. In: Mukund, M., Chakraborty, S. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 152–166. Springer, Heidelberg (2012)
6. Bollig, B., Cyriac, A., Gastin, P., Zeitoun, M.: Temporal Logics for Concurrent Recursive Programs: Satisfiability and Model Checking. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 132–144. Springer, Heidelberg (2011)
7. Bozzelli, L., La Torre, S., Peron, A.: Verification of well-formed communicating recursive state machines. Theor. Comput. Sci. 403(2-3), 382–405 (2008)
8. Breveglieri, L., Cherubini, A., Citrini, C., Crespi-Reghizzi, S.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. 7(3), 253–292 (1996)
9. Esparza, J., Schwoon, S.: A BDD-Based Model Checker for Recursive Programs. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 324–336. Springer, Heidelberg (2001)
10. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170. IEEE Computer Society (2007)
11. La Torre, S., Napoli, M.: Reachability of Multistack Pushdown Systems with Scope-Bounded Matching Relations. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 203–218. Springer, Heidelberg (2011)
12. La Torre, S., Parlato, G.: Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width (2012), `http://users.ecs.soton.ac.uk/gp4/papers/scoped2012.pdf`
13. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL, pp. 283–294 (2011)
14. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE Computer Society (1977)
15. Qadeer, S., Rehof, J.: Context-Bounded Model Checking of Concurrent Software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)

The Algorithmic Complexity of k-Domatic Partition of Graphs[*]

Hongyu Liang

Institute for Interdisciplinary Information Sciences,
Tsinghua University, Beijing, China
lianghy08@mails.tsinghua.edu.cn

Abstract. Let $G = (V, E)$ be a simple undirected graph, and k be a positive integer. A *k-dominating set* of G is a set of vertices $S \subseteq V$ satisfying that every vertex in $V \setminus S$ is adjacent to at least k vertices in S. A *k-domatic partition* of G is a partition of V into k-dominating sets. The *k-domatic number* of G is the maximum number of k-dominating sets contained in a k-domatic partition of G. In this paper we study the k-domatic number from both algorithmic complexity and graph theoretic points of view. We prove that it is $\mathcal{NP}$-complete to decide whether the k-domatic number of a bipartite graph is at least 3, and present a polynomial time algorithm that approximates the k-domatic number of a graph of order n within a factor of $(\frac{1}{k} + o(1)) \ln n$, generalizing the $(1 + o(1)) \ln n$ approximation for the 1-domatic number given in [5]. In addition, we determine the exact values of the k-domatic number of some particular classes of graphs.

1 Introduction

In this paper we consider only simple and undirected graphs, and we follow [3] for notations and terminologies in graph theory. Let $G = (V, E)$ be a simple and undirected graph. For a vertex $v \in V$, let $N_G(v)$ denote the set of neighbors of v, and $deg_G(v) = |N_G(v)|$ is the *degree* of v. When no ambiguity arises, we sometimes drop the subscript G. Let $\delta(G) = \min_{v \in V}\{deg(v)\}$ be the *minimum degree* of G. For an integer $k \geq 1$, a *k-coloring* of G is a mapping $c : V \to \{1, 2, \ldots, k\}$ such that $c(u) \neq c(v)$ whenever $\{u, v\} \in E$. We say G is *k-colorable* if G has a k-coloring.

Domination theory is a very important branch of graph theory which has found applications in numerous areas; see [12,13] for a comprehensive treatment and some detailed surveys on (earlier) results of domination in graphs. A set of vertices $S \subseteq V$ is called a *dominating set* of G if every vertex in $V \setminus S$ has at least one neighbor in S. The *domination number* of G is the minimum size of a dominating set of G. A *domatic partition* of G is a partition of V into

[*] This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, and the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

(disjoint) dominating sets of G. The *domatic number* of G, denoted by $d(G)$, is the maximum number of dominating sets in a domatic partition of G. The concept of domatic number was introduced by Cockayne and Hedetniemi [2], which has been proven very useful in various situations such as locating facilities in a network [9], clusterhead rotation in sensor networks [17], prolonging the lifetime and conserving energy of networks [14,18], and many others.

Let $k \geq 1$ be a fixed integer. A *k-dominating set* of G is a set of vertices $S \subseteq V$ with the property that every vertex in $V \setminus S$ has at least k neighbors in S. Clearly a 1-dominating set is just a dominating set. The notion of k-dominating set was proposed by Fink and Jacobson [6,7], and has since then been extensively studied for both its theoretical interest and its practical applications in fault-tolerant domination in networks; see, e.g., [1,4,8,19,20] and the references therein. It is known that deciding the size of the minimum k-dominating set of a graph is NP-hard [21]. (In the literature, some researchers use the name k-dominating set to refer to another variant of dominating set, namely the distance-k dominating set [11,16].)

A k-domatic partition of G is a partition of V into (disjoint) k-dominating sets of G. The *k-domatic number* of G, denoted by $d_k(G)$, is the maximum number of k-dominating sets in a k-domatic partition of G. Thus $d_1(G) = d(G)$. The concept of k-domatic number was first studied by Zelinka [22] under the name "k-ply domatic number," and was later rediscovered and studied under its current name by Kämmerling and Volkmann [15]. This concept is useful for modeling networks that need domatic partitions with higher degree of domination. As an example, imagine that we wish to locate resources in a network to facilitate the users (i.e., nodes). A user in the network can access resources only from itself and his neighboring nodes. A user is surely happy if there is one resource at his location, but if not, he would only be satisfied if he could access at least k copies of resources from its neighbors, keeping the possibility of multiple choices as a compensation of distance. Then the set of nodes with resources satisfying all the users is exactly a k-dominating set of the network. Now suppose we wish to distribute different types of resources (to enhance the quality of life of users) with the natural constraint that at most one kind of resource can be placed at each node. Then the maximum number of resource types that can be put in the network is precisely the k-domatic number of its underlying graph.

Despite being a natural generalization of the domatic number whose combinatorial and algorithmic aspects have both been well understood, the k-domatic number lacks an investigation from a complexity viewpoint, which motivates our study.

In this paper, we explore the k-domatic number mainly from the algorithmic complexity point of view, and obtain several results that fill the blank in this line of research. In Section 2 we prove that for every $k \geq 1$, it is $\mathcal{NP}$-complete to decide whether the k-domatic number of a given bipartite graph is at least 3. This generalizes the NP-completeness result for the 1-domatic number [10]. We then present in Section 3 a polynomial time algorithm that approximates the k-domatic number of a given graph of order n within factor $(\frac{1}{k} + o(1)) \ln n$, which generalizes the $(1 + o(1)) \ln n$ approximation for the domatic number given in [5]. Finally, as a minor contribution, we determine in Section 4 the exact values of the k-domatic number of some special classes of graphs.

2 Complexity of Computing the k-Domatic Number

In this section we show the hardness of computing the k-domatic number of a graph. Our main theorem is as follows.

Theorem 1. *For every fixed integer $k \geq 1$, it is $\mathcal{NP}$-complete to decide whether the k-domatic number of a given graph is at least 3.*

To establish Theorem 1 we introduce a new variant of the coloring problem, which may have its own interest in other scenarios. Let k be a fixed positive integer and $H = (V, E)$ be a $2k$-uniform hypergraph, i.e., a hypergraph in which each edge contains exactly $2k$ vertices. A mapping $c : V \to \{1, 2, 3\}$ is called a *balanced 3-coloring* of H if for every $e \in E$, there exist $1 \leq i < j \leq 3$ such that $|c^{-1}(i) \cap e| = |c^{-1}(j) \cap e| = k$; that is, every edge of H contains exactly k vertices of color i and k vertices of color j (and no vertices of the color other than i and j). Define the $2k$-Uniform Hypergraph Balanced 3-Coloring Problem as follows:

$2k$-Uniform Hypergraph Balanced 3-Coloring Problem ($2k$HB3C, for short)

Instance: A $2k$-uniform hypergraph H.

Question: Does H have a balanced 3-coloring?

Lemma 1. *For every fixed integer $k \geq 1$, $2k$HB3C is $\mathcal{NP}$-complete.*

Proof. Let k be a fixed positive integer. The $2k$HB3C problem is clearly in $\mathcal{NP}$, since we can verify in polynomial time whether a given mapping is a balanced 3-coloring of H by exhaustively checking all its edges. We now present a polynomial-time reduction from the Graph 3-Coloring problem (G3C for short), which is a classical $\mathcal{NP}$-complete problem [10], to $2k$HB3C. An instance of G3C consists of a graph G, and the goal is to decide whether G is 3-colorable. Let $G = (V, E)$ be a graph serving as the input to G3C. We will construct a $2k$-uniform hypergraph H from G. Informally speaking, the hypergraph H can be obtained as follows: For each edge $e \in E$, we associate it with a $2k$-uniform hypergraph H_e, where H_e has vertex set $X_e \cup Y_e$ with $|X_e| = |Y_e| = 3k$, and contains all possible hyperedges that consist of exactly k vertices from X_e and another k vertices from Y_e. The number of such hyperedges is $\binom{3k}{k} \cdot \binom{3k}{k} < 2^{6k}$, which is a constant since k is a fixed integer. Let H be the union of all such (disjoint) hypergraphs. Finally, for each $e \in E$, add to H a hyperedge which consists of both vertices in e, the first $k - 1$ vertices in X_e, and the first $k - 1$ vertices in Y_e. This finishes the construction of H. It is clear that H can be constructed in polynomial time.

We now give a rigorous definition of H. For every $e \in E$, let

- $X_e = \{x_{e,i} \mid 1 \le i \le 3k\}$ and $Y_e = \{y_{e,i} \mid 1 \le i \le 3k\}$;
- $E'_e = \{X \cup Y \mid X \subseteq X_e; Y \subseteq Y_e; |X| = |Y| = k\}$;
- $e' = e \cup \{x_{e,i}, y_{e,i} \mid 1 \le i \le k-1\}$.

Let $V' = V \cup \bigcup_{e \in E}(X_e \cup Y_e)$, and $E' = \{e' \mid e \in E\} \cup \bigcup_{e \in E} E'_e$. Finally let $H = (V', E')$. It is easy to verify that H is a $2k$-uniform hypergraph with $|V'| = |V| + 6k|E|$ and $|E'| = (1 + \binom{3k}{k} \cdot \binom{3k}{k})|E|$.

We will prove that G is 3-colorable if and only if H has a balanced 3-coloring.

First consider the "only if" direction. Assume that G is 3-colorable and $c : V \to \{1,2,3\}$ is a 3-coloring of G. Define a function $c' : V' \to \{1,2,3\}$ as follows. First let $c'(v) = c(v)$ for all $v \in V$. For each edge $e = \{u,v\} \in E$, suppose $c(u) = a$ and $c(v) = b$ where $a, b \in \{1,2,3\}$ (note that $a \ne b$). Then, let $c'(x) = a$ for all $x \in X_e$ and $c'(y) = b$ for all $y \in Y_e$. We verify that the mapping c' defined above is a balanced 3-coloring of H. This can be seen as follows:

- For each hyperedge $h = X \cup Y$ with $X \subseteq X_e$ and $Y \subseteq Y_e$ for some $e \in E$, by our definition, h contains exactly k vertices of the same color with that of one endpoint of e, and another k vertices of the same color with that of the other endpoint of e. Since the two endpoints of e have different colors, h satisfies the property of balanced 3-coloring.
- For each hyperedge $e' = e \cup \{x_{e,i}, y_{e,i} \mid 1 \le i \le k-1\}$ for some e, similar to the previous case, h consists of precisely k vertices of one color and the other k vertices of another color.

Therefore, c' is a balanced 3-coloring of H.

We next consider the "if" direction. Suppose that c' is a balanced 3-coloring of H. Let $e = \{u,v\}$ be an arbitrary edge in E. We claim that all the vertices in X_e have the same color i for some $i \in \{1,2,3\}$, all those in Y_e have the same color j for some $j \in \{1,2,3\}$, and $i \ne j$. This will imply that u and v have different colors under c', since otherwise the hyperedge $e' = e \cup \{x_{e,i}, y_{e,i} \mid 1 \le i \le k-1\}$ is not balanced. We now prove the above claim. As $|X_e| = 3k$, there exists $i \in \{1,2,3\}$ such that the number of vertices in X_e with color i is at least k; without loss of generality we assume that $c'(x_{e,1}) = c'(x_{e,2}) = \ldots = c'(x_{e,k}) = i$. Since the hyperedge $\{x_{e,1}, x_{e,2}, \ldots, x_{e,k}\} \cup Y$ exists for all $Y \subseteq Y_e$ with $|Y| = k$, we know that in every size-k subset of Y_e, all the vertices have the same color. Thus, all vertices in Y_e has the same color, say j, and obviously $j \ne i$. Analogously, all vertices in X_e has the same color i, proving the claim. According to our previous analysis, the claim implies that $c'(u) \ne c'(v)$ for all $\{u,v\} \in E$. Therefore, the mapping $c : V \to \{1,2,3\}$ defined by $c(v) = c'(v)$ for all $v \in V$ is a 3-coloring of G, and hence G is 3-colorable.

This finishes the reduction from G3C to $2k$HB3C, and thus concludes the proof of Lemma 1.

We now proceed to prove Theorem 1.

Proof (of Theorem 1). Let k be a fixed positive integer. We reduce $2k$HB3C to the problem of deciding whether $d_k(G) \ge 3$ for a given graph G. Note that the latter problem is clearly in $\mathcal{NP}$. Let $H = (V, E)$ be a $2k$-uniform hypergraph given as an input to the $2k$HB3C problem. We construct a graph $G = (V', E')$ as follows. Let $V' = X \cup Y \cup Z$, where $X = \{x_e \mid e \in E\}$, $Y = \{y_v \mid v \in V\}$, and $Z = \{z_i \mid 1 \leq i \leq 3k\}$. Let $E' = \{\{x_e, y_v\} \mid v \in e \in E\} \cup \{\{y_v, z_i\} \mid v \in V; 1 \leq i \leq 3k\} \cup \{\{z_i, z_j\} \mid 1 \leq i < j \leq 3k\}$. Thus, $G[X \cup Y]$ is the incidence graph of H, $G[Y \cup Z]$ contains a complete bipartite subgraph with partition (Y, Z), and $G[Z]$ is a clique. It is clear that the construction of G can be finished in polynomial time.

We shall show that H has a balanced 3-coloring if and only if $d_k(G) \geq 3$, which will complete the reduction and prove the $\mathcal{NP}$-completeness of the desired problem.

First consider the "only if" direction. Assume that H has a balanced 3-coloring $c : V \to \{1, 2, 3\}$. For each $e \in E$ let $C_e = \{i \mid \exists\ v \in e \text{ s.t. } c(v) = i\}$; clearly $|C_e| = 2$. We now design a partition (V_1', V_2', V_3') of V' as follows: For each $i \in \{1, 2, 3\}$, let $V_i' = \{x_e \mid i \notin C_e\} \cup \{y_v \mid c(v) = i\} \cup \{z_j \mid (i-1)k + 1 \leq j \leq ik\}$. It is easy to see that this is indeed a partition of V'. Furthermore, we will prove that for each $i \in \{1, 2, 3\}$, V_i' is a k-dominating set of G. Fix $i \in \{1, 2, 3\}$. Notice that $V' \setminus V_i' = \{x_e \mid i \in C_e\} \cup \{y_v \mid c(v) \neq i\} \cup \{z_j \mid j \in \{1, \ldots, 3k\} \setminus \{(i-1)k + 1, \ldots, ik\}\}$. By our construction of G, every vertex in $(Y \cup Z) \setminus V_i'$ is adjacent to k vertices in V_i', which are $z_{(i-1)k+1}, \ldots, z_{ik}$. For each $x_e \in X \setminus V_i'$, we have $i \in C_e$ by our definition of V_i'. Thus, there exists $u \in e$ for which $c(u) = i$. Because c is a balanced 3-coloring of H, there exist exactly k vertices in e that have value i under c, which indicates that those k vertices are all included in V_i'. Therefore, x_e is adjacent to at least k vertices in V_i'. This proves that V_i', for every $i \in \{1, 2, 3\}$, is a k-dominating set of G, and hence $d_k(G) \geq 3$, finishing the proof of the "only if" direction of the reduction.

We now turn to the "if" direction. Assume that $d_k(G) \geq 3$ and (V_1', V_2', V_3') is a k-domatic partition of G. Define a mapping $c : V \to \{1, 2, 3\}$ as follows: For every $v \in V$, let $c(v) = i$ where i is the unique integer satisfying that $y_v \in V_i'$. We show that c is a balanced 3-coloring of H. Consider an arbitrary edge $e \in E$, and assume without loss of generality that $x_e \in V_1'$ (and thus $x_e \notin V_2' \cup V_3'$). By the definition of k-dominating sets, for each $j \in \{2, 3\}$, $|N_G(x_e) \cap V_j'| \geq k$. As $|N_G(x_e)| = |\{y_v \mid v \in e\}| = 2k$ and $V_2' \cap V_3' = \emptyset$, we have $|N_G(x_e) \cap V_2'| = |N_G(x_e) \cap V_3'| = k$. Thus, $c(y_v) = 2$ for exactly k vertices $v \in e$, and $c(y_v) = 3$ for the other k ones, showing the validity of the coloring on edge e. Hence, c is indeed a balanced 3-coloring of H. This concludes the "if" direction of the reduction.

The proof of Theorem 1 is thus completed.

We remark that the $\mathcal{NP}$-completeness result holds even if the input graph is bipartite. To see this, we modify the construction of G in the proof as follows: Add $3k$ vertices $\{z_i' \mid 1 \leq i \leq 3k\}$ to H, add an edge between every possible pair (z_i, z_j'), and let $H[Z]$ be an empty graph (instead of being a complete graph as in the previous proof). Then it is easy to verify that G is a bipartite graph. The remaining part of the proof goes through analogously. The only modification is that when proving the "only if" direction of the reduction, we define the partition (V_1', V_2', V_3') as $V_i' = \{x_e \mid i \notin C_e\} \cup \{y_v \mid c(v) = i\} \cup \{z_j, z_j' \mid (i-1)k + 1 \leq j \leq ik\}$. Therefore we obtain:

Corollary 1. *Deciding whether the k-domatic number of a bipartite graph is at least 3 is $\mathcal{NP}$-complete for every fixed positive integer k.*

The following corollary is immediate.

Corollary 2. *For every fixed integer $k \geq 1$, computing the k-domatic number of a bipartite graph is $\mathcal{NP}$-hard.*

3 Approximation Algorithm for *k*-Domatic Number

Since computing the k-domatic number is $\mathcal{NP}$-hard, we are interested in designing approximation algorithms for it. In this section we present a logarithmic-factor approximation algorithm for computing the k-domatic number of a graph, generalizing the result of [5] for the 1-domatic number.

Theorem 2. *For every fixed integer $k \geq 1$, the k-domatic number of a given graph of order n can be approximated within a factor of $(\frac{1}{k} + o(1)) \ln n$ in polynomial time.*

Proof. Fix an integer $k \geq 1$. Let $G = (V, E)$ be a graph of order $n \geq N_0$, where N_0 is a sufficiently large but fixed integer (which may depend on k). (Note that the k-domatic number of a graph of order $n \leq N_0$ can be computed in constant time.) If $\delta(G) \leq \ln n + 3k \ln\ln n$, due to Theorem 2.9 in [15], we have $d_k(G) \leq \frac{\delta(G)}{k} + 1 \leq (\frac{1}{k} + o(1)) \ln n$. In this case, a trivial k-domatic partition that consists of only V itself is already a $(\frac{1}{k} + o(1)) \ln n$ approximate solution. Therefore, we assume in what follows that $\delta(G) > \ln n + 3k \ln\ln n$.

Let $t = \delta(G)/(\ln n + 3k \ln\ln n)$. For every vertex $v \in V$, assign a label $l(v) \in \{1, 2, \ldots, t\}$ to v uniformly at random; that is, $l(v) = i$ with probability $1/t$ for all $i \in \{1, 2, \ldots, t\}$. Let S_i, $1 \leq i \leq t$, be the set of vertices that receive label i. Evidently $\{S_1, S_2, \ldots, S_t\}$ is a partition of V. For $v \in V$ and $i \in \{1, 2, \ldots, t\}$, let $\mathcal{E}(v, i)$ denote the event that at most $k - 1$ neighbors of v have label i. If there is no $v \in V$ for which $\mathcal{E}(v, i)$ holds, then every vertex $v \in V$ has at least k neighbors in S_i, and hence S_i is a k-dominating set of G. For all $v \in V$ and $i \in \{1, 2, \ldots, t\}$, we have

$$\begin{aligned} Pr[\mathcal{E}(v,i)] &= \sum_{j=0}^{k-1} \binom{deg(v)}{j} \left(\frac{1}{t}\right)^j \left(1 - \frac{1}{t}\right)^{deg(v)-j} \\ &\leq \sum_{j=0}^{k-1} (deg(v))^j \left(\frac{1}{t}\right)^j \left(1 - \frac{1}{t}\right)^{deg(v)-j} \\ &= \left(1 - \frac{1}{t}\right)^{deg(v)} \cdot \sum_{j=0}^{k-1} \left(\frac{deg(v)}{t} \left(1 - \frac{1}{t}\right)^{-1}\right)^j , \end{aligned}$$

where the first inequality follows from the fact that $\binom{n_1}{n_2} \leq n_1^{n_2}$ for two positive integers $n_1 \geq n_2$.

As $\frac{deg(v)}{t}(1-\frac{1}{t})^{-1} = \frac{deg(v)}{t-1} \geq \frac{deg(v)}{\delta(G)} \geq 1$, we have

$$\begin{aligned}
Pr[\mathcal{E}(v,i)] &\leq \left(1-\frac{1}{t}\right)^{deg(v)} \cdot \sum_{j=0}^{k-1} \left(\frac{deg(v)}{t}\left(1-\frac{1}{t}\right)^{-1}\right)^{k-1} \\
&= \left(1-\frac{1}{t}\right)^{deg(v)} \cdot k \cdot \left(\frac{deg(v)}{t}\right)^{k-1}\left(1-\frac{1}{t}\right)^{-k+1} \\
&= k\left(1-\frac{1}{t}\right)^{deg(v)-k+1}\left(\frac{deg(v)}{t}\right)^{k-1} \\
&\leq k \cdot \exp\left(-\frac{deg(v)-k+1}{t} + (k-1)\ln\left(\frac{deg(v)}{t}\right)\right) \\
&\qquad \text{(where we use } 1+x \leq e^x \text{ for all } x \in \mathbb{R}, \text{and denote } \exp(m) := e^m).
\end{aligned}$$

Define a function f as $f(x) = -x + (k-1)\ln x$. Clearly f is non-increasing on $[X_0, \infty)$ for some sufficiently large but fixed X_0 (depending on k only). As $\frac{\deg(v)}{t} \geq \frac{\delta(G)}{t} = \Omega(\ln n)$, by choosing large enough $n \geq N_0$ we have $f(\frac{deg(v)}{t}) \leq f(\frac{\delta(G)}{t})$, and thus

$$\begin{aligned}
Pr[\mathcal{E}(v,i)] &\leq k \cdot \exp\left(-\frac{\delta(G)-k+1}{t} + (k-1)\ln\left(\frac{\delta(G)}{t}\right)\right) \\
&= k \cdot \exp\left(-\frac{\delta(G)-k+1}{\delta(G)/(\ln n + 3k\ln\ln n)} + (k-1)\ln\left(\frac{\delta(G)}{\delta(G)/(\ln n + 3k\ln\ln n)}\right)\right) \\
&= k \cdot \exp\left(-(\ln n + 3k\ln\ln n)(1-O(1/\ln n)) + (k-1)\ln(\ln n + 3k\ln\ln n)\right) \\
&\qquad \text{(where we use } \delta(G) > \ln n \text{ and } k = O(1)) \\
&\leq \exp\left(-\ln n - 2k\ln\ln n + o(\ln\ln n)\right) \\
&\leq \exp\left(-\ln n - k\ln\ln n\right) \\
&= n^{-1}(\ln n)^{-k}.
\end{aligned}$$

Call a pair (v,i) *bad* if the event $\mathcal{E}(v,i)$ happens. By linearity of expectation, the expected number of bad pairs is

$$\sum_{v\in V; 1\leq i\leq t} Pr[\mathcal{E}(v,i)] \leq nt \cdot n^{-1}(\ln n)^{-k} = t \cdot o(1).$$

Recall that $S_i = \{v \in V \mid l(v) = i\}$ for each $i \in \{1,2,\ldots,t\}$. Notice that S_i is a k-dominating set of G if and only if there is no $v \in V$ such that (v,i) is a bad pair. Clearly a bad pair (v,i) can "prevent" at most one such set, namely S_i, from being a k-dominating set of G. Hence, the expected number of k-dominating sets among $\{S_i \mid i \in \{1,2,\ldots,t\}\}$ is at least $t - t\cdot o(1) = (1-o(1))t$. By checking the t sets $S_1, S_2, \ldots, S_t$ one by one, we can find all the k-dominating sets among them. Add the vertices not covered by these sets to them arbitrarily. Then, we obtain a k-domatic partition of G of (expected) size $(1 - o(1))t$. This solution has an approximation factor of

$$\frac{d_k(G)}{(1-o(1))t} \leq \frac{\frac{\delta(G)}{k}+1}{(1-o(1))\delta(G)/(\ln n + 3k \ln\ln n)} \leq \left(\frac{1}{k}+o(1)\right)\ln n \; .$$

Finally we show that this algorithm can be efficiently derandomized by the method of conditional probabilities. Order the vertices in V arbitrarily, say $v_1, v_2, \ldots, v_n$. We assign labels to the vertices according to this order, from v_1 to v_n. Suppose we are dealing with v_i, and the labels of $v_1, \ldots, v_{i-1}$ have already been fixed to be $l_1, \ldots, l_{i-1}$, respectively. We try all the possible labels $1, 2, \ldots, t$ one by one, and assign v_i with the label l_i that *minimizes* the expected number of bad pairs conditioned on that $(\forall 1 \leq s \leq i)\ l(v_s) = l_s$. (Recall that $l(v)$ is the label of v; here we regard it as a random variable.) This expected number can be computed in polynomial time, because it is equal to

$$\sum_{v \in V; 1 \leq j \leq t} Pr[\mathcal{E}(v,j) \mid (\forall 1 \leq s \leq i)\ l(v_s) = l_s],$$

where, denoting by r_j the number of neighbors of v_i that has already been given label j, we have

$$Pr[\mathcal{E}(v,j)] = \begin{cases} 0, & \text{if } r_j \geq k; \\ 1, & \text{if } r_j < k \text{ and } \sum_{q=1}^{t} r_q = deg(v); \\ \sum_{j'=0}^{k-1-r_j} \binom{deg(v)-\sum_{q=1}^{t} r_q}{j'} \left(\frac{1}{t}\right)^{j'} \left(1-\frac{1}{t}\right)^{deg(v)-\sum_{q=1}^{t} r_q - j'} & \text{, otherwise.} \end{cases}$$

Since k is fixed, we can compute every $Pr[\mathcal{E}(v,j)]$ in polynomial time, and there are only $|V| \cdot t \leq n^2$ of them.

By our choice of labels, after all labels have been determined, the number of bad pairs does not exceed the expected number of bad pairs estimated before. The remaining arguments go through analogously as before, and we can obtain a solution of approximation factor $(\frac{1}{k} + o(1)) \ln n$. This completes the proof of Theorem 2.

4 *k*-Domatic Number of Special Graphs

In this section we determine the exact values of the k-domatic number of some special classes of graphs. By Theorem 2.9 in [15], $d_k(G) \leq \frac{\delta(G)}{k} + 1$. As $d_k(G)$ is a positive integer, we have $d_k(G) = 1$ whenever $k > \delta(G)$. Therefore, when considering $d_k(G)$ we only care those k for which $2 \leq k \leq \delta(G)$. (The case $k = 1$ corresponds to the domatic number, which has been extensively studied in the literature.)

For every integer $n \geq 2$, let F_n denote the fan graph with vertex set $V = \{v_0, v_1, v_2, \ldots, v_n\}$ and edge set $E = \{v_i v_{i+1} \mid 1 \leq i \leq n-1\} \cup \{v_0 v_i \mid 1 \leq i \leq n\}$. Obviously $\delta(F_n) = 2$.

Theorem 3. *Let $n \geq 2$ be an integer. Then,*

$$d_2(F_n) = \begin{cases} 1 & \textit{if } n \in \{2,4\}; \\ 2 & \textit{otherwise.} \end{cases}$$

Proof. We have $d_2(F_n) \leq \delta(F_n)/2 + 1 = 2$. When n is odd, it can be verified that $V_0 := \{v_i \mid 0 \leq i \leq n; i \text{ is even}\}$ and $V_1 := \{v_i \mid 0 \leq i \leq n; i \text{ is odd}\}$ are both 2-dominating sets of F_n, and clearly (V_0, V_1) is a partition of V. Thus $d_2(F_n) = 2$ when n is odd. If $n = 2$ or 4, it can be checked exhaustively that $d_2(F_n) = 1$. Now consider the case where n is even and $n \geq 6$. Let $Z_0 = \{v_i \mid 0 \leq i \leq n-4; i \text{ is even}\} \cup \{v_{n-1}\}$, and $Z_1 = V \setminus Z_0$. It is easy to see that for each $j \in \{0,1\}$, every vertex in Z_j is adjacent to at least two vertices in Z_{1-j}. Thus Z_0 and Z_1 are both 2-dominating sets of G, indicating that $d_2(F_n) \geq 2$. Hence $d_2(F_n) = 2$, and the proof of Theorem 3 is complete.

For every integer $n \geq 3$, let W_n denote the wheel graph with vertex set $V = \{v_0, v_1, v_2, \ldots, v_n\}$ and edge set $E = \{v_i v_{i+1} \mid 1 \leq i \leq n-1\} \cup \{v_n v_1\} \cup \{v_0 v_i \mid 1 \leq i \leq n\}$. Clearly $\delta(W_n) = 3$.

Theorem 4. *$d_2(W_n) = 2$ for every integer $n \geq 3$.*

Proof. Let $V_0 = \{v_i \mid 0 \leq i \leq n; i \text{ is even}\}$ and $V_1 = \{v_i \mid 0 \leq i \leq n; i \text{ is odd}\}$. It is easy to verify that V_0 and V_1 are both 2-dominating sets of W_n (regardless of the parity of n), and thus $d_2(W_n) \geq 2$. On the other hand, we have $d_2(W_n) \leq \lfloor \delta(W_n)/2 \rfloor + 1 = 2$. Hence $d_2(W_n) = 2$.

Theorem 5. *$d_3(W_n) = 1$ for every integer $n \geq 3$.*

Proof. First note that $d_3(W_n) \leq \lfloor \delta(W_n)/3 \rfloor + 1 = 2$. Assume that $d_3(W_n) = 2$ and (V_0, V_1) is a 3-domatic partition of G. Also assume without loss of generality that $v_0 \in V_0$. If $v_i \notin V_1$ for some $1 \leq i \leq n$, then all the three neighbors of v_i must belong to V_1, implying that $v_0 \in V_1$ which is a contradiction. Thus $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $V_0 = \{v_0\}$. But then V_0 is not a 3-dominating set of G. Therefore $d_3(W_n) = 2$ cannot hold, and thus $d_3(W_n) = 1$.

Acknowledgement. The author is grateful to the anonymous referees for their helpful suggestions on improving the presentation of this paper.

References

1. Caro, Y., Roditty, Y.: A note on the k-domination number of a graph. Internat. J. Math. Math. Sci. 13, 205–206 (1990)
2. Cockayne, E.J., Hedetniemi, S.T.: Towards a theory of domination in graphs. Networks 7, 247–261 (1977)
3. Diestel, R.: Graph Theory, 4th edn. Springer (2010)
4. Favaron, O., Hansberg, A., Volkmann, L.: On k-domination and minimum degree in graphs. J. Graph Theory 57, 33–40 (2008)

5. Feige, U., Halldórsson, M.M., Kortsarz, G., Srinivasan, A.: Approximating the domatic number. SIAM J. Comput. 32(1), 172–195 (2002)
6. Fink, J.F., Jacobson, M.S.: n-domination in graphs. In: Graph Theory with Applications to Algorithms and Computer Science, pp. 282–300 (1985)
7. Fink, J.F., Jacobson, M.S.: On n-domination, n-dependence and forbidden subgraphs. In: Graph Theory with Applications to Algorithms and Computer Science, pp. 301–311 (1985)
8. Fujisawa, J., Hansberg, A., Kubo, T., Saito, A., Sugita, M., Volkmann, L.: Independence and 2-domination in bipartite graphs. Australas. J. Combin. 40, 265–268 (2008)
9. Fujita, S., Yamashita, M., Kameda, T.: A study on r-configurations – a resource assignment problem on graphs. SIAM J. Discrete Math. 13, 227–254 (2000)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
11. Hansberg, A., Meierling, D., Volkmann, L.: Distance domination and distance irredundance in graphs. Electron. J. Comb. 14 (2007)
12. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: Advanced Topics. Marcel Dekker (1998)
13. Haynes, T.W., Hedetniemi, S.T.: ST, and P.J. Slater. Fundamentals of Domination in Graphs. Marcel Dekker (1998)
14. Islam, K., Akl, S.G., Meijer, H.: Maximizing the lifetime of wireless sensor networks through domatic partition. In: Proceedings of the 34th IEEE Conference on Local Computer Networks, LCN (2009)
15. Kämmerling, K., Volkmann, L.: The k-domatic number of a graph. Czech. Math. J. 59(2), 539–550 (2009)
16. Meir, A., Moon, J.W.: Relations between packing and covering number of a tree. Pacific J. Math. 61, 225–233 (1975)
17. Misra, R., Mandal, C.: Efficient clusterhead rotation via domatic partition in self-organizing sensor networks. Wireless Communications & Mobile Computing 9(8), 1040–1058 (2009)
18. Pemmaraju, S.V., Pirwani, I.A.: Energy conservation via domatic partitions. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc (2006)
19. Pepper, R.: Implications of some observations about the k-domination number. Congr. Numer. 206, 65–71 (2010)
20. Rautenbach, D., Volkmann, L.: New bounds on the k-domination number and the k-tuple domination number. Appl. Math. Lett. 20, 98–102 (2007)
21. Telle, J.A.: Complexity of domination-type problems in graphs. Nord. J. Comput. 1(1), 157–171 (1994)
22. Zelinka, B.: On k-ply domatic numbers of graphs. Math. Slovaca 34(3), 313–318 (1984)

Unique Parallel Decomposition in Branching and Weak Bisimulation Semantics

Bas Luttik

Eindhoven University of Technology

Abstract. We consider the property of unique parallel decomposition modulo branching and weak bisimilarity. First, we show that totally normed behaviours always have parallel decompositions, but that these are not necessarily unique. Then, we establish that finite behaviours have unique parallel decompositions. We derive the latter result from a general theorem about unique decompositions in partial commutative monoids.

1 Introduction

A recurring question in process theory is to what extent the behaviours definable in a certain process calculus admit a unique decomposition into indecomposable parallel components. Milner and Moller [18] were the first to address the question. They proved a unique parallel decomposition theorem for a simple process calculus, which allows the specification of finite behaviour up to strong bisimilarity and includes parallel composition in the form of pure interleaving without interaction between the components. They also presented counterexamples showing that unique parallel decomposition may fail in process calculi in which it is possible to specify infinite behaviour, or in which certain coarser notions of behavioural equivalence are used.

Moller, in [19], proved several more unique parallel decomposition results, replacing interleaving parallel composition by CCS parallel composition, and then also considering weak bisimilarity. These results were established with subsequent refinements of an ingenious proof technique attributed to Milner. Christensen, in [5], further refined the proof technique to make it work for the *normed* behaviours recursively definable modulo strong bisimilarity, and for *all* behaviours recursively definable modulo distributed bisimilarity.

With each successive refinement of Milner's proof technique, the technical details became more complicated, but the general idea of the proof remained the same. In [15] we made an attempt to isolate the deep insights from the technical details, by identifying a sufficient condition on partial commutative monoids that facilitates an abstract version of Milner's proof technique. To concisely present the sufficient condition, we have put forward the notion of *decomposition order*; it is established in [15], by means of an abstract version of Milner's technique, that if a partial commutative monoid can be endowed with a decomposition order, then it has unique decomposition.

Application of the general result of [15] in commutative monoids of behaviour is often straightforward: a well-founded order naturally induced on behaviour by (a terminating fragment of) the transition relation typically satisfies the properties of a decomposition order. All the aforementioned unique parallel decomposition results can be directly obtained in this way, except Moller's result that finite behaviours modulo weak bisimilarity have unique decomposition. It turns out that a decomposition order cannot straightforwardly be obtained from the transition relation if certain transitions are deemed unobservable by the behavioural equivalence under consideration.

In this paper, we address the question of how to establish unique parallel decomposition in settings with a notion of unobservable behaviour. Our main contribution will be an adaptation of the general result in [15] to make it suitable for establishing unique parallel decomposition also in settings with a notion of unobservable behaviour. To illustrate the result, we shall apply it to establish unique parallel decomposition for finite behaviour modulo branching or weak bisimilarity. We shall also show, by means of a counterexample, that unique parallel decomposition fails for infinite behaviours modulo branching and weak bisimilarity, even if only a very limited form of infinite behaviour is considered (totally normed behaviour definable in a process calculus with prefix iteration).

A positive answer to the unique parallel decomposition question seems to be primarily of theoretical interest, as a tool for proving other theoretical properties of interest about process calculi. For instance, Moller's proofs in [20,21] that PA and CCS cannot be finitely axiomatised without auxiliary operations, Hirshfeld and Jerrum's proof in [12] that bisimilarity is decidable for normed PA, and the completeness proofs for the equational axiomatisations of PA and CCS with auxiliary operations in [8] and [1], all rely on unique parallel decomposition. There is an intimate relationship between unique parallel decomposition and of cancellation with respect to parallel composition; the properties are in most circumstances equivalent. In [4], cancellation with respect parallel composition was first proved and exploited to prove the completeness of an axiomatisation of distributed bisimilarity. Unique parallel decomposition could be of practical interest too, e.g., to devise methods for finding the maximally parallel implementation of a behaviour [6], or for improving verification methods [11].

This article is organised as follows. In Section 2 we introduce the process calculus that we shall use to illustrate our theory of unique decomposition. There, we also present counterexamples to the effect that infinite behaviours in general may not have a decomposition, and totally normed behaviours may have more than one decomposition. In Section 3 we recap the theory of decomposition put forward in [15] and discuss why it is not readily applicable to establish unique parallel decomposition for finite behaviours modulo branching and weak bisimilarity. In Section 4 we adapt the theory of [15] to make it suitable for proving unique parallel decomposition results in process calculi with a notion of unobservability. We end the paper in Section 5 with a short conclusion.

This article is an extended abstract of [14], which includes additional examples and detailed explanations, and more elaborate proofs.

Table 1. The operational semantics

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \frac{}{\alpha^*P \xrightarrow{\alpha} \alpha^*P} \qquad \frac{P \xrightarrow{\alpha} P'}{\alpha^*P \xrightarrow{\alpha} P'}$$

2 Processes Up to Branching and Weak Bisimilarity

We define a simple language of process expressions together with an operational semantics, and notions of branching and weak bisimilarity. We shall then investigate to what extent process expressions modulo branching or weak bisimilarity admit parallel decompositions. We shall present examples of process expressions without a decomposition, and of totally normed process expressions with two distinct decompositions.

Syntax. We fix a set $\mathcal{A}$ of *actions*, and declare a special action τ that we assume is not in $\mathcal{A}$. We denote by $\mathcal{A}_\tau$ the set $\mathcal{A} \cup \{\tau\}$, and we let a range over $\mathcal{A}$ and α over $\mathcal{A}_\tau$. The set $\mathcal{P}$ of *process expressions* is generated by the following grammar:

$$P ::= \mathbf{0} \mid \alpha.P \mid P + P \mid P \parallel P \mid \alpha^*P \qquad (\alpha \in \mathcal{A}_\tau).$$

The language above is BCCS (the core of Milner's CCS [16]) extended with a construction $\_ \parallel \_$ to express interleaving parallelism and the prefix iteration construction $\alpha^*\_$ to specify a restricted form of infinite behaviour. We include only a very basic notion of parallel composition in our calculus, but note that this is just to simplify the presentation. Our unique decomposition theory extends straightforwardly to more intricate notions of parallel composition, e.g., modelling some form of communication between components. To be able to omit some parentheses when writing process expressions, we adopt the conventions that $\alpha.$ and α^* bind stronger, and that $+$ binds weaker than all the other operations.

Operational semantics and branching and weak bisimilarity. We define on $\mathcal{P}$ binary relations $\xrightarrow{\alpha}$ $(\alpha \in \mathcal{A}_\tau)$ by means of the operational rules in Table 1. We shall henceforth write $P \twoheadrightarrow P'$ if there exist $P_0, \dots, P_n$ $(n \geq 0)$ such that $P = P_0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_n = P'$. Furthermore, we shall write $P \xrightarrow{(\alpha)} P'$ if $P \xrightarrow{\alpha} P'$ or $\alpha = \tau$ and $P = P'$.

Definition 1 (Branching bisimilarity [10]). *A symmetric binary relation $\mathcal{R}$ on $\mathcal{P}$ is a* branching bisimulation *if for all $P, Q \in \mathcal{P}$ such that $P \mathrel{\mathcal{R}} Q$ and for all $\alpha \in \mathcal{A}_\tau$ it holds that*

if $P \xrightarrow{\alpha} P'$ for some $P' \in \mathcal{P}$, then there exist $Q'', Q' \in \mathcal{P}$ such that $Q \twoheadrightarrow Q'' \xrightarrow{(\alpha)} Q'$ and $P \mathrel{\mathcal{R}} Q''$ and $P' \mathrel{\mathcal{R}} Q'$.

We write $P \leftrightarroweq_b Q$ if there exists a branching bisimulation $\mathcal{R}$ such that $P \mathcal{R} Q$.

The relation $\leftrightarroweq_\mathrm{b}$ is an equivalence relation on $\mathcal{P}$ (this is not as trivial as one might expect; for a proof see [3]). It is also compatible with the construction of parallel composition in our syntax, which means that, for all $P_1, P_2, Q_1, Q_2 \in \mathcal{P}$:

$$P_1 \leftrightarroweq_\mathrm{b} Q_1 \text{ and } P_2 \leftrightarroweq_\mathrm{b} Q_2 \text{ implies } P_1 \parallel P_2 \leftrightarroweq_\mathrm{b} Q_1 \parallel Q_2 \ . \tag{1}$$

(The relation $\leftrightarroweq_\mathrm{b}$ is also compatible with $\alpha.$, but not with $+$ and α^*. In this paper, we shall only rely on compatibility with $\parallel$.)

Definition 2 (Weak bisimilarity [17]). *A symmetric binary relation $\mathcal{R}$ on $\mathcal{P}$ is a* weak bisimulation *if for all $P, Q \in \mathcal{P}$ such that $P \mathcal{R} Q$ and for all $\alpha \in \mathcal{A}_\tau$ it holds that*

if $P \xrightarrow{\alpha} P'$ for some $P' \in \mathcal{P}$, then there exist $Q', Q'', Q''' \in \mathcal{P}$ such that $Q \twoheadrightarrow Q'' \xrightarrow{(\alpha)} Q''' \twoheadrightarrow Q'$ and $P' \mathcal{R} Q'$.

We write $P \leftrightarroweq_w Q$ if there exists a weak bisimulation $\mathcal{R}$ such that $P \mathcal{R} Q$.

Like $\leftrightarroweq_\mathrm{b}$, the relation $\leftrightarroweq_\mathrm{w}$ is an equivalence relation on $\mathcal{P}$, and compatible with parallel composition. Note that $\leftrightarroweq_\mathrm{b} \subseteq \leftrightarroweq_\mathrm{w}$; we shall often implicitly use this property below.

A process expression is *indecomposable* if it is not behaviourally equivalent to $\mathbf{0}$ or a non-trivial parallel composition (a parallel composition is trivial if one of its components is behaviourally equivalent to $\mathbf{0}$). We say that a process theory has *unique parallel decomposition* if every process expression is behaviourally equivalent to a unique (generalised) parallel composition of indecomposable process expressions. Uniqueness means that the indecomposables of any two decompositions of a process expression are pairwise behaviourally equivalent up to a permutation.

Milner and Moller in [18] already observed that there exist infinite behaviours without a decomposition modulo strong bisimilarity; their example $a^*\mathbf{0}$ also does not have a decomposition modulo branching and weak bisimilarity. To exclude such examples of infinite behaviours with decompositions, we need to confine our attention to process expressions with terminating behaviour. (A formalisation of aforementioned notions pertaining to unique decomposition is postponed until the next section.)

For $a \in \mathcal{A}$ and process expressions P and Q we write $P \overset{a}{\twoheadrightarrow} Q$ whenever there exist process expressions P' and Q' such that $P \twoheadrightarrow P' \xrightarrow{a} Q' \twoheadrightarrow Q$. We say that P is *silent* and write $P{\downarrow}$ if there do not exist $a \in \mathcal{A}$ and Q such that $P \overset{a}{\twoheadrightarrow} Q$.

Definition 3. *A process expression P is* totally normed *if there exist a natural number $k \in \mathbf{N}$, process expressions $P_0, \ldots, P_k \in \mathcal{P}$ and actions $a_1, \ldots, a_k \in \mathcal{A}$ such that $P = P_0 \overset{a_1}{\twoheadrightarrow} \cdots \overset{a_k}{\twoheadrightarrow} P_k$ and $P_k{\downarrow}$. The* weak norm *$wn(P)$ of a totally normed process expression P is defined by*

$$wn(P) = \min\{k : \exists P_0, \ldots, P_k \in \mathcal{P}.\ \exists a_1, \ldots, a_k \in \mathcal{A}.\ P = P_0 \overset{a_1}{\twoheadrightarrow} \cdots \overset{a_k}{\twoheadrightarrow} P_k{\downarrow}\} \ .$$

It is immediate from their definitions that both branching and weak bisimilarity preserve weak norm: if two process expressions are branchingly or weakly bisimilar, then they have equal weak norms. It is also easy to establish that a parallel composition is weakly normed if, and only if, both parallel components are weakly normed. In fact, weak norm is additive with respect to parallel composition: the weak norm of a parallel composition is the sum of the weak norms of its parallel components. Note that a process expression with weak norm 0 is behaviourally equivalent to $\mathbf{0}$.

With a straightforward induction on weak norm it can be established that totally normed process expressions have a decomposition. But sometimes even more than one, as is illustrated in the following example.

Example 4. *Consider the process expressions* $P = a^*\tau.b.\mathbf{0}$ *and* $Q = b.\mathbf{0}$. *It is clear that* P *and* Q *are* not *branching bisimilar. Both* P *and* Q *have weak norm* 1*, and from this it immediately follows that they are both indecomposable. Note that, according to the operational semantics,* $P \parallel P$ *gives rise to the following three transitions:*

1. $P \parallel P \xrightarrow{a} P \parallel P$;
2. $P \parallel P \xrightarrow{\tau} P \parallel Q$; *and*
3. $P \parallel P \xrightarrow{\tau} Q \parallel P$.

Further note that $P \parallel Q \xrightarrow{a} P \parallel Q$ *and* $Q \parallel P \xrightarrow{a} Q \parallel P$. *(The complete transition graph associated with* $P \parallel P$ *by the operational semantics is shown in Figure 1.) Using these facts it is straightforward to verify that the symmetric closure of the binary relation*

$$\mathcal{R} = \{(P \parallel P, P \parallel Q), (P \parallel P, Q \parallel P)\} \\ \cup \{(P \parallel Q, Q \parallel P), (P \parallel \mathbf{0}, \mathbf{0} \parallel P), (Q \parallel \mathbf{0}, \mathbf{0} \parallel Q)\}$$

is a branching bisimulation, and hence $P \parallel P \leftrightarroweq_b P \parallel Q$. *It follows that* $P \parallel P$ *and* $P \parallel Q$ *are distinct decompositions of the same process up to branching bisimilarity.*

Incidentally, the processes in the above counterexample also refute claims in [9] to the effect that processes definable with a totally normed BPP specification have a unique decomposition modulo branching bisimilarity and weak bisimilarity.

Apparently, more severe restrictions are needed.

Definition 5. *Let* $k \in \mathbf{N}$*; a process expression* P *is* weakly bounded *by* k *if for all* $\ell \in \mathbf{N}$ *the existence of* $P_1, \ldots, P_\ell \in \mathcal{P}$ *and* $a_1, \ldots, a_\ell \in \mathcal{A}$ *such that* $P \xrightarrow{a_1}\!\!\!\!\rightarrow \cdots \xrightarrow{a_\ell}\!\!\!\!\rightarrow P_\ell$ *implies that* $\ell \leq k$. *We say that* P *is* weakly bounded *if* P *is bounded by* k *for some* $k \in \mathbf{N}$.

Lemma 6. *Let* P *and* Q *be process expressions such that* $P \leftrightarroweq_w Q$. *Then* P *is weakly bounded if, and only if,* Q *is weakly bounded.*

In the remainder of this paper we shall establish that weakly bounded process expressions have a unique parallel decomposition both modulo branching and weak bisimilarity. We shall derive these results from a more general result about unique decomposition in commutative monoids.

$$\begin{tikzcd}
{} \arrow[d] & & \\
P \parallel P \arrow[loop left, "a"] \arrow[r, "\tau"] \arrow[d, "\tau"] & P \parallel Q \arrow[loop above, "a"] \arrow[r, "b"] \arrow[d, "\tau"] & P \parallel \mathbf{0} \arrow[loop above, "a"] \arrow[d, "\tau"] \\
Q \parallel P \arrow[loop left, "a"] \arrow[r, "\tau"] \arrow[d, "b"] & Q \parallel Q \arrow[r, "b"] \arrow[d, "b"] & Q \parallel \mathbf{0} \arrow[d, "b"] \\
\mathbf{0} \parallel P \arrow[loop left, "a"] \arrow[r, "\tau"] & \mathbf{0} \parallel Q \arrow[r, "b"] & \mathbf{0} \parallel \mathbf{0}
\end{tikzcd}$$

Fig. 1. Transition graph associated with $P \parallel P$

3 Partial Commutative Monoids and Decomposition

In this section we recall the abstract algebraic notion of partial commutative monoid, and formulate the property of unique decomposition. We shall see that the process theories discussed in the previous section give rise to commutative monoids of processes with parallel composition as binary operation. The notion of unique decomposition associated with these commutative monoids coincides with the notion of unique parallel decomposition as discussed.

Then, we shall recall the notion of decomposition order on partial commutative monoids proposed in [15]. We shall investigate whether the notion of decomposition order can be employed to prove unique parallel decomposition of weakly bounded process expressions modulo branching and weak bisimilarity.

Definition 7. *A* (partial) commutative monoid *is a set M with a distinguished element e and a (partial) binary operation on M (for clarity in this definition denoted by $\cdot$) such that for all $x, y, z \in M$:*

$$\begin{aligned}
&x \cdot (y \cdot z) \simeq (x \cdot y) \cdot z && \textit{(associativity)};\\
&x \cdot y \simeq y \cdot x && \textit{(commutativity)};\\
&x \cdot e \simeq e \cdot x \simeq x && \textit{(identity)}.
\end{aligned}$$

The symbol $\cdot$ will be omitted if this is unlikely to cause confusion. Also, we shall sometimes use other symbols ($\parallel$, $+$, ...) to denote the binary operation of a partial commutative monoid.

Remark 8. *We adopt the convention that an expression designating an element of a partial commutative monoid M is defined only if all its subexpressions are defined. Furthermore, if t_1 and t_2 are expressions and $\mathcal{R}$ is a binary relation on M (e.g., equality or a partial order), then $t_1 \mathcal{R} t_2$ holds only if both t_1 and t_2 are* *defined and their values are related in $\mathcal{R}$. For a more succinct formulation we used in Definition 7 the symbol $\simeq$ introduced by Kleene [13]: if t_1 and t_2 are expressions designating elements of M, then $t_1 \simeq t_2$ means that either t_1 and t_2 are both defined and have the same value, or t_1 and t_2 are both undefined.*

We mention a key example of a commutative monoid that will serve to illustrate the theory of decomposition that we present in this paper.

Example 9. *Let X be any set. A* (finite) multiset *over X is a mapping $m : X \to \mathbf{N}$ such that $m(x) > 0$ for at most finitely many $x \in X$; the number $m(x)$ is called the* multiplicity *of x in m. The set of all multisets over X is denoted by $\mathcal{M}(X)$. If m and n are multisets, then their sum $m \uplus n$ is obtained by coordinatewise addition of multiplicities, i.e., $(m \uplus n)(x) = m(x) + n(x)$ for all $x \in X$. The* empty multiset *$\Box$ is the multiset that satisfies $\Box(x) = 0$ for all $x \in X$. With these definitions, $\mathcal{M}(X)$ is a commutative monoid. If $x_1, \ldots, x_k$ is a sequence of elements of X, then $\wr x_1, \ldots, x_k \int$ denotes the multiset m such that $m(x)$ is the number of occurrences of x in $x_1, \ldots, x_k$.*

Process expressions modulo branching or weak bisimilarity also give rise to commutative monoids. Recall that $\leftrightarrows_{\mathrm{b}}$ and $\leftrightarrows_{\mathrm{w}}$ are equivalence relations on the set of process expressions. We denote the equivalence class of a process expression P modulo $\leftrightarrows_{\mathrm{b}}$ or $\leftrightarrows_{\mathrm{w}}$, respectively, by $[P]_b$ and $[P]_w$. Then, we define

$$\mathbf{B} = \mathcal{P}/\leftrightarrows_{\mathrm{b}} = \{[P]_b : P \in \mathcal{P}\} \quad \text{and} \quad \mathbf{W} = \mathcal{P}/\leftrightarrows_{\mathrm{w}} = \{[P]_w : P \in \mathcal{P}\} \ .$$

In this paper, the similarities between the commutative monoids $\mathbf{B}$ and $\mathbf{W}$ will be more important than the differences. It will often be necessary to define notions for both commutative monoids, in a very similar way. For succinctness of presentation, we allow ourselves a slight *abus de language* and most of the time deliberately omit the subscripts b and w from our notation for equivalence classes. Thus, we will be able to efficiently define notions and prove facts simultaneously for $\mathbf{B}$ and $\mathbf{W}$.

For example, since both $\leftrightarrows_{\mathrm{b}}$ and $\leftrightarrows_{\mathrm{w}}$ are compatible with $\parallel$, we can define a binary operation $\parallel$ simultaneously on $\mathbf{B}$ and $\mathbf{W}$ simply by $[P] \parallel [Q] = [P \parallel Q]$, by which we then mean to define a binary operation $\parallel$ on $\mathbf{B}$ and a binary relation $\parallel$ on $\mathbf{W}$, respectively, by $[P]_b \parallel [Q]_b = [P \parallel Q]_b$ and $[P]_w \parallel [Q]_w = [P \parallel Q]_w$. Henceforth, we leave it to the reader to specialise notions, and also statements about these notions and their proofs, to $\mathbf{B}$ and $\mathbf{W}$ (or one of its submonoids to be introduced below).

We agree to write just $\mathbf{0}$ for $[\mathbf{0}]$. It is straightforward to establish that the binary operation $\parallel$ is commutative and associative (both on $\mathbf{B}$ and $\mathbf{W}$), and that $\mathbf{0}$ is the identity element for $\parallel$.

Proposition 10. $\mathbf{B}$ *and* $\mathbf{W}$ *are commutative monoids under* $\parallel$.

Note that, by Lemma 6, whenever an equivalence class $[P]$ contains a weakly bounded process expression, it consists entirely of weakly bounded process expressions. We define subsets $\mathbf{B}_{\mathit{fin}} \subseteq \mathbf{B}_{\mathit{tn}} \subseteq \mathbf{B}$ and $\mathbf{W}_{\mathit{fin}} \subseteq \mathbf{W}_{\mathit{tn}} \subseteq \mathbf{W}$ by

$$\begin{aligned}
\mathbf{B}_{\mathit{fin}} &= \{[P]_b : P \in \mathcal{P} \ \&\ P \text{ is weakly bounded}\} \ ; \\
\mathbf{B}_{\mathit{tn}} &= \{[P]_b : P \in \mathcal{P} \ \&\ P \text{ is totally normed}\} \ ; \\
\mathbf{W}_{\mathit{fin}} &= \{[P]_w : P \in \mathcal{P} \ \&\ P \text{ is weakly bounded}\} \ ; \text{ and} \\
\mathbf{W}_{\mathit{tn}} &= \{[P]_w : P \in \mathcal{P} \ \&\ P \text{ is totally normed}\} \ .
\end{aligned}$$

Corollary 11. *The sets $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{B}_{\mathit{tn}}$ are commutative submonoids of $\mathbf{B}$, and the sets $\mathbf{W}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{tn}}$ are commutative submonoids of $\mathbf{W}$.*

Notation 12. *Let $x_1, \dots, x_k$ be a (possibly empty) sequence of elements of a monoid M; we define its* generalised product *$x_1 \cdots x_k$ inductively as follows: (1) if $n = 0$, then $x_1 \cdots x_k \simeq e$, and (2) if $n > 0$, then $x_1 \cdots x_k \simeq (x_1 \cdots x_{k-1})x_k$. Occasionally, we shall write $\prod_{i=1}^{k} x_i$ instead of $x_1 \cdots x_k$. Furthermore, we write x^n for the k-fold composition of x, i.e., $x^k \simeq \prod_{i=1}^{k} x_i$ with $x_i = x$ for all $1 \leq i \leq k$.*

An indecomposable element of a commutative monoid is an element that cannot be written as a product of two elements that are both not the identity element of the monoid.

Definition 13. *An element p of a commutative monoid M is called* indecomposable *if $p \neq e$ and $p = xy$ implies $x = e$ or $y = e$.*

Example 14.
1. *The indecomposable elements of $\mathcal{M}(X)$ are the* singleton *multisets, i.e., the multisets m for which it holds that $\sum_{x \in X} m(x) = 1$.*
2. *The indecomposable elements of $\mathbf{B}_{\mathit{fin}}$, $\mathbf{B}_{\mathit{tn}}$, $\mathbf{B}$, $\mathbf{W}_{\mathit{fin}}$, $\mathbf{W}_{\mathit{tn}}$, and $\mathbf{W}$ are the equivalence classes of process expressions that are not behaviourally equivalent to $\mathbf{0}$ or a non-trivial parallel composition.*

We define a decomposition in a partial commutative monoid to be a finite multiset of indecomposable elements. Note that this gives the right notion of equivalence on decompositions, for two finite multisets $\wr x_1, \dots, x_k \int$ and $\wr y_1, \dots, y_\ell \int$ are the equal iff the sequence $y_1, \dots, y_\ell$ can be obtained from the sequence $x_1, \dots, x_k$ by a permutation of its elements.

Definition 15. *Let M be a partial commutative monoid. A* decomposition *in M is a finite multiset $\wr p_1, \dots, p_k \int$ of indecomposable elements of M such that $p_1 \cdots p_k$ is defined. The element $p_1 \cdots p_k$ in M will be called the* composition *associated with the decomposition $\wr p_1, \dots, p_k \int$, and, conversely, we say that $\wr p_1, \dots, p_k \int$ is a decomposition of the element $p_1 \cdots p_k$ of M. Decompositions $d = \wr p_1, \dots, p_k \int$ and $d' = \wr p'_1, \dots, p'_\ell \int$ are* equivalent *in M (notation: $d \equiv d'$) if they have the same compositions, i.e., if $p_1 \cdots p_k = p'_1 \cdots p'_\ell$. A decomposition d in M is* unique *if $d \equiv d'$ implies $d = d'$ for all decompositions d' in M. We say that an element x of M has a unique decomposition if it has a decomposition and this decomposition is unique. If every element of M has a unique decomposition, then we say that M has* unique decomposition.

Example 16. *Every finite multiset m over X has a unique decomposition in $\mathcal{M}(X)$, which contains for every $x \in X$ precisely $m(x)$ copies of the singleton multiset $\wr x \int$.*

The general notion of unique decomposition for commutative monoids, when instantiated to one of the commutative monoids of processes considered in this paper, indeed coincides with the notion of unique parallel decomposition as discussed in the preceding section. We have already seen that the commutative monoids $\mathbf{B}_{tn}$, $\mathbf{B}$, $\mathbf{W}_{tn}$ and $\mathbf{W}$ do not have unique decomposition. Our goal in the remainder of this paper is to establish that the commutative monoids $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ do have unique decomposition.

Preferably, we would like to have a general sufficient condition on partial commutative monoids for unique decomposition that is easily seen to hold for $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$, and hopefully also for other commutative monoids of processes. We shall now first recall the sufficient criterion put forward in [15], which was specifically designed for commutative monoids of processes. Then, we shall explain that it cannot directly be applied to conclude that $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ have unique decomposition. In the next section, we shall subsequently modify the condition, so that it becomes applicable to the commutative monoids at hand.

Definition 17. *Let M be a partial commutative monoid; a partial order $\preccurlyeq$ on M is a* decomposition order *if*

(i) it is well-founded*, i.e., every nonempty subset of M has a $\preccurlyeq$-minimal element;*
(ii) the identity element e of M is the least element *of M with respect to $\preccurlyeq$, i.e., $e \preccurlyeq x$ for all x in M;*
(iii) it is strictly compatible*, i.e., for all $x, y, z \in M$*

$$\text{if } x \prec y \text{ and } yz \text{ is defined, then } xz \prec yz;$$

(iv) it is precompositional*, i.e., for all $x, y, z \in M$*

$$x \preccurlyeq yz \text{ implies } x = y'z' \text{ for some } y' \preccurlyeq y \text{ and } z' \preccurlyeq z; \text{ and}$$

(v) it is Archimedean*, i.e., for all $x, y \in M$*

$$x^n \preccurlyeq y \text{ for all } n \in \mathbf{N} \text{ implies that } x = e.$$

In [15] it was proved that the existence of a decomposition order on a partial commutative monoid is a necessary and sufficient condition for unique decomposition. The advantage of establishing unique decomposition via a decomposition order is that it circumvents first establishing cancellation, which in some cases is hard without knowing that the partial commutative monoid has unique decomposition. We refer to [15] for a more in-depth discussion.

In commutative monoids of processes, an obvious candidate decomposition order is the order induced on the commutative monoid by the transition relation. We define a binary relation $\longrightarrow$ on $\mathbf{B}$ and $\mathbf{W}$ by

$$[P] \longrightarrow [P'] \text{ if there exist } Q \in [P],\ Q' \in [P'] \text{ and } \alpha \in \mathcal{A}_\tau \text{ such that } Q \xrightarrow{\alpha} Q' \ .$$

We shall denote the inverse of the reflexive-transitive closure of $\longrightarrow$ (both on $\mathbf{B}$ and $\mathbf{W}$) by $\preccurlyeq$, i.e., $\preccurlyeq = (\longrightarrow^*)^{-1}$.

Lemma 18. *If P and Q are process expressions such that $[Q] \preccurlyeq [P]$, then there exists $Q' \in [Q]$ such that $P \longrightarrow^* Q'$.*

The following lemma implies that every set of process expressions has minimal elements with respect to the reflexive-transitive closure of the transition relation. Caution: it holds true of our process calculus only thanks to the very limited facility of defining infinite behaviour, by means of simple loops.

Lemma 19. *If $P_0, \ldots, P_i, \ldots$ $(i \in \mathbf{N})$ is an infinite sequence of process expressions, and $\alpha_0, \ldots, \alpha_i, \ldots$ $(i \in \mathbf{N})$ is an infinite sequence of elements in $\mathcal{A}_\tau$ such that $P_i \xrightarrow{\alpha_i} P_{i+1}$ for all $i \in \mathbf{N}$, then there exists $j \in \mathbf{N}$ such that $P_k = P_\ell$ for all $k, \ell \geq j$.*

Proposition 20. *$\preccurlyeq$ is a well-founded precompositional partial order on each of the commutative monoids $\mathbf{B}$, $\mathbf{B}_{tn}$, $\mathbf{B}_{fin}$, $\mathbf{W}$, $\mathbf{W}_{tn}$, and $\mathbf{W}_{fin}$.*

Note that if the iteration prefix in our process calculus is replaced by any of the familiar more general forms of iteration or recursion, then $\preccurlyeq$ as defined above will not be anti-symmetric, nor well-founded. Nevertheless, it is sometimes possible to define an anti-symmetric and well-founded partial order on processes based on the transition relation in a setting with a more general form of infinite behaviour, at least for totally normed processes. (See, e.g., [15] for an example of an anti-symmetric and well-founded order on normed processes definable in ACP with recursion, which is based on the restriction of the transition relation.)

The ordering $\preccurlyeq$ defined on $\mathbf{B}_{tn}$, $\mathbf{B}$, $\mathbf{W}_{tn}$ and $\mathbf{W}$ is not a decomposition order: on $\mathbf{B}$ and $\mathbf{W}$ it does not satisfy conditions (ii), (iii) and (v) of Definition 17, and on $\mathbf{B}_{tn}$ and $\mathbf{W}_{tn}$ it does not satisfy condition (iii) of Definition 17. (The latter is illustrated in Example 4.)

Proposition 21. *$\preccurlyeq$ on $\mathbf{B}_{tn}$, $\mathbf{B}_{fin}$, $\mathbf{W}_{tn}$ and $\mathbf{W}_{fin}$ is Archimedean and $\mathbf{0}$ is its least element.*

We should now still ask ourselves the question whether $\preccurlyeq$ on $\mathbf{B}_{fin}$ and $\mathbf{W}_{fin}$ is strictly compatible. An important step towards proving the property for, e.g., $\mathbf{B}_{fin}$ would be to establish, for all weakly bounded process expressions P, Q and R, the following implication: $P \xrightarrow{\tau} Q$ & $P \parallel R \leftrightarroweq_{\mathrm{b}} Q \parallel R \Longrightarrow P \leftrightarroweq_{\mathrm{b}} Q$. Example 4 illustrates that this implication does not hold for all totally normed processes, suggesting that the implication is perhaps hard to establish from first principles. In fact, all our attempts in this direction so far have failed. Note, however, that establishing the implication would be straightforward if we could use that $\parallel$ is cancellative (i.e., $P \parallel R \leftrightarroweq_{\mathrm{b}} Q \parallel R$ implies $P \leftrightarroweq_{\mathrm{b}} Q$), and this, in turn, would be easy if we could use that $\mathbf{B}_{fin}$ has unique decomposition.

The difficulty of establishing strict compatibility is really with strictness; it is straightforward to establish the following non-strict variant. Let M be a partial commutative monoid; a partial order $\preccurlyeq$ on M is *compatible* if for all $x, y, z \in M$:

if $x \preccurlyeq y$ and yz is defined, then $xz \preccurlyeq yz$.

Proposition 22. *$\preccurlyeq$ on $\mathbf{B}_{tn}$, $\mathbf{B}_{\mathit{fin}}$, $\mathbf{W}_{tn}$, and $\mathbf{W}_{\mathit{fin}}$ is* compatible*.*

A partial order on a partial commutative monoid that has all the properties of a decomposition order except that it is compatible but not strictly compatible, we shall henceforth call a *weak* decomposition order.

Definition 23. *Let M be a partial commutative monoid; a partial order $\preccurlyeq$ on M is a* weak decomposition order *if it is well-founded, has the identity element $e \in M$ as least element, is compatible, precompositional and Archimedean.*

The following corollary summarises Propositions 20, 21 and 22.

Corollary 24. *$\preccurlyeq$ on $\mathbf{B}_{tn}$, $\mathbf{B}_{\mathit{fin}}$, $\mathbf{W}_{tn}$, and $\mathbf{W}_{\mathit{fin}}$ is a weak decomposition order.*

In [15] it is proved that the existence of a decomposition order is a sufficient condition for a partial commutative monoid to have unique decomposition. Note that since $\preccurlyeq$ is a weak decomposition order on $\mathbf{B}_{tn}$ and $\mathbf{W}_{tn}$, and according to Example 4 these commutative monoids do not have unique decomposition, the existence of a *weak* decomposition order is *not* a sufficient condition for having unique decomposition; it should be supplemented with additional requirements to get a sufficient condition.

Strictness of compatibility —which is the only difference between the notion of decomposition order of [15] and the notion of weak decomposition order put forward here— is used both in the proof of *existence* of decompositions and in the proof that decompositions are *unique*. We shall now first establish the existence of decompositions in $\mathbf{B}_{tn}$, $\mathbf{B}_{\mathit{fin}}$, $\mathbf{W}_{tn}$, and $\mathbf{W}_{\mathit{fin}}$ separately. In the next section, we shall discuss uniqueness of decompositions in $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$. We shall propose a general subsidiary property that will allow us to establish uniqueness of decompositions in commutative monoids with a weak decomposition order, and establish that it holds in $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$.

Proposition 25. *In the commutative monoids $\mathbf{B}_{tn}$, $\mathbf{B}_{\mathit{fin}}$, $\mathbf{W}_{tn}$, and $\mathbf{W}_{\mathit{fin}}$ every element has a decomposition.*

4 Uniqueness

The failure of $\preccurlyeq$ on $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ to be strictly compatible prevents us from getting our unique decomposition results as an immediate consequence of the result in [15]. Nevertheless, most of the ideas in the proof of uniqueness of decompositions in [15] can be adapted and reused in the context of commutative monoids endowed with a weak decomposition order, albeit with the technical details more involved. There is one special case in the unique decomposition proof that cannot be settled for commutative monoids with a weak decomposition order in general; this special case can be settled with an additional requirement on $\preccurlyeq$ that is satisfied both in $\mathbf{B}_{\mathit{fin}}$ and in $\mathbf{W}_{\mathit{fin}}$.

For the remainder of this paper, let M be a partial commutative monoid in which every element has a decomposition, and let $\preccurlyeq$ be a weak decomposition order on M.

The decomposition extension of $\preccurlyeq$. The uniqueness proof in [15] considers a minimal counterexample against unique decomposition, i.e., an element of the commutative monoid with at least two distinct decompositions, say d_1 and d_2, that is $\preccurlyeq$-minimal in the set of all such elements. Then, an important technique in the proof is to select a particular indecomposable in one of the two decompositions and replace it by predecessors with respect to the decomposition order. From minimality together with strict compatibility it is then concluded that the resulting decomposition is unique, which plays a crucial role in subsequent arguments towards a contradiction. To avoid the use of strict compatibility, we need a more sophisticated notion of minimality for the considered counterexample. The idea is to not just pick a $\preccurlyeq$-minimal element among the elements with two or more decompositions; we also choose the presupposed pair of distinct decompositions (d_1, d_2) in such a way that it is minimal with respect to a well-founded ordering induced by $\preccurlyeq$ on pairs of decompositions.

Let X be a set. If m and n are multisets over X, then we write $m - n$ for the multiset difference of m and n. We define the *decomposition extension* $\lhd$ of $\prec$ by $d \lhd d'$ if, and only if, there exist, for some $k \geq 1$, a sequence of indecomposables $p_1, \ldots, p_k \in M$, a sequence $x_1, \ldots, x_k \in M$, and a sequence of decompositions $d_1, \ldots, d_k$ such that

(i) $x_i \prec p_i$ $(1 \leq i \leq k)$;
(ii) each d_i is a decomposition of x_i $(1 \leq i \leq k)$; and
(iii) $d = (d' - \wr p_1, \ldots, p_k \int) \uplus (d_1 \uplus \cdots \uplus d_k)$.

We write $d \unlhd d'$ if $d = d'$ or $d \lhd d'$. Note that if $d \unlhd d'$, x is the composition of d, and y is the composition of d', then, by compatibility, $x \preccurlyeq y$.

Lemma 26. *The partial order $\unlhd$ on decompositions is well-founded.*

In our uniqueness proof, we shall use the well-foundedness of both $\preccurlyeq$ and the Cartesian order $\unlhd_\times$ induced on pairs of decompositions by $\unlhd$. For two pairs of decompositions (d_1, d_2) and (d_1', d_2'), we write $(d_1, d_2) \unlhd_\times (d_1', d_2')$ if $d_1 \unlhd d_1'$ and $d_2 \unlhd d_2'$. A pair of decompositions (d_1, d_2) is said to be a *counterexample* against unique decomposition if d_1 and d_2 are distinct but equivalent, i.e., if $d_1 \equiv d_2$, but not $d_1 = d_2$. A counterexample (d_1, d_2) against unique decomposition is *minimal* if it is both minimal with respect to $\preccurlyeq$ and minimal with respect to $\unlhd_\times$. If unique decomposition would fail then there would exist a minimal counterexample. For the subset of processes with two or more decompositions is nonempty, and therefore, by well-foundedness of $\preccurlyeq$, it has a $\preccurlyeq$-minimal element, say x. Then, by well-foundedness of $\unlhd_\times$ on pairs of decompositions, the nonempty set of pairs of distinct decompositions with x as their composition has a minimal element, say (d_1, d_2).

The general idea of the proof is that we derive a contradiction from the assumption that there exists a minimal counterexample (d_1, d_2) against unique decomposition. The decompositions d_1 and d_2 should be distinct, so the set of indecomposables that occur more often in one of the decompositions than in the other is nonempty. This set is clearly also finite, so it has $\preccurlyeq$-maximal elements. We declare p to be such a $\preccurlyeq$-maximal element, and assume, without loss of generality, that p occurs more often in d_1 than in d_2. Then we have that

(A) $d_1(p) > d_2(p)$; and
(B) $d_1(q) = d_2(q)$ for all indecomposables q such that $p \prec q$.

We shall distinguish two cases, based on how the difference between d_1 and d_2 manifests itself, and derive a contradiction in both cases:

1. $d_1(p) > d_2(p) + 1$ or $d_1(q) \neq 0$ for some indecomposable q distinct from p; we refer to this case by saying that d_1 and d_2 are *too far apart.*
2. $d_1(p) = d_2(p) + 1$ and $d_1(q) = 0$ for all q distinct from p; we refer to this case by saying that d_1 and d_2 are *too close together.*

Case 1: d_1 and d_2 are too far apart. We argue that d_1 has a predecessor d' in which p occurs more often than in any predecessor of d_2, while, on the other hand, the choice of a minimal counterexample implies that every predecessor of d_1 is also a predecessor of d_2. The arguments leading to a contradiction in this case are analogous to the arguments in the proof in [15]; the only important difference is the use of the ordering $\trianglelefteq$ instead of $\preccurlyeq$.

Case 2: d_1 and d_2 are too close together. In [15] it is proved, via a sophisticated argument, that the composition of d_2' is a $\preccurlyeq$-predecessor of p. Hence, by strict compatibility, the composition of d_2 is an $\preccurlyeq$-predecessor of d_1, which is in contradiction with the assumption that the decompositions d_1 and d_2 are equivalent.

That $\preccurlyeq$ is not strictly compatible, but just compatible, leaves the possibility that d_1 and d_2 are equivalent even if the composition of d_2' is a predecessor of p. For $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ this possibility can be ruled out by noting that the composition of d_2' can be reached from p by τ-transitions, and proving that every transition of p can be simulated by a transition of the composition of d_2'. The following notion formalises this reason in the abstract setting of commutative monoids with a weak decomposition order.

Definition 27. *Let M be a partial commutative monoid, and let $\preccurlyeq$ be a weak decomposition order on M. We say that $\preccurlyeq$ satisfies* power cancellation *if for all $x, y \in M$, for every indecomposable $p \in M$ such that $p \not\preccurlyeq x, y$, and for all $k \in \mathbf{N}$ it holds that $p^k x = p^k y$, then $x = y$.*

Suppose that $\preccurlyeq$ on M has power cancellation, let $k = d_2(p)$ and let x be the composition of d_2'. Then from $d_1 \equiv d_2$ it follows that $p^k p = p^k x$. Clearly, $p \not\preccurlyeq p$ and, since d_2' consists of indecomposables q such that $p \not\preccurlyeq q$, it follows that also $p \not\preccurlyeq x$. Hence, since $\preccurlyeq$ has power cancellation, $p = x$, so $d_2' = \wr p \int$. It follows that $d_1 = d_2$, which contradicts that (d_1, d_2) is a counterexample against unique decomposition.

Theorem 28. *Let M be a commutative monoid with a weak decomposition order that satisfies power cancellation. If every element of M has a decomposition, then M has unique decomposition.*

In the previous section we have already established that in the commutative monoids $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ every element has a decomposition and that $\preccurlyeq$ is a weak decomposition order on $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$. To be able to conclude from Theorem 28 that $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ have unique decomposition, it remains to establish that $\preccurlyeq$ on these commutative monoids satisfies power cancellation.

Proposition 29. *$\preccurlyeq$ on $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ satisfies power cancellation.*

By Corollary 24 and Propositions 25 and 29, the commutative monoids $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ are endowed with a weak decomposition order $\preccurlyeq$ satisfying power cancellation, and, moreover, all elements of $\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ have at least one decomposition. Hence, by Theorem 28, we obtain the following corollary.

Corollary 30. *$\mathbf{B}_{\mathit{fin}}$ and $\mathbf{W}_{\mathit{fin}}$ have unique decomposition.*

5 Concluding Remarks

We have presented a general sufficient condition on partial commutative monoids that implies the property of unique decomposition, and is applicable to commutative monoids of behaviour incorporating a notion of unobservability. We have illustrated the application of our condition in the context of a very simple process calculus with an operation for pure interleaving as parallel composition. The applicability is, however, not restricted to settings with this particular type of parallel composition. In fact, it is to be expected that our condition, similarly as in [15], can also be used to prove unique decomposition results in settings with more complicated notions of parallel composition operator allowing, e.g., synchronisation between components. We leave for future investigations to what extent our theory of unique decomposition can be applied to variants of π-calculus; the report [7], in which unique parallel decomposition is established for a fragment of Applied π-calculus, will serve as a starting point.

In [2], Balabonski and Haucourt address the problem of unique parallel decomposition in the context of a concurrent programming language with a geometric semantics. It is less clear whether our general theory of unique decomposition is applicable there too; at least, the geometric semantics does not as naturally induce a candidate decomposition order on processes as in a process calculus with a transition system semantics. It would be interesting to compare the approaches.

References

1. Aceto, L., Fokkink, W.J., Ingólfsdóttir, A., Luttik, B.: A finite equational base for CCS with left merge and communication merge. ACM Trans. Comput. Log. 10(1) (2009)
2. Balabonski, T., Haucourt, E.: A Geometric Approach to the Problem of Unique Decomposition of Processes. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 132–146. Springer, Heidelberg (2010)
3. Basten, T.: Branching bisimilarity is an equivalence indeed? Information Processing Letters 58(3), 141–147 (1996)

4. Castellani, I., Hennessy, M.: Distributed bisimulations. J. ACM 36(4), 887–911 (1989)
5. Christensen, S.: Decidability and Decomposition in Process Algebras. PhD thesis, University of Edinburgh (1993)
6. Corradini, F., Gorrieri, R., Marchignoli, D.: Towards parallelization of concurrent systems. RAIRO Inform. Théor. Appl. 32(4-6), 99–125 (1998)
7. Dreier, J., Ene, C., Lafourcade, P., Lakhnech, Y.: On unique decomposition of processes in the applied π-calculus. Technical Report TR-2012-3, Verimag Research Report (2011)
8. Fokkink, W.J., Luttik, S.P.: An ω-Complete Equational Specification of Interleaving. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 729–743. Springer, Heidelberg (2000)
9. Fröschle, S., Lasota, S.: Normed processes, unique decomposition, and complexity of bisimulation equivalences. ENTCS 239, 17–42 (2009)
10. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM 43(3), 555–600 (1996)
11. Groote, J.F., Moller, F.: Verification of Parallel Systems Via Decomposition. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 62–76. Springer, Heidelberg (1992)
12. Hirshfeld, Y., Jerrum, M.: Bisimulation Equivalence is Decidable for Normed Process Algebra. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 412–421. Springer, Heidelberg (1999)
13. Kleene, S.C.: Introduction to Metamathematics. D. Van Nostrand Co., Inc., New York (1952)
14. Luttik, B.: Unique parallel decomposition in branching and weak bisimulation semantics. CoRR, abs/1205.2117 (2012)
15. Luttik, B., van Oostrom, V.: Decomposition orders—another generalisation of the fundamental theorem of arithmetic. Theor. Comput. Sci. 335(2-3), 147–186 (2005)
16. Milner, R.: Communication and Concurrency. Prentice-Hall International (1989)
17. Milner, R.: Operational and algebraic semantics of concurrent processes. In: Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics (B), pp. 1201–1242. The MIT Press (1990)
18. Milner, R., Moller, F.: Unique decomposition of processes. Theoret. Comput. Sci. 107, 357–363 (1993)
19. Moller, F.: Axioms for Concurrency. PhD thesis, University of Edinburgh (1989)
20. Moller, F.: The Importance of the Left Merge Operator in Process Algebras. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 752–764. Springer, Heidelberg (1990)
21. Moller, F.: The nonexistence of finite axiomatisations for CCS congruences. In: Proceedings of LICS 1990, pp. 142–153. IEEE Computer Society (1990)

Modal Interface Automata

Gerald Lüttgen[1] and Walter Vogler[2]

[1] Software Technologies Group, University of Bamberg, 96045 Bamberg, Germany
gerald.luettgen@swt-bamberg.de

[2] Institute for Computer Science, University of Augsburg, 86135 Augsburg, Germany
vogler@informatik.uni-augsburg.de

Abstract. De Alfaro and Henzinger's Interface Automata (IA) and Nyman et al.'s recent combination IOMTS of IA and Larsen's Modal Transition Systems (MTS) are established frameworks for specifying interfaces of system components. However, neither IA nor IOMTS consider conjunction that is needed in practice when a component satisfies multiple interfaces, while Larsen's MTS-conjunction is not closed. In addition, IOMTS-parallel composition exhibits a compositionality defect.

This paper defines conjunction on IA and MTS and proves the operators to be 'correct', i.e., the greatest lower bounds wrt. IA- and resp. MTS-refinement. As its main contribution, a novel interface theory called Modal Interface Automata (MIA) is introduced: MIA is a rich subset of IOMTS, is equipped with compositional parallel and conjunction operators, and allows a simpler embedding of IA than Nyman's. Thus, it fixes the shortcomings of related work, without restricting designers to deterministic interfaces as Raclet et al.'s modal interface theory does.

1 Introduction

Interfaces play an important role when designing complex software and hardware systems. Early interface theories that deal with types of data and operations only, have recently been extended to also capture protocol aspects of component interaction. One prominent example of such a rich interface theory is de Alfaro and Henzinger's *Interface Automata* (IA) [4, 5], which is based on labelled transition systems (LTS) but distinguishes a component's input and output actions. The theory comes with an asymmetric parallel composition operator, where a component may wait on inputs but never on outputs. Thus, a component output must be consumed immediately, or an error occurs. In case no potential system environment may restrict the system components' behaviour so that all errors are avoided, the components are deemed to be incompatible.

Semantically, IA employs a refinement notion based on an alternating simulation, such that a component satisfies an interface if (a) it implements all input behaviour prescribed by the interface and (b) the interface permits all output behaviour executed by the implementing component. Notably, this means that a component that consumes all inputs but never produces any output satisfies any interface. To be able to mandate output-transitions, Larsen, Nyman and Wasowski have built their interface theory on Modal Transition Systems

(MTS) [7] rather than LTS, which enables one to distinguish between may- and must-transitions and thus to express compulsory outputs. The resulting *IOMTS* interface theory [8], into which IA can be embedded, is equipped with an IA-style parallel composition and an MTS-style modal refinement. Unfortunately, IOMTS-modal refinement has a compositionality defect wrt. parallel composition, i.e., it is not a precongruence for parallel composition; a related result in [8] has already been shown incorrect by Raclet et al. in [12].

The present paper starts from the observation that the above interface theories are missing one important operator, namely conjunction on interfaces. Conjunction is needed in practice since components are often designed to satisfy multiple interfaces simultaneously, each of which specifies a particular aspect of component interaction. We thus start off by recalling the IA-setting and defining a conjunction operator $\wedge$ for IA; we prove that $\wedge$ is indeed conjunction, i.e., the greatest lower bound wrt. alternating simulation (cf. Sec. 2). Similarly, we do so for a slight extension of MTS (a subset of *Disjunctive MTS* [10], cf. Sec. 3), which paves us the way for our main contribution outlined below. Although Larsen has studied conjunction for MTS, his operator does – in contrast to ours – not preserve the MTS-property of syntactic consistency, i.e., a conjunction almost always has some required transitions (must-transitions) that are not allowed (missing may-transitions). An additional difficulty when compared to the IA-setting is that two MTS-interfaces may not have a common implementation; indeed, inconsistencies may arise when composing MTSs conjunctively. We handle inconsistencies by adapting ideas from our prior work on conjunction in a CSP-style process algebra [11] that uses, however, a very different parallel operator and refinement preorder. Note also that our setting employs event-based communication via handshake and thus differs significantly from the one of shared-memory communication studied by Abadi and Lamport in their paper on conjoining specifications [1].

Our paper's main contribution is a novel interface theory, called *Modal Interface Automata* (MIA), which is essentially a rich subset of IOMTS that still allows one to express output-must-transitions. It is equipped with an MTS-style conjunction $\wedge$ and an IOMTS-style parallel composition operator, as well as with a slight adaptation of IOMTS-refinement. We show that (i) MIA-refinement is a precongruence for both operators; (ii) $\wedge$ is indeed conjunction for this preorder; and (iii) IA can be embedded into MIA in a much cleaner, homomorphic fashion than into IOMTS [8] (cf. Sec. 4). Thereby, we remedy the shortcomings of related work while, unlike the language-based modal interface theory of [12], still permitting nondeterminism in interface specifications.

2 Conjunction for Interface Automata

Interface Automata (IA) were introduced by de Alfaro and Henzinger [4, 5] as a *reactive type* theory that abstractly describes the communication behaviour of software or hardware components in terms of their inputs and outputs. IAs are labelled transition systems where visible actions are partitioned into inputs and outputs. The idea is that interfaces interact with their environment according to the following rules. An interface cannot block an incoming input in any state but, if an input arrives unexpectedly, it is treated as a catastrophic system failure. This means that, if a state does not enable an input, this is a requirement on the environment not to produce this input. Vice versa, an interface guarantees not to produce any unspecified outputs, which are in turn inputs to the environment.

This intuition is reflected in the specific refinement relation of *alternating simulation* between IA and in the *parallel composition* on IA, which have been defined in [5] and are recalled in this section. Most importantly, however, we introduce and study a *conjunction operator* on IA, which is needed in practice to reason about components that are expected to satisfy multiple interfaces.

Definition 1 (Interface Automata [5]). An *Interface Automaton* (IA) is a tuple $Q = (Q, I, O, \longrightarrow)$, where

1. Q is a set of states,
2. I and O are disjoint input and output alphabets, respectively, not containing the special (in contrast to [5] unique), silent action τ,
3. $\longrightarrow \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$ is the *transition relation*.

The transition relation is required to be *input-deterministic*, i.e., $a \in I$, $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ implies $q' = q''$. In the remainder, we write $q \xrightarrow{a}$ if $q \xrightarrow{a} q'$ for some q', as well as $q \not\xrightarrow{a}$ for its negation.

We let A stand for $I \cup O$, let a (α) range over A ($A \cup \{\tau\}$), and introduce the following weak transition relations: $q \overset{\varepsilon}{\Longrightarrow} q'$ if $q(\xrightarrow{\tau})^* q'$, and $q \overset{o}{\Longrightarrow} q'$ for $o \in O$ if $\exists q''.\, q \overset{\varepsilon}{\Longrightarrow} q'' \xrightarrow{o} q'$; note that there are no τ-transitions after the o-transition. Moreover, we define $\hat{\alpha} = \varepsilon$ if $\alpha = \tau$, and $\hat{\alpha} = \alpha$ otherwise.

Definition 2 (Alternating Simulation [5]). Let P and Q be IAs with common input and output alphabets. Relation $\mathcal{R} \subseteq P \times Q$ is an *alternating simulation relation* if for any $(p, q) \in \mathcal{R}$:

(i) $q \xrightarrow{a} q'$ and $a \in I$ implies $\exists p'.\, p \xrightarrow{a} p'$ and $(p', q') \in \mathcal{R}$,
(ii) $p \xrightarrow{\alpha} p'$ and $\alpha \in O \cup \{\tau\}$ implies $\exists q'.\, q \overset{\hat{\alpha}}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$.

We write $p \sqsubseteq_{\mathrm{IA}} q$ and say that p *IA-refines* q if there exists an alternating simulation relation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$.

According to the basic idea of IA, if specification Q in state q allows some input a delivered by the environment, then the related implementation state p of P must allow this input immediately in order to avoid system failure. Conversely, if P in state p produces output a to be consumed by the environment, this output must be expected by the environment even if $q \overset{a}{\not\Longrightarrow}$; this is because Q could have moved unobservedly from state q to some q' that enables a. Since inputs are not treated in Def. 2 (ii), they are always allowed for p.

It is easy to see that IA-refinement $\sqsubseteq_{\mathrm{IA}}$ is a preorder on IA. Given input and output alphabets I and O, respectively, the IA $\mathit{BlackHole}_{I,O} =_{\mathrm{df}} (\{\mathit{blackhole}\}, I, O, \{(\mathit{blackhole}, a, \mathit{blackhole}) \mid a \in I\})$ IA-refines any other IA over I and O.

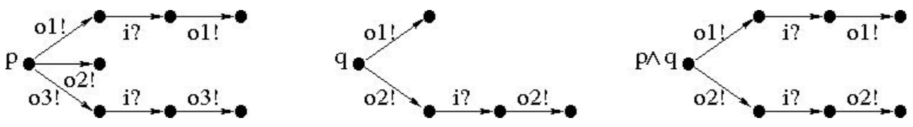


Fig. 1. Example illustrating IA-conjunction

2.1 Conjunction on IA

Two IAs with common alphabets are always logically consistent in the sense that they have a common implementation, e.g., the respective blackhole IA as noted above. This makes the definition of conjunction on IA relatively straightforward. Here and similarly later, we index a transition by the system's name to make clear from where it originates, in case this is not obvious from the context.

Definition 3 (Conjunction on IA). Let $P = (P, I, O, \longrightarrow_P)$ and $Q = (Q, I, O, \longrightarrow_Q)$ be IAs with common input and output alphabets and disjoint state sets P and Q. The conjunction $P \wedge Q$ is defined by $(\{p \wedge q \,|\, p \in P,\ q \in Q\} \cup P \cup Q, I, O, \longrightarrow)$, where $\longrightarrow$ is the least set satisfying $\longrightarrow_P \subseteq \longrightarrow$, $\longrightarrow_Q \subseteq \longrightarrow$ and the following operational rules:

(I1)	$p \wedge q \xrightarrow{a} p'$	if	$p \xrightarrow{a}_P p'$, $q \not\xrightarrow{a}_Q$ and $a \in I$
(I2)	$p \wedge q \xrightarrow{a} q'$	if	$p \not\xrightarrow{a}_P$, $q \xrightarrow{a}_Q q'$ and $a \in I$
(I3)	$p \wedge q \xrightarrow{a} p' \wedge q'$	if	$p \xrightarrow{a}_P p'$, $q \xrightarrow{a}_Q q'$ and $a \in I$
(O)	$p \wedge q \xrightarrow{a} p' \wedge q'$	if	$p \xrightarrow{a}_P p'$, $q \xrightarrow{a}_Q q'$ and $a \in O$
(T1)	$p \wedge q \xrightarrow{\tau} p' \wedge q$	if	$p \xrightarrow{\tau}_P p'$
(T2)	$p \wedge q \xrightarrow{\tau} p \wedge q'$	if	$q \xrightarrow{\tau}_Q q'$

Intuitively, conjunction is the synchronous product over actions (cf. Rules (I3), (O), (T1) and (T2)). Since inputs are always implicitly present, this also explains Rules (I1) and (I2); for example, in Rule (I1), q does not impose any restrictions on the behaviour after input a and is therefore dropped from the target state. Moreover, the conjunction operator is commutative and associative. As an aside, note that the rules with digit 2 in their names are the symmetric cases of the respective rules with digit 1; this convention will hold true throughout this paper. Fig. 1 applies the rules above to an illustrating example; here and in the following figures, we write $a?$ for an input a and $a!$ for an output a.

Theorem 4 (∧ is And). *Let P, Q, R be IAs with states p, q and r, respectively. Then, $r \sqsubseteq_{IA} p$ and $r \sqsubseteq_{IA} q$ if and only if $r \sqsubseteq_{IA} p \wedge q$.*

Hence, $\wedge$ gives the greatest lower-bound wrt. $\sqsubseteq_{\mathrm{IA}}$, i.e., an implementation satisfies the conjunction of interfaces exactly if it satisfies each of them. This is a desired property in system design where each interface describes one aspect of the overall specification. The above theorem also implies compositional reasoning; from universal algebra one easily gets:

Corollary 5. For IAs P, Q, R with states p, q and r: $p \sqsubseteq_{\mathrm{IA}} q \implies p \wedge r \sqsubseteq_{\mathrm{IA}} q \wedge r$.

2.2 Parallel Composition on IA

We recall the parallel composition operator | on IA of [5], which is defined in two stages: first a standard product $\otimes$ between two IAs is introduced, where common actions are synchronized and hidden. Then, error states are identified, and all states are pruned from which reaching an error state is unavoidable.

Definition 6 (Parallel Product on IA [5]). IAs P_1, P_2 are *composable* if $A_1 \cap A_2 = (I_1 \cap O_2) \cup (O_1 \cap I_2)$, i.e., each common action is input of one IA and output of the other IA. For such IAs we define the *product* $P_1 \otimes P_2 = (P_1 \times P_2, I, O, \longrightarrow)$, where $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$ and $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$ and where $\longrightarrow$ is given by the following operational rules:

(Par1) $(p_1, p_2) \xrightarrow{\alpha} (p_1', p_2)$ if $p_1 \xrightarrow{\alpha} p_1'$ and $\alpha \notin A_2$

(Par2) $(p_1, p_2) \xrightarrow{\alpha} (p_1, p_2')$ if $p_2 \xrightarrow{\alpha} p_2'$ and $\alpha \notin A_1$

(Par3) $(p_1, p_2) \xrightarrow{\tau} (p_1', p_2')$ if $p_1 \xrightarrow{a} p_1'$ and $p_2 \xrightarrow{a} p_2'$ for some a.

Note that, in case of synchronization and according to Rule (Par3), one only gets internal τ-transitions.

Definition 7 (Parallel Composition on IA [5]). A state (p_1, p_2) of a parallel product $P_1 \otimes P_2$ is an *error state* if there is some $a \in A_1 \cap A_2$ such that (a) $a \in O_1$, $p_1 \xrightarrow{a}$ and $p_2 \not\xrightarrow{a}$, or (b) $a \in O_2$, $p_2 \xrightarrow{a}$ and $p_1 \not\xrightarrow{a}$.

A state of $P_1 \otimes P_2$ is *incompatible* if it may reach an error state autonomously, i.e., only by output or internal actions that are, intuitively, locally controlled. Formally, the set $E \subseteq P_1 \times P_2$ of incompatible states is the least set such that $(p_1, p_2) \in E$ if (i) (p_1, p_2) is an error state or (ii) $(p_1, p_2) \xrightarrow{\alpha} (p_1', p_2')$ for some $\alpha \in O \cup \{\tau\}$ and $(p_1', p_2') \in E$.

The *parallel composition* $P_1 | P_2$ of P_1, P_2 is obtained from $P_1 \otimes P_2$ by *pruning*, i.e., removing all states in E and all transitions involving such states as source or target. If $(p_1, p_2) \in P_1 | P_2$, we write $p_1 | p_2$ and call p_1 and p_2 *compatible*.

Parallel composition is well-defined since input-determinism is preserved.

Theorem 8 (Compositionality of IA-Parallel Composition [5]). *Let P_1, P_2 and Q_1 be IAs with $p_1 \in P_1$, $p_2 \in P_2$, $q_1 \in Q_1$ and $p_1 \sqsubseteq_{IA} q_1$. Assume that Q_1 and P_2 are composable; then, (a) P_1 and P_2 are composable and (b) if q_1 and p_2 are compatible, then so are p_1 and p_2 and $p_1 | p_2 \sqsubseteq_{IA} q_1 | p_2$.*

This result relies on the fact that IAs are input-deterministic. While the theorem is already stated in [5], its proof is only sketched therein. Here, it is a simple corollary of Thms. 23 in Sec. 4.2 and Thms. 25 and 26(b) in Sec. 4.3 below.

We conclude by presenting a small example of IA-parallel composition in Fig. 2, which is adapted from [5]. The client does not accept its input *retry*. Thus, if the environment of *Client* $\otimes$ *TryOnce* would produce *nack*, the system would autonomously produce *reset* and run into a catastrophic error. To avoid this, the environment of *Client*| *TryOnce* is required not to produce *nack*. This view is called optimistic: there exists an environment in which *Client* and *Try-Once* can cooperate without errors, and *Client*| *TryOnce* describes the necessary requirements for such an environment. In the pessimistic view as advocated in [2], *Client* and *TryOnce* are regarded as incompatible due to the potential error.

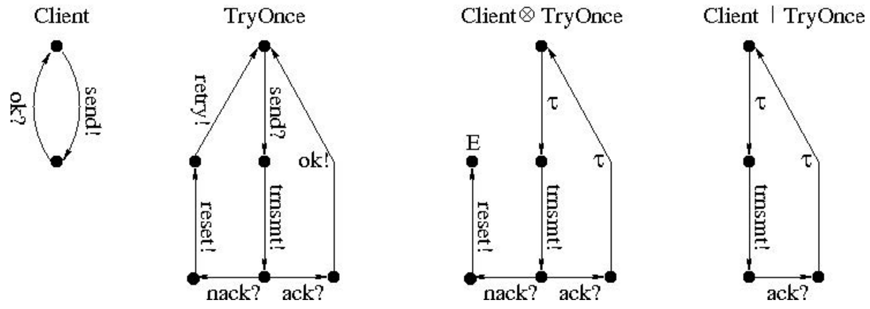


Fig. 2. Example illustrating IA-parallel composition, where IA *TryOnce* has inputs {*send, ack, nack*} and outputs {*trnsmt, ok, reset, retry*}, while IA *Client* has inputs {*ok, retry*} and outputs {*send*}

3 Conjunction for Modal Transition Systems

Modal Transition Systems (MTS) were investigated by Larsen [7] as a specification framework based on labelled transition systems but with two kinds of transitions: must-transitions specify required behaviour, may-transitions specify allowed behavior, and absent transitions specify forbidden behaviour. Any refinement of an MTS-specification must preserve required and forbidden behaviour and may turn allowed behaviour into required or forbidden behaviour. Technically, this is achieved via an alternating-style simulation relation, called *modal refinement*, where any must-transition of the specification must be simulated by an implementation, while any may-transition of the implementation must be simulated by the specification.

Larsen [7] defined conjunction on MTS, but the resulting systems often violate syntactic consistency and are hard to understand. To improve this, we allow an a-must-transition to have several alternative target states, i.e., we work with Disjunctive MTS (DMTS). Larsen and Xinxin also generalized Larsen's construction to DMTS [10], but again ignoring syntactic consistency. We will thus define conjunction on a syntactically consistent subclass of DMTS, called *dMTS*, but more generally in a setting with internal τ-actions as defined in [5, 8].

3.1 Disjunctive Modal Transition Systems

We extend standard MTS only as far as needed for defining conjunction, by introducing disjunctive must-transitions that are disjunctive wrt. exit states only (see Fig. 4). The following extension also has no τ-must-transitions since these are not considered in the definition of the observational modal refinement of [8].

Definition 9 (disjunctive Modal Transition System). A *disjunctive Modal Transition System* (dMTS) is a tuple $Q = (Q, A, \longrightarrow, \dashrightarrow)$, where

1. Q is a set of states,
2. A is an alphabet not containing the special, silent action τ,

3. $\longrightarrow \subseteq Q \times A \times (\mathcal{P}(Q) \setminus \emptyset)$ is the *must-transition* relation,
4. $\dashrightarrow \subseteq Q \times (A \cup \{\tau\}) \times Q$ is the *may-transition* relation.

We require *syntactic consistency*, i.e., $q \xrightarrow{a} Q'$ implies $\forall q' \in Q'.\, q \overset{a}{\dashrightarrow} q'$.

More generally, the must-transition relation in a standard DMTS [10] may be a subset of $Q \times (\mathcal{P}(A \times Q) \setminus \emptyset)$. For notational convenience, we write $q \xrightarrow{a} q'$ whenever $q \xrightarrow{a} \{q'\}$; all must-transitions in standard MTS have this form.

Our refinement relation on dMTS abstracts from internal computation steps in the same way as [8], i.e., by considering the following *weak may-transitions* for $\alpha \in A \cup \{\tau\}$: $q \overset{\varepsilon}{\Longrightarrow} q'$ if $q \overset{\tau}{\dashrightarrow}^{*} q'$, and $q \overset{\alpha}{\Longrightarrow} q'$ if $\exists q''.\, q \overset{\varepsilon}{\Longrightarrow} q'' \overset{\alpha}{\dashrightarrow} q'$.

Definition 10 (Observational Modal Refinement, see [8]). Let P, Q be dMTSs with common alphabet. Relation $\mathcal{R} \subseteq P \times Q$ is an *(observational) modal refinement relation* if for any $(p, q) \in \mathcal{R}$:

(i) $q \xrightarrow{a} Q'$ implies $\exists P'.\, p \xrightarrow{a} P'$ and $\forall p' \in P'\, \exists q' \in Q'.\, (p', q') \in \mathcal{R}$,
(ii) $p \overset{\alpha}{\dashrightarrow} p'$ implies $\exists q'.\, q \overset{\hat{\alpha}}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$.

We write $p \sqsubseteq_{\mathrm{dMTS}} q$ and say that p *dMTS-refines* q if there exists an observational modal refinement relation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$.

Except for disjunctiveness, dMTS-refinement is exactly defined as for MTS in [8], i.e., the τ-must-transitions allowed in their variant of MTS are not treated in Cond. (i) of observational modal refinement. Thus, they are treated as only may-transitions and not included in our setting.

3.2 Conjunction on dMTS

Similarly to parallel composition for IA, conjunction will be defined in two stages. State pairs can be logically inconsistent due to unsatisfiable must-transitions; in the second stage, we remove such pairs incrementally.

Definition 11 (Conjunctive Product on dMTS). Let $P = (P, A, \longrightarrow_P, \dashrightarrow_P)$ and $Q = (Q, A, \longrightarrow_Q, \dashrightarrow_Q)$ be dMTSs with common alphabet. The conjunctive product $P \& Q =_{\mathrm{df}} (P \times Q, A, \longrightarrow, \dashrightarrow)$ is defined by its operational transition rules as follows:

(Must1)	$(p, q) \xrightarrow{a} \{(p', q') \mid p' \in P',\, q \overset{a}{\Longrightarrow}_Q q'\}$	if $p \xrightarrow{a}_P P'$ and $q \overset{a}{\Longrightarrow}_Q$
(Must2)	$(p, q) \xrightarrow{a} \{(p', q') \mid p \overset{a}{\Longrightarrow}_P p',\, q' \in Q'\}$	if $p \overset{a}{\Longrightarrow}_P$ and $q \xrightarrow{a}_Q Q'$
(May1)	$(p, q) \overset{\tau}{\dashrightarrow} (p', q)$	if $p \overset{\tau}{\Longrightarrow}_P p'$
(May2)	$(p, q) \overset{\tau}{\dashrightarrow} (p, q')$	if $q \overset{\tau}{\Longrightarrow}_Q q'$
(May3)	$(p, q) \overset{\alpha}{\dashrightarrow} (p', q')$	if $p \overset{\alpha}{\Longrightarrow}_P p'$ and $q \overset{\alpha}{\Longrightarrow}_Q q'$

It might be surprising that a single transition in the product might stem from a transition sequence in one of the components (cf. the first four items above) and that the components can also synchronize on τ (cf. Rule (May3)). The necessity of this is discussed below; we only note here that conjunction is inherently different from parallel composition.

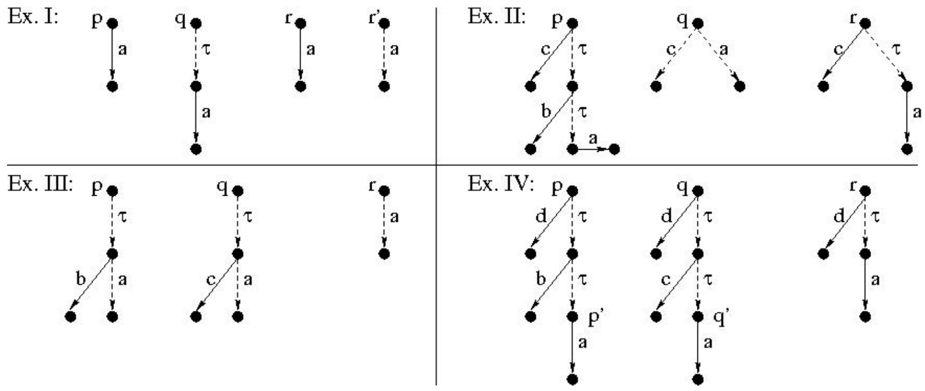


Fig. 3. Examples motivating the rules of Def. 11

Definition 12 (Conjunction on dMTS). Given a conjunctive product $P\&Q$, the set $F \subseteq P \times Q$ of *(logically) inconsistent states* is defined as the least set satisfying the following rules:

(F1)	$p \xrightarrow{a}_P,\ q \overset{a}{\not\dashrightarrow}_Q$	implies	$(p,q) \in F$
(F2)	$p \overset{a}{\not\dashrightarrow}_P,\ q \xrightarrow{a}_Q$	implies	$(p,q) \in F$
(F3)	$(p,q) \xrightarrow{a} R'$ and $R' \subseteq F$	implies	$(p,q) \in F$

The conjunction $P \wedge Q$ of dMTSs P, Q is obtained by deleting all states $(p,q) \in F$ from $P\&Q$. This also removes any may- or must-transition exiting a deleted state and any may-transition entering a deleted state; in addition, deleted states are removed from targets of disjunctive must-transitions. We write $p \wedge q$ for the state (p,q) of $P \wedge Q$; these are the consistent states by construction, and $p \wedge q$ is only defined for such a state.

Regarding well-definedness, first observe that $P\&Q$ is a dMTS, where syntactic consistency follows from Rule (May3). Now, $P \wedge Q$ is a dMTS, too: if R' becomes empty for some $(p,q) \xrightarrow{a} R'$, then also (p,q) is deleted when constructing $P \wedge Q$ from $P\&Q$ according to (F3).

Before we formally state that operator $\wedge$ is indeed conjunction on dMTS, we present several examples depicted in Fig. 3, which motivate the rules of Def. 11. Note that, in this figure and the following figures, any (disjunctive) must-transition drawn also represents implicitly the respective may-transition(s), unless stated otherwise. In each example in Fig. 3, r is a common implementation of p and q (but not r' in Ex. I), whence these must be logically consistent. Thus, Ex. I explains Rule (Must1). If we only had $\overset{\tau}{\dashrightarrow}$ in the precondition of Rule (May1), $p \wedge q$ of Ex. II would just consist of a c-must- and an a-may-transition; the only τ-transition would lead to a state in F due to b. This would not allow the τ-transition of r, explaining Rule (May1). In Ex. III and with only $\overset{\alpha}{\dashrightarrow}$ in the preconditions of Rule (May3), $p \wedge q$ would just have three τ-transitions

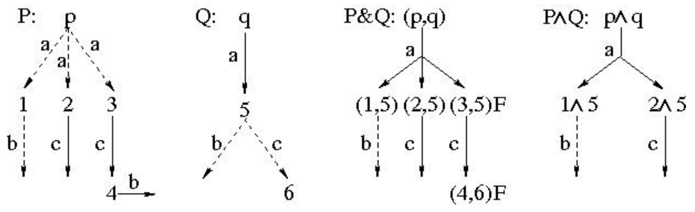


Fig. 4. Example illustrating dMTS-conjunction

to inconsistent states (due to b, c, respectively). This explains the weak transitions for $\alpha \neq \tau$ in Rule (May3). According to Rules (May1) and (May2), $p \wedge q$ in Ex. IV has four τ-transitions to states in F (due to d). With preconditions based on at least one $\overset{\tau}{\dashrightarrow}$ instead of $\overset{\tau}{\Rrightarrow}$ in the τ-case of Rule (May3), there would be three more τ-transitions to states in F (due to b or c). Thus, it is essential that Rule (May3) also allows the synchronization of two weak τ-transitions, which in this case gives $p \wedge q \overset{\tau}{\dashrightarrow} p' \wedge q'$.

Fig. 4 shows a small example illustrating the treatment of disjunctive must-transitions in the presence of inconsistency. In $P\&Q$, the a-must-transition of Q combines with the three a-transitions of P to a truly disjunctive must-transition with a three-element target set. The inconsistency of state $(4,6)$ due to b propagates back to state $(3,5)$. The inconsistent states are removed in $P \wedge Q$.

Theorem 13 ($\wedge$ is And). *Let P, Q, R be dMTSs. Then, (i) $(\exists r \in R.\, r \sqsubseteq_{dMTS} p$ and $r \sqsubseteq_{dMTS} q)$ if and only if $p \wedge q$ is defined. Further, in case $p \wedge q$ is defined: (ii) $r \sqsubseteq_{dMTS} p$ and $r \sqsubseteq_{dMTS} q$ if and only if $r \sqsubseteq_{dMTS} p \wedge q$.*

This key theorem states in Item (ii) that conjunction behaves as it should, i.e., $\wedge$ on dMTSs is the greatest lower bound wrt. $\sqsubseteq_{\text{dMTS}}$. Item (i) concerns the intuition that two specifications p and q are logically inconsistent if they do not have a common implementation; formally, $p \wedge q$ is undefined in this case. Alternatively, we could have added an explicit inconsistent element *ff* to our setting, so that $p \wedge q = \mathit{ff}$. This element *ff* would be defined to be a refinement of every p' and equivalent to any $(p', q') \in F$ of some $P\&Q$. Additionally, $\mathit{ff} \wedge p'$ and $p' \wedge \mathit{ff}$ would be defined as *ff*, for any p'. The following corollary of Thm. 13 now follows from universal algebra, as above:

Corollary 14. dMTS-refinement is compositional wrt. conjunction.

Thus, we have succeeded in our ambition to define a syntactically consistent conjunction for MTS, albeit for an MTS variant with disjunctive must-transitions.

Larsen [7] also defines a conjunction operator on MTS, but almost always the result violates syntactic consistency. A simple example is shown in Fig. 5, where q refines p in Larsen's setting as well as in our dMTS-setting. Since Larsen's $p \wedge q$ is not syntactically consistent, this $p \wedge q$ and q are, contrary to the first impression, equivalent. In our dMTS-setting $P \wedge Q$ is isomorphic to Q, which will also hold

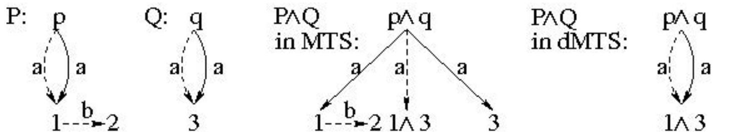


Fig. 5. Example illustrating Larsen's MTS-conjunction; $\overset{a}{\dashrightarrow}$ drawn separately

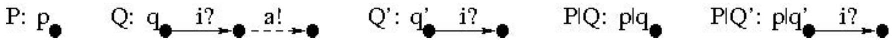


Fig. 6. Example demonstrating the compositionality flaw of IOMTS

for our MIA-setting below (with action b read as output and where a could be either an input or an output).

The above shortcoming has been avoided by Larsen et al. in [9] by limiting conjunction to so-called *independent* specifications that make inconsistencies obsolete. This restriction also excludes the above example. Another MTS-inspired theory including a conjunction operator has been introduced by Raclet et al. [12]. While their approach yields the desired $p \wedge q$ as in our dMTS-setting, it is language-based and thus deals with deterministic systems only.

4 Modal Interface Automata

An essential point of Larsen, Nyman and Wasowski's seminal paper [8] is to enrich IA with modalities to get a flexible specification framework where inputs and outputs can be prescribed, allowed or prohibited. To do so, they consider IOMTS, i.e., MTS where visible actions are partitioned into inputs and outputs, and define parallel composition in IA-style.

Our example of Fig. 6 shows that their approach has a serious flaw, namely observational modal refinement is not a precongruence for the parallel composition of [8]. In this example, the IOMTS P has input alphabet $\{a\}$ and empty output alphabet, while Q and Q' have input alphabet $\{i\}$ and output alphabet $\{a\}$. Obviously, $q' \sqsubseteq_{\text{dMTS}} q$. When composing P and Q in parallel, $p|q$ would reach an error state after an i-must-transition in [8] since the potential output a of Q is not expected by P. In contrast, $p|q'$ has an i-must- and i-may-transition not allowed by $P|Q$, so that $p|q' \not\sqsubseteq_{\text{dMTS}} p|q$. This counterexample also holds for (strong) modal refinement as defined in [8] and is particularly severe since all systems are deterministic. The problem is that $p|q$ forbids input i.

In [8], precongruence of parallel composition is not mentioned. Instead, a theorem relates the parallel composition of two IOMTSs to a different composition on two refining implementations, where an implementation in [8] is an IOMTS in which may- and must-transitions coincide. This theorem is incorrect as is pointed out in [12] and repaired in the deterministic setting of that paper; the repair is still not a precongruence result, but compares the results of two different operators. However, a natural solution to the precongruence problem can be adopted from the IA-framework [5] where inputs are always allowed implicitly. Consequently, if an input is specified, it will always be a must.

In the remainder, we thus define and study a new specification framework, called *Modal Interface Automata* (MIA), that takes the dMTS-setting for an alphabet consisting of input and output actions, requires input-determinism, and demands that every input-may-transition is also an input-must-transition. The advantage over IA is that outputs can be prescribed via output-must-transitions, which precludes trivial implementations like *BlackHole* discussed in Sec. 2.

Definition 15 (Modal Interface Automaton). A *Modal Interface Automaton* (MIA) is a tuple $Q = (Q, I, O, \longrightarrow, \dashrightarrow)$, where $(Q, I \cup O, \longrightarrow, \dashrightarrow)$ is a dMTS with disjoint alphabets I and O for inputs and outputs and where for all $i \in I$: (a) $q \overset{i}{\dashrightarrow} q'$ and $q \overset{i}{\dashrightarrow} q''$ implies $q' = q''$; (b) $q \overset{i}{\dashrightarrow} q'$ implies $q \xrightarrow{i} q'$.

Observe that syntactic consistency and input determinism imply that input-must-transitions always have the form $q \xrightarrow{i} \{q'\}$. Thus, only output-must-transitions can be truly disjunctive.

Definition 16 (MIA-Refinement). Let P, Q be MIAs with common input and output alphabets. Relation $\mathcal{R} \subseteq P \times Q$ is an *(observational) MIA-refinement relation* if for any $(p, q) \in \mathcal{R}$:

(i) $q \xrightarrow{a} Q'$ implies $\exists P'.\, p \xrightarrow{a} P'$ and $\forall p' \in P' \, \exists q' \in Q'.\, (p', q') \in \mathcal{R}$,
(ii) $p \overset{\alpha}{\dashrightarrow} p'$ with $\alpha \in O \cup \{\tau\}$ implies $\exists q'.\, q \overset{\hat{\alpha}}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$.

We write $p \sqsubseteq_{\mathrm{MIA}} q$ and say that p *MIA-refines* q if there exists an observational MIA-refinement relation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$. Moreover, we also write $p \sqsupseteq\sqsubseteq_{\mathrm{MIA}} q$ in case $p \sqsubseteq_{\mathrm{MIA}} q$ and $q \sqsubseteq_{\mathrm{MIA}} p$ (which is an equivalence weaker than 'bisimulation').

One can easily check that $\sqsubseteq_{\mathrm{MIA}}$ is a preorder and the largest observational MIA-refinement relation. Its definition coincides with dMTS-refinement except that Cond. (ii) is restricted to outputs and the silent action τ. Thus, inputs are always allowed implicitly and, in effect, treated just like in IA-refinement. Due to the output-must-transitions in the MIA-setting, MIA-refinement can model, e.g., STG-bisimilarity [13] for digital circuits.

4.1 Conjunction on MIA

Similar to conjunction on dMTS, we define conjunction on MIA by first constructing a conjunctive product and then eliminating all inconsistent states.

Definition 17 (Conjunctive Product on MIA). Let $P = (P, I, O, \longrightarrow_P, \dashrightarrow_P)$ and $Q = (Q, I, O, \longrightarrow_Q, \dashrightarrow_Q)$ be MIAs with common input and output alphabets and disjoint state sets P and Q. The conjunctive product $P \& Q =_{\mathrm{df}} ((P \times Q) \cup P \cup Q, I, O, \longrightarrow, \dashrightarrow)$ inherits the transitions of P and Q and has additional transitions as follows, where $i \in I$, $o \in O$ and $\alpha \in O \cup \{\tau\}$:

$$\begin{array}{lll}
\textit{(OMust1)} & (p,q) \xrightarrow{o} \{(p',q') \,|\, p' \in P',\, q \overset{o}{\dashrightarrow}_Q q'\} & \text{if } p \xrightarrow{o}_P P' \text{ and } q \overset{o}{\dashrightarrow}_Q \\
\textit{(OMust2)} & (p,q) \xrightarrow{o} \{(p',q') \,|\, p \overset{o}{\dashrightarrow}_P p',\, q' \in Q'\} & \text{if } p \overset{o}{\dashrightarrow}_P \text{ and } q \xrightarrow{o}_Q Q' \\
\textit{(IMust1)} & (p,q) \xrightarrow{i} p' & \text{if } p \xrightarrow{i}_P p' \text{ and } q \not\xrightarrow{i}_Q \\
\textit{(IMust2)} & (p,q) \xrightarrow{i} q' & \text{if } p \not\xrightarrow{i}_P \text{ and } q \xrightarrow{i}_Q q' \\
\textit{(IMust3)} & (p,q) \xrightarrow{i} (p',q') & \text{if } p \xrightarrow{i}_P p' \text{ and } q \xrightarrow{i}_Q q' \\
\textit{(May1)} & (p,q) \overset{\tau}{\dashrightarrow} (p',q) & \text{if } p \overset{\tau}{\dashrightarrow}_P p' \\
\textit{(May2)} & (p,q) \overset{\tau}{\dashrightarrow} (p,q') & \text{if } q \overset{\tau}{\dashrightarrow}_Q q' \\
\textit{(May3)} & (p,q) \overset{\alpha}{\dashrightarrow} (p,q') & \text{if } p \overset{\alpha}{\dashrightarrow}_P p' \text{ and } q \overset{\alpha}{\dashrightarrow}_Q q'
\end{array}$$

(Plus the may-rules corresponding to Rules (IMust1)–(IMust3) above.)

This product is defined analogously to IA-conjunction for inputs (plus the corresponding 'may' rules) and to the dMTS-product for outputs and τ. It thus combines the effects shown in Fig. 1 (where all outputs are treated as may) and Fig. 4 (where all actions are outputs).

Definition 18 (Conjunction on MIA). Given a conjunctive product $P\&Q$, the set $F \subseteq P \times Q$ of (logically) *inconsistent states* is defined as the least set satisfying the following rules:

$$\begin{array}{lll}
\textit{(F1)} & p \xrightarrow{o}_P,\; q \not\overset{o}{\dashrightarrow}_Q,\; o \in O & \text{implies} \quad (p,q) \in F \\
\textit{(F2)} & p \not\overset{o}{\dashrightarrow}_P,\; q \xrightarrow{o}_Q,\; o \in O & \text{implies} \quad (p,q) \in F \\
\textit{(F3)} & (p,q) \xrightarrow{a} R' \text{ and } R' \subseteq F & \text{implies} \quad (p,q) \in F
\end{array}$$

The conjunction $P \wedge Q$ of MIAs P, Q with common input and output alphabets is obtained by deleting all states $(p,q) \in F$ from $P\&Q$. We write $p \wedge q$ for state (p,q) of $P \wedge Q$; all such states are defined – and consistent – by construction.

The conjunction $P \wedge Q$ is a MIA and is thus well-defined. This can be seen by a similar argument as we have used above in the context of dMTS-conjunction, while input-determinism can be established by an argument similar to that in the IA-setting. Note that, in contrast to the dMTS situation, Rules (F1) and (F2) only apply to outputs. Fig. 4 is also an example for conjunction in the MIA-setting if all actions are read as outputs.

Theorem 19 ($\wedge$ is And). *Let P, Q, R be MIAs. We have (i) ($\exists r \in R.\, r \sqsubseteq_{MIA} p$ and $r \sqsubseteq_{MIA} q$) if and only if $p \wedge q$ is defined. Further, in case $p \wedge q$ is defined: (ii) $r \sqsubseteq_{MIA} p$ and $r \sqsubseteq_{MIA} q$ if and only if $r \sqsubseteq_{MIA} p \wedge q$.*

Corollary 20. MIA-refinement is compositional wrt. conjunction.

4.2 Parallel Composition on MIA

In analogy to the IA-setting [5], we provide a parallel operator on MIA. Here, error states are identified, and all states are removed from which reaching an error state is unavoidable in some implementation, as is done for IOMTS in [8].

Definition 21 (Parallel Product on MIA). MIAs P_1, P_2 are *composable* if $A_1 \cap A_2 = (I_1 \cap O_2) \cup (O_1 \cap I_2)$, as in IA. For such MIAs we define the *product* $P_1 \otimes P_2 = (P_1 \times P_2, I, O, \longrightarrow, \dashrightarrow)$, where $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$ and $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$ and where $\longrightarrow$ and $\dashrightarrow$ are defined as follows:

(Must1)	$(p_1, p_2) \xrightarrow{a} P'_1 \times \{p_2\}$	if	$p_1 \xrightarrow{a} P'_1$ and $a \notin A_2$
(Must2)	$(p_1, p_2) \xrightarrow{a} \{p_1\} \times P'_2$	if	$p_2 \xrightarrow{a} P'_2$ and $a \notin A_1$
(May1)	$(p_1, p_2) \overset{\alpha}{\dashrightarrow} (p'_1, p_2)$	if	$p_1 \overset{\alpha}{\dashrightarrow} p'_1$ and $\alpha \notin A_2$
(May2)	$(p_1, p_2) \overset{\alpha}{\dashrightarrow} (p_1, p'_2)$	if	$p_2 \overset{\alpha}{\dashrightarrow} p'_2$ and $\alpha \notin A_1$
(May3)	$(p_1, p_2) \overset{\tau}{\dashrightarrow} (p'_1, p'_2)$	if	$p_1 \overset{a}{\dashrightarrow} p'_1$ and $p_2 \overset{a}{\dashrightarrow} p'_2$ for some a.

Recall that there are no τ-must-transitions as they are irrelevant for refinement.

Definition 22 (Parallel Composition on MIA). Given a parallel product $P_1 \otimes P_2$, a state (p_1, p_2) is an *error state* if there is some $a \in A_1 \cap A_2$ such that (a) $a \in O_1$, $p_1 \overset{a}{\dashrightarrow}$ and $p_2 \overset{a}{\not\longrightarrow}$, or (b) $a \in O_2$, $p_2 \overset{a}{\dashrightarrow}$ and $p_1 \overset{a}{\not\longrightarrow}$.

Again we define the set $E \subseteq P_1 \times P_2$ of *incompatible* states as the least set such that $(p_1, p_2) \in E$ if (i) (p_1, p_2) is an error state or (ii) $(p_1, p_2) \overset{\alpha}{\dashrightarrow} (p'_1, p'_2)$ for some $\alpha \in O \cup \{\tau\}$ and $(p'_1, p'_2) \in E$. The *parallel composition* $P_1 | P_2$ of P_1 and P_2 is now obtained from $P_1 \otimes P_2$ by *pruning*, as in IA.

Parallel products and parallel compositions are well-defined MIAs. Syntactic consistency is preserved, as is input-determinism since input-transitions are directly inherited from one of the *composable* systems. In addition, targets of disjunctive must-transitions are never empty since all must-transitions that remain after pruning are taken from the product without modification.

Observe that pruning is different from removing inconsistent states in conjunction. For truly disjunctive transitions $(p_1, p_2) \xrightarrow{a} P'$ (i.e., $a \in O$) of the product $P_1 \otimes P_2$, the state (p_1, p_2) is removed if $P' \cap E \neq \emptyset$. Technically, this follows from syntactic consistency and Cond. (ii) above. Intuitively, this is because P' has w.l.o.g. the form $P'_1 \times \{p_2\}$ in the product of P_1 and P_2, with some $(p'_1, p'_2) \in P' \cap E$; the implementor of P_1 might choose to implement $p_1 \xrightarrow{a} p'_1$ such that – when P_1's implementation is composed with P_2's – the error state is reached. This cannot be reasonably prevented by altering the above definition while preserving the precongruence property for parallel composition:

Theorem 23 (Compositionality of MIA-Parallel Composition). *Let P_1, P_2 and Q_1 be MIAs with $p_1 \in P_1$, $p_2 \in P_2$, $q_1 \in Q_1$ and $p_1 \sqsubseteq_{MIA} q_1$. Assume that Q_1 and P_2 are composable; then, (a) P_1 and P_2 are composable and (b) if q_1 and p_2 are compatible, then so are p_1 and p_2 and $p_1 | p_2 \sqsubseteq_{MIA} q_1 | p_2$.*

This precongruence property of MIA-refinement would not hold if we would do away with input-determinism in MIA. To see this, consider the example of Fig. 7 for which $p \sqsubseteq_{\text{MIA}} q$; however, $p|r \sqsubseteq_{\text{MIA}} q|r$ does not hold since q and r are compatible while p and r are not. An analogue reasoning applies to IA, although we do not know of a reference in the IA literature where this has been observed.

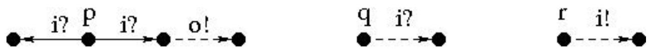


Fig. 7. Example illustrating the need of input-determinism for MIA

4.3 Embedding of IA into MIA

To conclude, we provide an embedding of IA into MIA in the line of [8]:

Definition 24 (IA-Embedding). Let P be an IA. The embedding $[P]_{\mathrm{MIA}}$ of P into MIA is defined as the MIA $(P, I, O, \longrightarrow, \dashrightarrow)$, where (i) $p \xrightarrow{i} p'$ if $p \xrightarrow{i}_P p'$ and $i \in I$, and (ii) $p \overset{\alpha}{\dashrightarrow} p'$ if $p \xrightarrow{\alpha}_P p'$ and $\alpha \in I \cup O \cup \{\tau\}$.

This embedding is much simpler than the one of [8] since MIA more closely resembles IA than IOMTS does. In particular, the following theorem is obvious:

Theorem 25 (IA-Embedding Respects Refinement). *For IAs P, Q with $p \in P$, $q \in Q$: $p \sqsubseteq_{IA} q$ in P and Q iff $p \sqsubseteq_{MIA} q$ in $[P]_{MIA}$ and $[Q]_{MIA}$.*

Our embedding respects operators | and $\wedge$, unlike the one in [8]:

Theorem 26 (IA-Embedding is a Homomorphism). *For IAs P, Q with $p \in P$, $q \in Q$: (a) $p \wedge q$ (in $[P]_{MIA} \wedge [Q]_{MIA}$) $\sqsupseteq\sqsubseteq_{MIA}$ $p \wedge q$ (in $[P \wedge Q]_{MIA}$); (b) $p|q$ (in $[P]_{MIA}|[Q]_{MIA}$) $\sqsupseteq\sqsubseteq_{MIA}$ $p|q$ (in $[P|Q]_{MIA}$).*

5 Conclusions and Future Work

We introduced *Modal Interface Automata* (MIA), an interface theory that is more expressive than *Interface Automata* (IA) [5]: it allows one to mandate that a specification's refinement must implement some output, thus excluding trivial implementations, e.g., one that accepts all inputs but never emits any output. This was also the motivation behind *IOMTS* [8] that extends *Modal Transition Systems* (MTS) [7] by inputs and outputs; however, the IOMTS-parallel operator in the style of IA is not compositional. MIA is a subset of IOMTS, but it has a different refinement relation that is a precongruence for parallel composition.

Most importantly and in contrast to IA and IOMTS, the MIA theory is equipped with a conjunction operator for reasoning about components that satisfy multiple interfaces simultaneously. Along the way, we also introduced conjunction on IA and (a disjunctive extension of) MTS, and proved these operators to be the desired greatest lower bounds and thus compositional. Compared to the language-based modal interface theory of [12], our formalism supports nondeterministic specifications (wrt. outputs). Hence, MIA establishes a theoretically clean and practical interface theory that fixes the shortcomings of related work.

Regarding future work, we plan to study the algorithmic complexity implied by MIA-refinement [12], and MIA's expressiveness in comparison to other theories via thoroughness [6]. On the practical side, we wish to adapt existing tool support for interface theories, e.g., the *MIO Workbench* [3], to MIA.

References

[1] Abadi, M., Lamport, L.: Conjoining specifications. ACM TOPLAS 1(3), 507–534 (1995)

[2] Bauer, S., Hennicker, R., Wirsing, M.: Interface theories for concurrency and data. Theoret. Comp. Sc. 412(28), 3101–3121 (2011)

[3] Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On Weak Modal Compatibility, Refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer, Heidelberg (2010)

[4] de Alfaro, L., Henzinger, T.A.: Interface automata. In: FSE, pp. 109–120. ACM (2001)

[5] de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Engineering Theories of Software-Intensive Systems. NATO Science Series, vol. 195. Springer (2005)

[6] Fecher, H., de Frutos-Escrig, D., Lüttgen, G., Schmidt, H.: On the Expressiveness of Refinement Settings. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 276–291. Springer, Heidelberg (2010)

[7] Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)

[8] Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)

[9] Larsen, K.G., Steffen, B., Weise, C.: A Constraint Oriented Proof Methodology Based on Modal Transition Systems. In: Brinksma, E., Steffen, B., Cleaveland, W.R., Larsen, K.G., Margaria, T. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 17–40. Springer, Heidelberg (1995)

[10] Larsen, K.G., Xinxin, L.: Compositionality through an operational semantics of contexts. J. Logic Comput. 1(6), 761–795 (1991)

[11] Lüttgen, G., Vogler, W.: Ready simulation for concurrency: It's logical! Inform. and Comput. 208, 845–867 (2010)

[12] Raclet, J., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: A modal interface theory for component-based design. Fund. Inform. 107, 1–32 (2011)

[13] Schäfer, M., Vogler, W.: Component refinement and CSC-solving for STG decomposition. Theoret. Comp. Sc. 388(1-3), 243–266 (2007)

Proofs as Executions

Emmanuel Beffara[1,⋆] and Virgile Mogbil[2,⋆⋆]

[1] IML – FRE3529, CNRS – Université d'Aix-Marseille
[2] LIPN – UMR7030, CNRS – Université Paris 13

Abstract. This paper proposes a new interpretation of the logical contents of programs in the context of concurrent interaction, wherein proofs correspond to valid executions of a processes. A type system based on linear logic is used, in which a given process has many different types, each typing corresponding to a particular way of interacting with its environment and cut elimination corresponds to executing the process in a given interaction scenario. A completeness result is established, stating that every lock-avoiding execution of a process in some environment corresponds to a particular typing. Besides traces, types contain precise information about the flow of control between a process and its environment, and proofs are interpreted as composable schedulings of processes. In this interpretation, logic appears as a way of making explicit the flow of causality between interacting processes.

1 Introduction

The extension of the familiar Curry-Howard correspondence to interactive models of computation has been an active research topic for several decades. Several systems were proposed based on linear logic [11], following the fundamental observation that it is a logic of interaction. Interpretations of proofs as processes, first formalized by Abramsky [1], later refined by various people including the first author [2], stressed that proof nets [12] and process calculi have significant similarities in dynamics. At the same time, type systems for concurrency [24] revealed to be equivalent to variants of linear logic [14,6]. These approaches successfully stress the fact that concurrent calculi are very expressive and versatile models of interactive behaviour, however they are not satisfactory yet as a proof-theoretical account of concurrency, because they tend to impose determinism in execution, effectively constraining processes to functional behaviour.

Several approaches to the question of non-determinism in proof theory have been proposed using the additives of linear logic [16,17,21]. In a different style, differential logic was recently developed by Ehrhard and Regnier [10] and its untyped proof formalism was shown expressive enough to represent the π-calculus [9]. The present work proposes a different approach to the topic, by questioning the "proofs-as-programs" paradigm. Proof theory wants cut elimination to be confluent, because the meaning of proofs lies in their normal forms. On the other

⋆ Supported by the French ANR project LOGOI, ANR-10-BLAN-0213.
⋆⋆ Supported by the French ANR project COMPLICE, ANR-08-BLAN-0211-01.

hand, reduction in process calculi is execution: the meaning of a term is not its final irreducible form but what happens to get there, as interaction with other processes. Hence we propose to match proofs with executions rather than terms. But this raises the new question of what is the logical meaning of an execution. Here we must remember that cut elimination is a process of *explicitation* and cut-free proofs are explicit, direct reasonings justifying some fact. In our case, the fact is the interaction, which is a scheduling of a set of events in a system. The justification, then, is the *control flow* through the system, specifying when actions happen and when execution jumps from one process to another.

Technically, we illustrate this idea in the very simple setting of finitary CCS with no choice operator, in order to focus on the novel ideas of our approach, but ways to extend these techniques to a larger class of processes are sketched in the perspectives. The corresponding logic is multiplicative linear logic, with a family of modalities à la Hennessy-Milner [13] representing actions.

In our type system, multiplicatives represent causality and independence between parts of a run, using connectedness/acyclicity arguments to describe avoidance of deadlocks. Modalities represent observable transitions, with explicit scheduling constraints using the well-known stratifying effect of boxes in proof nets. Axiom rules have an unusual interpretation: they are void of interactive content (no forwarding or copycat behaviour), but they logically implement the transfer of control flow between different parts of a running process.

Comparison to other work. This handling of control flow using the symmetries of linear logic is reminiscent of the work of Mazurak and Zdancewicz [18] who use linear negation as an explicit scheduling operation. Our work differs from theirs and other works on typing for concurrency, in that we proceed "backwards": while Curry-Howard systems for concurrency embed logical systems into concurrent calculi, we embed executions of processes into a logical system.

The idea of matching proofs with executions is reminiscent of the *proof search* approach to computation. Indeed, the relationship between logical linearity and interaction has been explored for instance by Miller and Tiu [19,22] in sequent calculus and by Bruscoli [5] in deep inference. Our approach has fundamentally different bases: in these works, formulas are programs and proofs are reduction sequences, while in our settings a formula is an interaction scenario and proofs describe how a process can act according to this scenario, following its syntactic structure. Moreover, internal dynamics in processes actually corresponds to cut-elimination, which sets our work closer to proofs-as-programs than proof search.

Outline. The paper is organized as follows: Section 2 introduces a logic of schedulings based on linear logic and illustrates our interpretation. Section 3 defines a simple fragment of CCS and a notion of determinisation, used to represent executions as terms. Section 4 presents the proof nets for the logic of schedulings and its cut-elimination. Section 5 shows the typing of executions and the associated subject reduction property, and Section 6 establishes the completeness property that all lock-avoiding executions are typable. Appendices with detailed proofs can be found on the online version [4].

Table 1. Inference rules for MLL with action modalities (MLL$_\mathrm{a}$)

$$\frac{P \vdash \Gamma, A, B}{P \vdash \Gamma, A \parr B}\,(\parr) \qquad \frac{P \vdash \Gamma, A \quad Q \vdash B, \Delta}{P \mid Q \vdash \Gamma, A \otimes B, \Delta}\,(\otimes) \qquad \frac{P \vdash \Gamma, A \qquad Q \vdash A^\perp, \Delta}{P \mid Q \vdash \Gamma, \Delta}\,(\text{cut})$$

$$\frac{}{1 \vdash A, A^\perp}\,(\text{ax}) \qquad \frac{P \vdash \Gamma, A}{a.P \vdash \Gamma, \langle a \rangle A}\,(\text{act}) \qquad \frac{P \vdash \Gamma \quad a \notin \Gamma}{(\boldsymbol{\nu} a)P \vdash \Gamma}\,(\text{new})$$

$$\text{Derived rules:} \begin{cases} \dfrac{P : \Gamma, A \vdash B}{P : \Gamma \vdash A \multimap B}\,(\multimap\text{R}) \qquad \dfrac{P : \Gamma \vdash A \quad Q : \Delta, B \vdash C}{P \mid Q : \Gamma, \Delta, A \multimap B \vdash C}\,(\multimap\text{L}) \\ \dfrac{P : \Gamma \vdash A \multimap B \quad Q : \Delta \vdash A}{P \mid Q : \Gamma, \Delta \vdash B}\,(\text{mp}) \qquad \dfrac{P : \Gamma, A \vdash B}{a.P : \Gamma, \langle \bar{a} \rangle A \vdash B}\,(\text{act}) \end{cases}$$

– Rule (act) applies for names of both polarities.
– In rule (new), $a \notin \Gamma$ means that neither $\langle a \rangle$ nor $\langle \overline{a} \rangle$ occurs in Γ.

2 A Logic of Schedulings

We first present the logic we use to describe interactions and schedulings. It corresponds to the multiplicative fragment of linear logic [11], augmented with a family of modalities that describe actions. Note that this section introduces the logic in sequent calculus for simplicity, but the proper formalism for establishing our results is that of proof nets, presented in section 4; this choice of presentation is (hopefully) more pedagogical and has no technical consequences.

Definition 1 (MLL$_\mathrm{a}$). *The formulas of MLL$_a$ are built by the grammar*

$$A, B ::= \alpha \mid \alpha^\perp \mid A \otimes B \mid A \parr B \mid \langle a \rangle A \mid \langle \overline{a} \rangle A$$

where the α are literals and the a are CCS names. The negation $A^\perp$ *of a non-literal formula A is defined by de Morgan duality as $(A \otimes B)^\perp = A^\perp \parr B^\perp$ and $(\langle a \rangle A)^\perp = \langle \overline{a} \rangle A^\perp$. A* type *($\Gamma, \Delta \ldots$) is a finite multiset of formulas. Derivations are built from the rules of table 1, where the left side of $\vdash$ is a CCS term up to structural congruence (as of section 3).*

Although it is formulated as a type system for processes, this logic should be interpreted as a calculus for building schedulings. To explain this interpretation, we adopt a few notations that stress the functional aspect of the system: $P : A_1, \ldots, A_n \vdash B$ represents the judgement $P \vdash A_1^\perp, \ldots, A_n^\perp, B$ and the binary connective $A \multimap B$ stands for $A^\perp \parr B$. We easily get the derived rules of table 1: ($\multimap$R) and ($\multimap$L) are respectively a reformulation of ($\parr$) and ($\otimes$), and (mp) is *modus ponens* for linear implication, obtained with (ax), ($\otimes$) and (cut). The (ax) and (cut) rules have natural two-sided counterparts. This is an intuitionistic or implicative formulation, but we do need the full expressiveness of the MLL$_\mathrm{a}$ for the developments of the following sections.

A formula specifies a way for a process to interact with its environment and a proof provides a way to justify this interaction. A judgement $P : A_1, \ldots, A_n \vdash B$

then denotes a function that combines n interactions of types A_i for independent processes Q_i into an interaction of type B of the process $Q_1 \mid \cdots \mid Q_n \mid P$.

- A modality $\langle a\rangle A$ means doing the action a and then acting according to A. To lighten notations, we will represent successive modalities as a single one: $\langle abc\rangle\alpha$ means $\langle a\rangle\langle b\rangle\langle c\rangle\alpha$. Note that the silent action τ is not represented in types, since it is not part of the interactive behaviour of processes.
- Implication $A \multimap B$ is an interaction that provides a behaviour B expecting A from the environment, as made explicit by the rule (mp). The rule ($\multimap$R) means that some context may actually be provided by the environment.
- A variable α is a behaviour not known from the considered term. An interaction of type α means jumping to a continuation of type α, necessarily provided by the context: indeed, since a scheduling of this type may not provide any behaviour, it effectively gives control to some other process.

As we will formalize later on, the term P on the left side of a judgement is guaranteed to be able to provide the behaviour computed by the proof, and this behaviour will consume all the actions of P. Reciprocally, all the behaviours that consume all actions of P correspond to some proof.

Let us illustrate this by examining the possible ways of typing a term like $a.b.1 \mid c.1$. This term has three possible ways of interacting: each interleaving of the sequence (a, b) with the sequence (c) is a valid trace. A simple interleaving is the sequential execution of one part followed by the other, as (a, b, c). E.g.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{1 : C \vdash C}\,(\text{ax})}{b.1 : C \vdash \langle b\rangle C}\,(\text{act})}{a.b.1 : C \vdash \langle ab\rangle C}\,(\text{act}) \qquad \dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\text{ax})}{c.1 : \alpha \vdash \langle c\rangle\alpha}\,(\text{act})}{a.b.1 \mid c.1 : \alpha \vdash \langle abc\rangle\alpha}\,(\text{cut}) \qquad \text{with } C = \langle c\rangle\alpha.$$

The important point is the choice of the axiom on C: it stands for the fact the $a.b.1$ finally hands control to $c.1$ for which we have type C.

The interleaving (a, c, b) is more subtle: now $c.1$ will have to get control from $a.b.1$ after a and give back control to it after doing c. We can write this as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\text{ax})}{b.1 : \alpha \vdash \langle b\rangle\alpha}\,(\text{act}) \qquad \dfrac{}{1 : C \vdash C}\,(\text{ax})}{b.1 : \alpha, \langle b\rangle\alpha \multimap C \vdash C}\,(\multimap\text{L})}{a.b.1 : \alpha, \langle b\rangle\alpha \multimap C \vdash \langle a\rangle C}\,(\text{act})}{a.b.1 : \alpha \vdash (\langle b\rangle\alpha \multimap C) \multimap \langle a\rangle C}\,(\multimap\text{R}) \qquad \dfrac{\dfrac{\dfrac{}{1 : B \vdash B}\,(\text{ax})}{c.1 : B \vdash \langle c\rangle B}\,(\text{act})}{c.1 \vdash B \multimap \langle c\rangle B}\,(\multimap\text{R})}{a.b.1 \mid c.1 : \alpha \vdash \langle acb\rangle\alpha}\,(\text{mp}) \qquad \text{with } \begin{cases} B = \langle b\rangle\alpha \\ C = \langle cb\rangle\alpha \end{cases}$$

Again, the choice of the right types for the axioms is crucial because it depends on the continuation in interaction. Indeed, we have three steps (a, c, b) and as many types for continuations: $\langle cb\rangle\alpha$, $\langle b\rangle\alpha$ and α.

The other crucial point is the introduction of a in front of $b.1$, as the succession of rules (ax), ($\multimap$L), (act), ($\multimap$R). The conclusion type reads as "if using $\langle b\rangle\alpha$ the environment can do C, then, by combining with it, $a.b.1$ can do a then C". Operationally, $a.b.1$ starts by doing a, then jumps to C (the behaviour of the environment), and at some point the environment will give control back from C (that is the negative occurrence of C) and $b.1$ will then perform $\langle b\rangle\alpha$. This part is generic in C: we could use the same reasoning for any type C, including a type variable γ. In a more concise way, $(B \multimap \gamma) \multimap \langle a\rangle\gamma$ is an interruptible version of the modality $\langle a\rangle B$. Similarly, the typing of $c.1$ is generic in B. We only need to choose B and C appropriately for the (mp) rule, so that types unify properly.

Another aspect is when parallel composition is typed by a cut which means that a synchronisation (send/receive) happens between the composed processes:

$$\dfrac{\dfrac{\dfrac{}{1 : \varepsilon \vdash \varepsilon}\,(\text{ax})}{e.1 : \varepsilon \vdash \langle e\rangle\varepsilon}\,(\text{act}) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\text{ax})}{\bar{e}.1 : \langle e\rangle\alpha \vdash \alpha}\,(\text{act})}{d.\bar{e}.1 : \langle e\rangle\alpha \vdash \langle d\rangle\alpha}\,(\text{act}) \quad \dfrac{\dfrac{}{1 : \delta \vdash \delta}\,(\text{ax})}{\bar{d}.1 : \langle d\rangle\delta \vdash \delta}\,(\text{act})}{d.\bar{e}.1 \mid \bar{d}.1 : \langle e\rangle\alpha \vdash \alpha}\,(\text{cut})}{e.1 \mid d.\bar{e}.1 \mid \bar{d}.1 : \alpha \vdash \alpha}\,(\text{cut}) \quad \text{with} \begin{cases} \delta = \alpha \\ \varepsilon = \alpha \end{cases}$$

Here the conclusion type is a simple interaction with the environment. This term has different proofs providing the same type, e.g. using a intermediate trace for $e.1|d.\bar{e}.1$ instead of $d.\bar{e}.1|\bar{d}.1$ as in the proof above. Such variants are irrelevant in scheduling and will be removed by switching to proof nets in the next sections.

3 CCS Runs as Pairings

We consider processes of the standard language CCS [20]. The general language is defined by the following grammar. Note that we use 1 for the inactive process instead of the usual 0 because it is the neutral element of | which is a multiplicative operation. Moreover, actions a are decorated by *locations* ℓ:

$$P, Q ::= a^\ell.P \;\mid\; \bar{a}^\ell.P \;\mid\; 1 \;\mid\; (P \,|\, Q) \;\mid\; P + Q \;\mid\; *P \;\mid\; (\boldsymbol{\nu} a)P$$

where a is taken from an infinite set $\mathcal{N}$ of names and ℓ is taken from an infinite set $\mathcal{L}$ of locations. Each location is used at most once in any term. The main source of non-determinism is the fact that a given action name may occur several times in a given term, and locations are used to name the different occurrences.

For the purpose of the present study, we actually restrict to the following fragment. The reason for this will be explained in the following development.

Definition 2 (MCCS). *Multiplicative CCS is the fragment of CCS using neither choice (+) nor replication (∗). Structural congruence is the smallest congruence $\equiv$ that makes parallel composition associative commutative and* 1 *neutral.*

The set of locations occurring in P is written $\mathcal{L}(P)$. Given $\ell \in \mathcal{L}(P)$, the *subject* of ℓ is the name tagged by ℓ, written $\mathrm{subj}_P\,\ell$. The *polarity* of ℓ is that of the action tagged by its subject, written $\mathrm{pol}_P\,\ell$, element of $\{\pm 1\}$. Intuitively, a negative action $\bar{a}$ represents the sending of a signal on a channel a, and a positive action a represents the reception of such a signal.

Definition 3 (execution). *Execution is the relation over structural congruence classes, labelled by partial involutions over $\mathcal{L}$, defined by the rule*

$$\bar{a}^\ell.P \mid a^m.Q \mid R \quad \to_{ex}^{\{(\ell,m)\}} \quad P \mid Q \mid R$$

Let $\to_{ex*}$ be the reflexive transitive closure of $\to_{ex}$, with the annotations defined as $P \to_{ex*}^{\emptyset} P$ and if $P \to_{ex*}^{c} Q \to_{ex*}^{d} R$ then $P \to_{ex*}^{c \cup d} R$.

The annotation c in $P \to_{ex}^{c} Q$ describes which occurrences interact in the execution step, we write $P \to_{ex} Q$ if c is unimportant. Similarly, we keep locations implicit when they do not matter. Remark that, for a given P and c, there is at most one Q such that $P \to_{ex}^{c} Q$, since c describes the interaction completely.

Definition 4 (pairing). *A* pairing *of a term P is a partial involution c over $\mathcal{L}(P)$ such that for all $\ell \in \operatorname{dom} c$,* $\operatorname{subj} c(\ell) = \operatorname{subj} \ell$ *and* $\operatorname{pol} c(\ell) = -\operatorname{pol} \ell$.

Let $\sim_c$ be the smallest equivalence that contains c. Let $\leq_P$ be the partial order over $\mathcal{L}(P)$ such that $\ell <_P m$ for every subterm $x^\ell.Q$ of P with $m \in \mathcal{L}(Q)$. c is consistent *if* $\operatorname{dom} c$ *is downward closed for $\leq_P$ and $\sim_c <_P \sim_c$ is acyclic.*

Example 1. The total pairings of $P = a^1.c^2 \mid b^3.\bar{a}^4 \mid \bar{b}^5.\bar{c}^6 \mid a^7.\bar{b}^8 \mid b^9 \mid \bar{a}^0$ are $c_1 = \{(9,5),(1,0),(2,6),(3,8),(4,7)\}$, $c_2 = \{(3,5),(1,4),(2,6),(7,0),(9,8)\}$, $c_3 = \{(1,4),(3,8),(7,0),(9,5),(2,6)\}$, $c_4 = \{(1,0),(3,5),(7,4),(9,8),(2,6)\}$. Only c_1 is inconsistent as there is a cycle induced by $\{(3,8),(4,7)\}$. The maximal consistent pairing included in c_1 is $\{(9,5),(1,0),(2,6)\}$.

Observe that pairings and consistency are preserved by structural congruence, as a direct consequence of the fact that subjects, polarities and prefixing are preserved by structural congruence.

Proposition 1. *A pairing c of a term P is consistent if and only if there is a term Q such that $P \to_{ex*}^{c} Q$.*

Proof (sketch). In an execution $P_0 \to_{ex}^{c_1} P_1 \to_{ex}^{c_2} \cdots \to_{ex}^{c_n} P_n$, the c_i are disjoint, so their union is a pairing, and consistency is ensured by the fact that executions respect prefixing. Conversely, write $c = c_1 \uplus \cdots \uplus c_n$ with the c_i atomic. By definition, if c is consistent then $\leq_P$ induces a partial order over the domains of the c_i. Assume that the considered enumeration respects this order, then we can prove by recurrence that there is an execution sequence $P = P_0 \to_{ex}^{c_1} P_1 \cdots \to_{ex}^{c_n} P_n$, since each c_i joins two actions of P_{i-1} that are minimal for $\leq_{P_{i-1}}$.

We easily get the following (for a proof see the appendix [4, B.1]).

Proposition 2. *Let P be a term. Any two executions $P \to_{ex*}^{c} Q$ and $P \to_{ex*}^{c} R$ with the same pairing are permutations of each other, and in this case $Q \equiv R$.*

We will thus consider consistent pairings as the proper notion of execution for CCS terms. Maximal consistent pairings represent executions of processes until a state where no more execution is possible.

A useful tool in the study of pairings is the following notion of determinisation, by which we can turn a pairing of a term into a term that has no other pairing. In other words, determinisation is a way to represent a run of a term in the language of MCCS itself.

Definition 5 (deterministic term). *A term P is* deterministic *if it has at most one occurrence of each action.*

The pairings of a deterministic term form a lattice, consistent pairings too, so there is a unique maximal consistent pairing for any deterministic term.

The restriction operator $(\boldsymbol{\nu}a)$ serves two purposes: it limits the scope of a name, and it makes it possible to have names local to each copy of a subterm in the presence of replication; both these features are useless in the deterministic case, hence we leave it out on determinisation. We abide by Barendregt's convention that each bound channel is named distinctly from each other channel.

Definition 6 (determinisation). *Assume an injective map $\delta : \mathcal{N} \times \{\pm 1\} \times \mathcal{L} \to \mathcal{N}$. Given a partial involution c,* determinisation *along c is the operator ∂_c which commutes with parallel composition such that $\partial_c((\boldsymbol{\nu}a)P) = \partial_c(P)$ and*

$$\partial_c\left(a^\ell.P\right) = \delta(a,+1,\ell)^\ell.\partial_c(P), \quad \partial_c\left(\bar{a}^\ell.P\right) = \begin{cases} \overline{\delta(a,+1,\ell)}^\ell.\partial_c(P) & \text{if } \ell \in \operatorname{dom} c, \\ \delta(a,-1,\ell)^\ell.\partial_c(P) & \text{otherwise.} \end{cases}$$

By construction, $\partial_c(P)$ is deterministic, the pairings of $\partial_c(P)$ are the restrictions of c, consistency preserved, so c is the unique maximal pairing of $\partial_c(P)$.

Example 2. For the term P and the pairings of example 1, we obtain the following determinisations (with $\delta(a,+1,7) = d$ and $\delta(b,+1,9) = e$):
$c_3 = \{(1,4),(3,8),(7,0),(9,5),(2,6)\}$ induces $\partial_{c_3}(P) = a.c \mid b.\bar{a} \mid \bar{e}.\bar{c} \mid d.\bar{b} \mid e \mid \bar{d}$,
$c_4 = \{(1,0),(3,5),(7,4),(9,8),(2,6)\}$ induces $\partial_{c_4}(P) = a.c \mid b.\bar{d} \mid \bar{b}.\bar{c} \mid d.\bar{e} \mid e \mid \bar{a}$.

If we extended our study to the whole of CCS, determinisations would still be in MCCS, but the theory of pairings would have to be refined: external choice requires a notion of conflict in the space of locations (as in event structures [23]), replications requires the introduction of indices to distinguish copies.

4 Proof Nets for MLL with Action Modalities

Proofs in sequent calculus are well suited to inductive reasoning, however their use in proof theory is uneasy because their rigid structure obscures many arguments, like those below in particular. For this reason, we will turn to proof nets, using the standard machinery of linear logic [12,7]. Modality rules are represented using boxes (like promotions in standard linear logic, but with different typing rules). The only extra information we add to standard proof structures is the location of each box, to reflect the use of locations in CCS terms in the sequel. For readers not familiar with the standard definitions for proof nets, these are put in appendix [4, A.1]. We detail here specificities of $\mathrm{MLL_a}$.

Definition 7 (proof structure). *A* proof structure *consists of an ordered forest of nodes labelled by formulas, denoted x^A, with a set Ax of axiom links (pairs of leaves), a set Cut of cuts (pairs of roots) and a set Box of modality boxes, labelled by action modalities, such that each box β has a unique location $\ell(\beta)$. The roots that are not part of a cut are called the* conclusion nodes*. The* conclusion type *is the multiset of the labels of the conclusion nodes.*

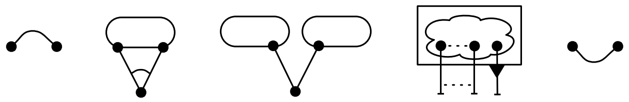


Fig. 1. Representation of proof structures: axiom link, ⅋ node, ⊗ node, boxes, cut

A modality box β is a set of nodes (the ports) associated to a proof structure S whose conclusions are in bijection with the ports. If the modality of β is $\langle a \rangle$, then the *principal port* is labelled $\langle a \rangle A$ and matches a conclusion of S labelled A, while *auxiliary ports* have the same label as their matching conclusion in S.

The graphical notation of proof structures is presented in figure 1. By definition there are arcs only to multiplicative nodes, moreover proof structures can be drawn considering the top-bottom orientation of arcs, so we keep arc orientation implicit by this convention. Arcs to a ⅋ node are joint by a circle on the side of this node. By construction, the conclusion labels suffice to deduce all labels, so we keep most of this information implicit.

Definition 8 (proof net). *A* proof net *is a proof structure built following MLL_a sequent calculus rules. An* immediate subnet *of a proof net π is an induced subgraph of π that is itself a proof net. A* subnet *of π is either an immediate subnet of π or (inductively) a subnet of a box of π.*

Well known correctness criteria [7,12,8] apply to characterise proof nets among proof structures by combinatorial means like acyclicity and connectedness, which allows the definition of proof nets without any reference to sequent calculus. We will not elaborate on this aspect because it is essentially independent from the present work.

Definition 9 (cut elimination). Annotated cut elimination *is the relation $\rightarrow_{ce}^{c}$ over proof structures, labelled by partial involutions c over $\mathcal{L}$, that is the reflexive transitive closure of the rules below (such that if $\pi \rightarrow_{ce}^{c} \pi' \rightarrow_{ce}^{d} \pi''$ then $\pi \rightarrow_{ce}^{c \cup d} \pi''$). We have $\pi \rightarrow_{ce}^{c} \pi'$ if π contains a cut $\kappa = \{x, y\}$ either at top level or inside a box and one of the following cases occurs:*

- *Multiplicative step and Axiom step: standard definition, with $c = \emptyset$.*
- *Modality step: If x and y are principal ports of two boxes β, β', then c permutes $\ell(\beta)$ and $\ell(\beta')$ and π' is obtained by replacing each box with its associated proof structure.*
- *Commutation step: If x is the auxiliary port of a box β, then $c = \emptyset$, and the cut and a subnet of π that contains y are moved inside β.*

Our proof system enjoys a standard cut-elimination theorem using this definition: if $\pi \rightarrow_{ce}^{c} \pi'$ and π is a proof net, then π' is a proof net with the same conclusion (this is proved by standard arguments using correctness criteria, hence we will not develop this point); if a proof π is irreducible by $\rightarrow_{ce}$, then it has no cut link (this is an immediate case analysis). Note however that $\rightarrow_{ce}$ is not confluent, because of commutation steps.

Definition 10 (head reduction). *Head reduction is the annotated relation $\to_h^c$ over proof structures defined as the restriction of $\to_{ce}^c$ that only applies at top level and does not use the commutation step of cut elimination.*

This particular strategy is relevant because it does not reduce inside boxes, that is under prefixes, it only affects cuts in active position (from the point of view of processes). However, this strategy does not eliminate all cuts in general.

In the analysis of proofs, the following notion of path will be useful. It describes a way to traverse arcs and axioms/cuts in a proof structure while respecting the logical meaning of formulas.

Definition 11 (path). *A* path *in a proof structure S is an alternating path in the underlying graph of S, such that alternations occur only at axioms, cuts and boxes. Each move between ports x and y of a box β must be associated with a path between the corresponding conclusions in β. We further require a typing constraint: a path can only move up a left (resp. right) branch if has moved down a left (resp. right) branch before, with a natural well-bracketing condition.*

For instance, a path starting from an axiom with type α may move down the tree of nodes, reach a cut, move up the other side of the cut, always in the branches that contain α, reach an axiom, and so on.

5 Typing Executions of MCCS Terms

Proofs in MLL_a will serve as a type system. Although this can be formulated in usual sequent style (as in table 1), the natural notion rather relates proof nets and structural congruence classes of terms.

Definition 12 (term assignment). *Let S be a proof structure. The MCCS term $\lfloor S \rfloor$ assigned to π is the parallel composition of the $\lfloor \beta \rfloor$ for each box β in S. In turn, for a box β with location ℓ and associated structure S_β, the term $\lfloor \beta \rfloor$ is $a^\ell.\lfloor S_\beta \rfloor$ if the principal port of β has modality $\langle a \rangle$ and $\bar{a}^\ell.\lfloor S_\beta \rfloor$ if the principal port of β has modality $\langle \overline{a} \rangle$. A term P is said to have type Γ if there is a proof net π of conclusion Γ such that $\lfloor \pi \rfloor \equiv P$. In this case we write $\pi : P \vdash \Gamma$.*

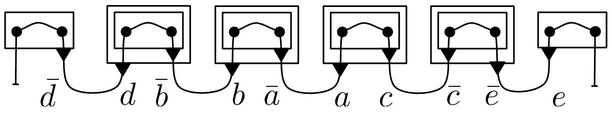
A proof net is a proof structure that is built using the rules of table 1, ignoring the terms on the left of the $\vdash$ symbols. It is obvious that these terms do reflect the definition of term assignment: A term P has type Γ if and only if there is a type derivation with conclusion $P \vdash \Gamma$ using the rules of table 1.

We now establish the correspondence between cut elimination in a proof and execution steps in the assigned terms. The first result justifies head reduction:

Proposition 3. *Let π be a proof structure. For every head reduction $\pi \to_h^c \pi'$ there is an execution $\lfloor \pi \rfloor \to_{ex*}^c \lfloor \pi' \rfloor$.*

Proof (sketch). Axiom and multiplicative cut elimination steps do not affect the assigned terms, besides their annotation is empty, so the result holds immediately for them. When a modality step applies, it reduces a cut between boxes with dual modalities (because of typing), hence the associated terms are ready to interact; the reduct is easily seen to be the assigned term of the reduct proof.

Example 3. Let π be the following proof net.



We have $\lfloor \pi \rfloor = a.c \mid b.\bar{a} \mid \bar{e}.\bar{c} \mid d.\bar{b} \mid e \mid \bar{d}$. (It is $\partial_{c_3}(P)$ of previous examples). As it is a deterministic term, we abusively identify locations with names. We consider the head reduction sequence $\pi \to_h^z \pi'$ (where π' is an axiom link) for $z = \{(d,\bar{d}), (b,\bar{b}), (a,\bar{a}), (e,\bar{e}), (c,\bar{c})\}$. We have $\lfloor \pi \rfloor \to_{ex*}^z \lfloor \pi' \rfloor \equiv 1$.

Subject reduction does not hold in general. Indeed, a given proof may hold several occurrences of a given modality, corresponding to different occurrences of an action in the term, and the structure of cuts may not match a given execution step. This is not a defect, since we actually intend to type pairings rather than processes: we do get subject reduction if we restrict to proofs that describe deterministic terms.

Definition 13 (linear proof). *A proof structure S is called* linear *if*

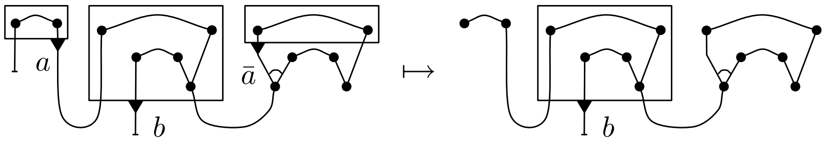
- *S contains at most one box for each modality,*
- *for each a, all occurrences of $\langle \overline{a} \rangle A$ in the labels in S have the same immediate subformula A, and if $\langle \overline{a} \rangle A$ and $\langle a \rangle B$ occur then A and B are dual,*
- *if S contains a box for both $\langle a \rangle A$ and $\langle \overline{a} \rangle A^\perp$, then neither formula occurs in the conclusion type of S.*

The essence of the linearity condition is the first constraint. Intuitively, the second and third constraints serve to guarantee that the property is preserved by composition. Indeed, if a formula $\langle a \rangle A$ occurs in the conclusion of a proof π, then the proof may be cut against a proof that contains a modality box for $\langle \overline{a} \rangle A^\perp$, which breaks linearity if π already contains a box for some $\langle \overline{a} \rangle B$. Note that the fact of being a linear proof is preserved by cut elimination.

Theorem 1 (subject reduction). *Let π be a linear proof of conclusion $P \vdash \Gamma$. For every execution $P \to_{ex*}^c P'$ there is a linear proof $\pi' : P' \vdash \Gamma$.*

Proof (sketch). An execution step $\lfloor \pi \rfloor \to_{ex}^{(\ell,m)} P$ involves immediate subterms $a^\ell.Q$ and $\bar{a}^m.R$ for $a \in \mathcal{N}$. Then π contains two top level boxes with respective principal ports $x^{\langle a \rangle A}$ and $y^{\langle \overline{a} \rangle A^\perp}$, for $A \in MLL_{\mathrm{a}}$. Since π is linear, x and y are elimination boxes for each other, ending a path ρ (as of definition 11) whose axioms contain modalities of x and y in their types. Let π' be the rewriting of π where such modalities are removed (boxes are replaced by their contents, axioms on $\langle a \rangle A$ by axioms on A). Clearly π' is a linear proof of conclusion $P' \vdash \Gamma$.

This theorem states that types are preserved by execution in deterministic terms. However, the proof uses a rewriting of the typing proofs that does not correspond to cut elimination in general. Indeed, consider the following example of typing, call π the l.h.s.:



Then the proof is linear, irreducible by head cut elimination, but the assigned term $\lfloor\pi\rfloor = \bar{a} \mid \bar{b} \mid a$ does execute into $\bar{b}$. In π, this involves a cut on the axiom inside the middle box. As done in theorem 1 the rewriting of π in a linear proof π' assigned to $\bar{b}$ is the r.h.s..

We can get a precise correspondence between execution and head cut elimination by imposing an additional constraint on the shape of proofs. In the statement below, an axiom is *immediately contained* in a box if it is an immediate subnet of the structure associated with this box.

Definition 14 (regular proof). *An axiom link immediately contained in a box β is* anchored *if there is a path from one of its conclusions to an auxiliary port of β and a path from its other conclusion to the principal port. A proof structure π is* regular *if all its axioms are anchored and for every pair of boxes with dual modalities, one of the boxes does not immediately contain any axiom.*

Theorem 2 (strong subject reduction). *Let π be a regular linear proof net. For every execution $\lfloor\pi\rfloor \rightarrow^{c}_{ex*} P$ there is a regular linear proof π' such that $\pi \rightarrow^{c}_{h} \pi'$ and $\lfloor\pi'\rfloor = P$.*

Proof (sketch). Consider an execution step $\lfloor\pi\rfloor \rightarrow^{(\ell,m)}_{ex} P$. As in the proof of theorem 1, linearity implies that there are boxes at top level and a path ρ between their principal ports $x^{\langle a\rangle A}$ and $y^{\langle\overline{a}\rangle A^{\perp}}$ for immediate subterms $a^{\ell}.Q$ and $\bar{a}^{m}.R$ of $\lfloor\pi\rfloor$. Since x is cut at top level, ρ traverses no box, otherwise linearity or regularity would be contradicted. Then ρ is a multiplicative cut path whose cut elimination $\rightarrow^{\emptyset}_{h}$ until x and y preserves $\lfloor\pi\rfloor$ as well as regularity and linearity.

6 Anti-execution and Completeness

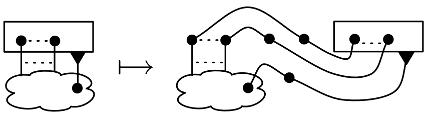
In this section we establish our correspondence theorem relating typings and executions. To achieve this goal we first provide a kind of reciprocal statement for subject reduction: if a term T can reduce into a typed term T', then we can type T with a proof that reduces to the typing of T'. Because we want logically correct proof structures, this operation requires some care.

Example 4. Consider the term $P := a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c$. We cannot type each thread with a simple type like $\langle a\rangle\alpha$, $\langle\bar{b}\rangle\alpha^{\perp}$ and then introduce a cut for each interaction, since we would get a cyclic proof structure, which is incorrect.

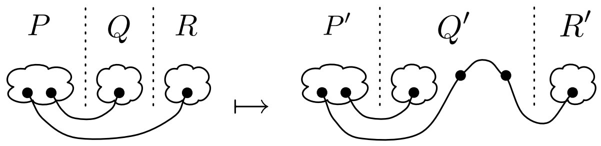
We now describe a general method for deducing a typing by "anti-execution" of a proof. We stay at a partly informal level for clarity, all formal statements are detailed in the appendix [4, B.3].

Consider a generic execution step $P \mid a.Q \mid \bar{a}.R \rightarrow_{ex} P \mid Q \mid R$. Assume the reduct is typed by some proof π. We want to put the parts of π that correspond to Q and R into boxes, with a cut between them, while rewriting the proof to avoid cycles. For this purpose, we proceed in four steps:

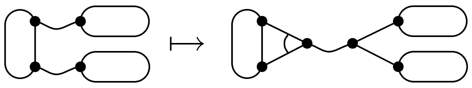
Selection consists in moving each box belonging to Q or R away from the main proof, by means of an axiom/cut pair, so that Q and R are represented by simple sets of boxes, cut with the main proof (which corresponds to P), with no multiplicative connectives:



Chaining consists in introducing an extra axiom/cut pair in the middle of each cut between P and R, so that there are cuts only between P and Q or Q and R, and not between P and R directly:

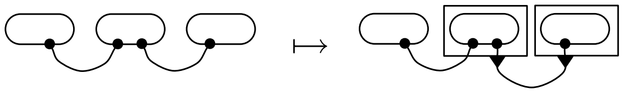


Simplification consists in making sure that there is actually exactly one cut between P and Q and one between Q and R, by multiplexing multiple cuts through multiplicatives:



Correctness criteria guarantee that we can always find two cuts for which there is one connected component on one side, two on the other.

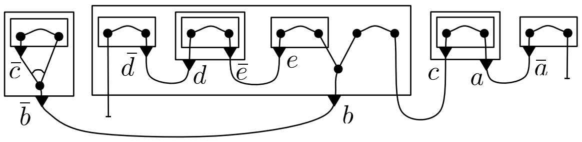
Boxing consists in putting Q and R into boxes, cut together, so that Q has one auxiliary port to P and R has no auxiliary port:



Following this method, we prove the following statement:

Proposition 4 (anti-execution). *Let $T_1 \rightarrow^c_{ex} T_2$ be an execution step and let $\pi_2 : T_2 \vdash \Gamma$ be a typing. There exists a typing $\pi_1 : T_1 \vdash \Gamma$ such that $\pi_1 \rightarrow^c_h \pi_2$.*

Example 5. Consider the term of P of example 1. We consider the execution $e = (a, \bar{a})(b, \bar{b})(c, \bar{c})(d, \bar{d})(e, \bar{e})$ of the determinized term $\partial_{c_4}(P) = a.c|b.\bar{d}|\bar{b}.\bar{c}|d.\bar{e}|e|\bar{a}$ for the (total and consistent) pairing $c_4 = \{(1, 0), (3, 5), (7, 4), (9, 8), (2, 6)\}$. A typing synthesized by the construction of proposition 4 is the following.



Lemma 1 (preserved regularity). *In the construction of proposition 4, if π_2 is regular, then so is π_1. If π_2 is linear and T_2 is deterministic, then π_1 is linear.*

Proof (sketch). Let $T_2 = P \mid Q \mid R$. If an axiom is introduced by anti-execution rewrite steps, used in proposition 4 then: i) it is added to P by selection and it will not be boxed, or ii) it is added to Q by chaining and becomes anchored by simplification and boxing. No axiom is introduced on the side of R, Q only contains chaining axioms, so regularity is satisfied for the new axioms. Besides, regularity is not broken for axioms previously present in the proof.

Example 6. In the previous example 5, one can also start execution by $(b, \bar{b})(a, \bar{a})$ as seen in the typing. All execution permutation of $\partial_{c_4}(P)$ in the pairing c_4 is allowed by the typing proof synthesized from the execution e.

We now summarize the previous results, about subject reduction and the reverse operation, into a precise statement relating typings and execution.

Lemma 2 (initial typing). *Every linear MCCS term where no name occurs with both modalities is typable by a cut-free regular proof.*

Proof. We simply build a proof of $T \vdash A_T, B_T$ with A_T non-modal by induction on T. For $T = 1$, use the axiom rule to get $1 \vdash \alpha^\perp, \alpha$. For $T = P \mid Q$, deduce $T \vdash A_P \parr A_Q, B_P \otimes B_Q$ by the tensor rule. For $T = a.P$, deduce $T \vdash A_P, \langle a \rangle B_P$ by the action rule, similarly for $\bar{a}.P$. The proof thus built is obviously regular since every axiom is at top level or anchored, and there are no pairs of boxes with dual modalities.

Theorem 3 (completeness). *For every execution $P \rightarrow^c_{ex*} Q$ there are typings $\pi_P : P \vdash \Gamma$ and $\pi_Q : Q \vdash \Gamma$ such that $\pi_P \rightarrow^c_h \pi_Q$. Moreover, for every execution sequence $P \rightarrow^{c_1}_{ex} P_1 \cdots \rightarrow^{c_n}_{ex} P_n = Q$ with $c_1 \cup \cdots \cup c_n = c$, there is a cut elimination sequence $\pi_P \rightarrow^{c_1}_h \pi_1 \cdots \rightarrow^{c_n}_h \pi_n = \pi_Q$, with $\lfloor \pi_i \rfloor = P_i$ for all i.*

Proof. By definition, the term $\partial_c(Q)$ is linear and has no dual actions, so by lemma 2 we can find a cut-free regular proof $\pi'_Q : \partial_c(Q) \vdash \Gamma$. If we apply proposition 4 repeatedly to π'_Q with the steps of the considered execution $\partial_c(P) \rightarrow^c_{ex*} \partial_c(Q)$, we get a proof $\pi'_P : \partial_c(P) \vdash \Gamma$ that reduces to π'_Q by a head reduction sequence labelled c. Let π_P and π_Q be the relabellings of π'_P and π'_Q by the inverse of ∂_c, then we have $\pi_P : P \vdash \Gamma$, $\pi_Q : Q \vdash \Gamma$ and $\pi_P \rightarrow^c_h \pi_Q$.

Every execution sequence of P with label c is an execution sequence of $\partial_c(P)$ with the same label. By lemma 1, π'_P enjoys strong subject reduction as of theorem 2, hence every run of $\partial_c(P)$ labelled by c corresponds to a head reduction sequence in π'_P labelled by c. By relabelling with ∂_c^{-1}, every run of P labelled by c corresponds to a head reduction sequence $\pi_P \rightarrow^c_h \pi_Q$.

In other words, every execution of a term can be exactly characterized up to permutation by typing, in the sense that the execution sequences of the term within the same pairing will be exactly the head reduction sequences of the associated typing proof. By combining determinisation (definition 6) and strong subject reduction (theorem 2) we get that, conversely, each regular typing of a term defines a set of executions stable by permutation.

7 Conclusion and Further Works

In this work we have developed, in the simple framework of multiplicative CCS, a precise logical description of executions of processes. A key technical tool is the use of pairings, by which we separate non-determinism in communication from the multiplicity of equivalent schedulings; this technique extends well to more expressive frameworks (full CCS, π-calculus, etc.). The logical interpretation we propose moves beyond the traditional Curry-Howard for concurrency by accepting non-deterministic terms, albeit with a change of interpretation in the correspondence. Indeed, the logic we use is well studied and has a wide range of existing tools (efficient correctness criteria, proof search, etc.) but its interpretation in our paradigm of proof-as-executions is new.

Logical Expressiveness. The restriction to purely multiplicative objects, in MCCS and MLL, lets us concentrate on the precise role of multiplicatives and axioms as descriptions of how a process interacts with its environment but hides the complexity inherent to the other defining features of concurrent systems like choice, recursion, name passing, etc. It should be stressed that extending the calculus or the logic are two different things. Extending the calculus enriches the set of possible executions, by introducing more subtle synchronization possibilities: choice allows for conflict between actions, replication allows for arbitrarily large runs with some uniformity, value passing allows for communication of ground values, name passing allows the set of synchronizable pairs to evolve along execution. After determinisation, all these features essentially disappear and deterministic runs can still be formulated in MCCS. On the other hand, enriching the logic leads to richer descriptions of the control flow in processes, for instance using a first order language with predicates to describe properties of continuations.

Causality. A crucial feature of our work is the interpretation of axioms as a way to transfer causality. This idea suggests new ways of analyzing causality in interactive systems, and the fact that the flow of causality is often as complicated as the flow of information. Besides, a similar fact is illustrated by the expressiveness of solos [15,3], where communication is used to carry all prefixing information in processes. Our interpretation may provide a logical insight on this matter.

Cut Elimination. In the present work, as in proofs-as-programs formalisms, composition of processes is represented by the cut rule and execution corresponds to a particular cut elimination strategy. An interesting direction for future work is the study of the meaning of full cut-elimination, from the proofs-as-executions point of view. The operationally relevant part is the elimination of dual actions, which means executing all internal transitions in advance. This implies making choices with respect to synchronisation. In other words, eliminating cuts in a MLL_a proof yields a more deterministic process that can still exhibit the behaviour given by the considered type.

References

1. Abramsky, S.: Proofs as processes. TCS 135(1), 5–9 (1994)
2. Beffara, E.: A concurrent model for linear logic. ENTCS 155, 147–168 (2006)
3. Beffara, E., Maurel, F.: Concurrent nets: a study of prefixing in process calculi. TCS 356(3), 356–373 (2006)
4. Beffara, E., Mogbil, V.: Proofs as executions. Technical Report 00586459, HAL (July 2012), http://hal.archives-ouvertes.fr/hal-00586459
5. Bruscoli, P.: A Purely Logical Account of Sequentiality in Proof Search. In: Stuckey, P.J. (ed.) ICLP 2002. LNCS, vol. 2401, pp. 302–316. Springer, Heidelberg (2002)
6. Caires, L., Pfenning, F.: Session Types as Intuitionistic Linear Propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010)
7. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Math. Logic 28(3), 181–203 (1989)
8. Jacobé de Naurois, P., Mogbil, V.: Correctness of linear logic proof structures is NL-complete. TCS 412(20), 1941–1957 (2011)
9. Ehrhard, T., Laurent, O.: Interpreting a Finitary π-calculus in Differential Interaction Nets. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 333–348. Springer, Heidelberg (2007)
10. Ehrhard, T., Regnier, L.: Differential interaction nets. TCS 364(2), 166–195 (2006)
11. Girard, J.-Y.: Linear logic. TCS 50(1), 1–102 (1987)
12. Girard, J.-Y.: Proof-nets: the parallel syntax for proof theory. Logic and Algebra 180 (1996)
13. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32(1), 137–161 (1985)
14. Honda, K., Laurent, O.: An exact correspondence between a typed π-calculus and polarised proof-nets. TCS 411(22–24), 2223–2238 (2010)
15. Laneve, C., Victor, B.: Solos in Concert. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 513–523. Springer, Heidelberg (1999)
16. Mairson, H.G., Terui, K.: On the Computational Complexity of Cut-Elimination in Linear Logic. In: Blundo, C., Laneve, C. (eds.) ICTCS 2003. LNCS, vol. 2841, pp. 23–36. Springer, Heidelberg (2003)
17. Maurel, F.: Nondeterministic Light Logics and NP-Time. In: Hofmann, M.O. (ed.) TLCA 2003. LNCS, vol. 2701, pp. 241–255. Springer, Heidelberg (2003)
18. Mazurak, K., Zdancewic, S.: Lolliproc: to concurrency from classical linear logic via curry-howard and control. In: ICFP, pp. 39–50 (2010)
19. Miller, D.: The π-Calculus as a Theory in Linear Logic: Preliminary Results. In: Lamma, E., Mello, P. (eds.) ELP 1992. LNCS, vol. 660, pp. 242–264. Springer, Heidelberg (1993)
20. Milner, R.: Communication and concurrency. Prentice Hall (1989)
21. Mogbil, V.: Non-deterministic Boolean Proof Nets. In: van Eekelen, M., Shkaravska, O. (eds.) FOPARA 2009. LNCS, vol. 6324, pp. 131–145. Springer, Heidelberg (2010)
22. Tiu, A., Miller, D.: A proof search specification of the π-calculus. ENTCS 138(1), 79–101 (2005)
23. Winskel, G.: Event structures. In: Advances in Petri nets: Applications and Relationships to Other Models of Concurrency, pp. 325–392. Springer, Heidelberg (1987)
24. Yoshida, N., Berger, M., Honda, K.: Strong normalisation in the π-calculus. In: LICS, pp. 311–322 (2001)

Efficient Algorithms for the MAX k-VERTEX COVER Problem[*]

Federico Della Croce[1] and Vangelis Th. Paschos[2,3]

[1] D.A.I., Politecnico di Torino, Italy
federico.dellacroce@polito.it
[2] PSL Research University, Université Paris-Dauphine, LAMSADE, CNRS, UMR 7243, France
paschos@lamsade.dauphine.fr
[3] Institut Universitaire de France

Abstract. We first devise moderately exponential exact algorithms for MAX k-VERTEX COVER, with time-complexity exponential in n but with polynomial space-complexity by developing a branch and reduce method based upon the measure-and-conquer technique. We then prove that, there exists an exact algorithm for MAX k-VERTEX COVER with complexity bounded above by the maximum among c^k and γ^τ, for some $\gamma < 2$, where τ is the cardinality of a minimum vertex cover of G (note that MAX k-VERTEX COVER $\notin$ **FPT** with respect to parameter k unless **FPT** = **W[1]**), using polynomial space. We finally study approximation of MAX k-VERTEX COVER by moderately exponential algorithms. The general goal of the issue of moderately exponential approximation is to catch-up on polynomial inapproximability, by providing algorithms achieving, with worst-case running times importantly smaller than those needed for exact computation, approximation ratios unachievable in polynomial time.

1 Introduction

In the MAX k-VERTEX COVER problem a graph $G(V, E)$ with $|V| = n$ vertices $1, \ldots, n$ and $|E|$ edges (i, j) is given together with an integer value $k < n$. The goal is to find a subset $K \subset V$ with cardinality k, that is $|K| = k$, such that the total number of edges covered by K is maximized. In its decision version, MAX k-VERTEX COVER can be defined as follows: "given G, k and ℓ, does G contain k vertices that cover at least ℓ edges?". MAX k-VERTEX COVER is **NP**-hard (it contains the minimum vertex cover problem as particular case), but it is polynomially approximable within approximation ratio 3/4, while it cannot be solved by a polynomial time approximation schema unless **P** = **NP**. The interested reader can be referred to [19,30] for more information about approximation issues for this problem.

[*] Research supported by the French Agency for Research under the program TODO, ANR-09-EMER-010 and by a Lagrange fellowship of the Fondazione CRT, Torino, Italy.

In the literature, we often find this problem under the name PARTIAL VERTEX COVER problem. It is mainly studied from a parameterized complexity point of view (see [17] for information on fixed-parameter (in)tractability). A problem is fixed-parameter tractable with respect to a parameter t, if it can be solved (to optimality) with time-complexity $O(f(t)p(n))$ where f is a function that depends on the parameter t, and p is a polynomial on the size n of the instance. In what follows, when dealing with fixed parameter tractability of MAX k-VERTEX COVER, we shall use notation MAX k-VERTEX COVER(t) to denote that we speak about fixed parameter tractability with respect to parameter t. Parameterized complexity issues for MAX k-VERTEX COVER are first studied in [3] where it is proved that PARTIAL VERTEX COVER is fixed-parameter tractable with respect to parameter ℓ, next in [28] where it is proved that it is **W[1]**-hard with respect to parameter k (another proof of the same result can be found in [9]) and finally in [31] where the fixed-parameter tractability results of [3] are further improved. Let us also quote the paper by [24], where it is proved that in apex-minor-free graphs graphs, PARTIAL VERTEX COVER can be solved with complexity that is subexponential in k.

The seminal Courcelle's Theorem [13] (see also [21,20] as well as [37] for a comprehensive study around this theorem) assures that decision problems defined on graphs that are expressible in terms of monadic second-order logic formulæ are fixed parameter tractable when the treewidth[1] of the the input-graph G, denoted by w, is used as parameter. Courcelle's Theorem can be also extended to a broad class of optimization problems [1]. As MAX k-VERTEX COVER belongs to this class, it is fixed parameter tractable with respect to w. In most of cases, "rough" application of this theorem, involves very large functions $f(w)$ (see definition of fixed-parameter tractability given above).

In [34], it is proved that *given a nice tree decomposition, there exists a fixed-parameter algorithm (based upon dynamic programming), with respect to parameter w that solves* MAX k-VERTEX COVER *in time $O(2^w k(w^2 + k) \cdot |I|)$, where $|I|$ is the number of nodes of the nice tree decomposition and in exponential space.* In other words, MAX k-VERTEX COVER$(w) \in$ **FPT**, but the fixed-parameter algorithm of [34] uses exponential space. Let us note that in any graph G, denoting by τ the size of a minimum vertex cover of G, it holds that $w \leqslant \tau$. So, MAX k-VERTEX COVER$(\tau) \in$ **FPT** too, but through the use of exponential space (recall that, as adopted above, MAX k-VERTEX COVER(τ) denotes the MAX k-VERTEX COVER problem parameterized by the size τ of a minimum vertex cover.

Very frequently, a serious problem about fixed-parameter tractability with respect to w is that it takes too much time to compute the "nice tree decomposition" that also derives the value of w. More precisely, this takes time $O^*(1.7549^n)$

[1] A tree decomposition of a graph $G(V, E)$ is a pair (X, T) where T is a tree on vertex set $V(T)$ the vertices of which we call nodes and $X = (\{X_i : i \in V(T)\})$ is a collection of subsets of V such that: (i) $\cup_{i \in V(T)} X_i = V$, (ii) for each edge $(v, w) \in E$, there exist an $i \in V(T)$ such that $\{v, w\} \in X_i$, and (iii) for each $v \in V$, the set of nodes $\{i : v \in Xi\}$ forms a subtree of T. The width of a tree decomposition $(\{X_i : i \in V(T)\}, T)$ equals $\max_{i \in V(T)}\{|Xi| - 1\}$. The treewidth of a graph G is the minimum width over all tree decompositions of G.

(notation $O^*(\cdot)$ ignores polynomial factors) by making use of exponential space and time $O^*(2.6151^n)$ by making use of polynomial space [25]. Note that the problem of deciding if the treewidth of a graph is at most w is fixed-parameter tractable and takes time $O(2^{O(w^3)}n)$ [33].

Dealing with solution of MAX k-VERTEX COVER by exact algorithms with running times (exponential) functions of n, let us note that a trivial optimal algorithm for MAX k-VERTEX COVER takes time $O^*(\binom{n}{k}) = O^*(n^k)$, and polynomial space, producing all the subsets of V of size k. This turns to a worst-case $O^*(2^n)$ time (since $\binom{n}{k} \leqslant 2^n$ with equality for $k = \frac{n}{2}$). An improvement of this bound is presented in [9], where an exact algorithm with complexity $O^*(n^{\omega\lceil k/3\rceil + O(1)})$ was proposed based upon a generalization of the $O^*(n^{\omega t})$ algorithm of [35] for finding a $3t$-clique in a graph, where $\omega = 2.376$. This induces a complexity $O^*(n^{0.792k})$, but exponential space is needed. As far as we know, no exact algorithm with running time $O^*(\gamma^n)$, for some $\gamma < 2$, is known for MAX k-VERTEX COVER.

In this paper, we first devise an exact branch and reduce algorithm based upon the measure-and-conquer paradigm by [22] (Section 2) requiring running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$, where Δ denotes the maximum degree of G, and polynomial space. The algorithm is then tailored to graphs with maximum degree 3 inducing a running time $O^*(1.3339^n)$ (Section 4). In Section 3, we devise a fixed parameter algorithm, with respect to parameter τ where, as mentioned above, τ is the cardinality of a minimum vertex cover of G that works in time $O^*(2^\tau)$ and needs only polynomial space. By elaborating a bit more this result we then show that the time-complexity of this algorithm is indeed either $O^*(\gamma^\tau)$ for some $\gamma < 2$ or $O^*(c^k)$, for some $c > 2$. In other words, this algorithm either works in time better than 2^τ or it is fixed parameter with respect to the size k of the desired cover. Finally, we show that the technique used for proving that MAX k-VERTEX COVER$(\tau) \in$ **FPT**, can be used to prove inclusion in the same class of many other well-known combinatorial problems. A corollary of the inclusion of MAX k-VERTEX COVER(τ) in **FPT**, is that MAX k-VERTEX COVER in bipartite graphs can be solved in time $O^*(2^{n/2}) \simeq O^*(1.414^n)$. Finally, in Section 5, we address the question of approximating MAX k-VERTEX COVER within ratios "prohibited" for polynomial time algorithms, by algorithms running with moderately exponential complexity. The general goal of this issue is to cope with polynomial inapproximability, by developing algorithms achieving, with worst-case running times significantly lower than those needed for exact computation, approximation ratios unachievable in polynomial time. This approach has already been considered for several other paradigmatic problems such as MINIMUM SET COVER [7,15], MIN COLORING [2,6], MAX INDEPENDENT SET and MIN VERTEX COVER [5], MIN BANDWIDTH [16,26], ... Similar issues arise in the field of FPT algorithms, where approximation notions have been introduced, for instance, in [10,18]. In this framework, we particularly quote [32] where it is proved that, although not in **FPT**, MAX k-VERTEX COVER(k) is approximable by an FPT (with respect to k) approximation schema, where function $f(k)$ (in the time-complexity of this schema) is quite large, i.e., around something like $O^*(k^{2k^2})$.

2 An $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$-Time Polynomial Space Algorithm in General Graphs

In what follows, we denote by α_j the total number of vertices adjacent to j that have been discarded in the previous levels of the search tree. We denote by d_j the degree of vertex j and by $N(j)$ the set of vertices adjacent to j, that is the neighborhood of j. Notice that, whenever a branch on a vertex j occurs, for each $l \in N(j)$, if j is selected then d_l is decreased by one unit as edge (j, l) is already covered by j. Alternatively, j is discarded: correspondingly d_l is not modified and α_l is increased by one unit. We propose in this section a branch and reduce approach based on the measure-and-conquer paradigm (see for instance [22]). Consider a classical binary branching scheme on some vertex j where j is either selected or discarded. Contrarily to the classical branch-and-reduce paradigm where for each level of the search tree we define as *fixed* those vertices that have already been selected or discarded, while we define as *free* the other vertices, when using measure-and-conquer, we do not count in the measure the fixed vertices, namely the vertices that have been either selected or discarded at an earlier stage of the search tree and we count with a weight w_h the free vertices h. The vertex j to be selected is the one with largest coefficient $c_j = d_j - \alpha_j$. Let $c_{\max}$ denote such a coefficient, hence $c_{\max} \leqslant \Delta$. Then, each free vertex h is assigned a weight $w_h = w_{[i]}$ with $i = c_i = d_h - \alpha_h$ and we impose $w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots \leqslant w_{[c_{\max}]} = 1$ that is the weights of the vertices are strictly increasing in their c_j coefficients.

We so get recurrences on the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of its vertices. Since initially $p = n$, the overall running time is expressed as a function of n. This is valid since when $p = 0$, there are only vertices with weight $w_{[0]}$ in the graph and, in this case, the problem is immediately solved by selecting the $k - \gamma$ vertices with largest α_j (if $\gamma < k$ vertices have been selected so far). Correspondingly free vertices j with no adjacent free vertices receive weight $w_{[0]} = 0$.

We claim that MAX k-VERTEX COVER can be solved with running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ by the following algorithm called `MAXKVC`:

> Select j such that c_j is maximum and branch according to the following exhaustive cases:
> 1. if $c_j \geqslant 3$, then branch on j and either select or discard j;
> 2. else, $c_j \leqslant 2$ and `MAXKVC` is polynomially solvable.

Theorem 1. *Algorithm `MAXKVC` solves* MAX k-VERTEX COVER *with running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ using polynomial space.*

Proof. To prove the above statement, we first show that the branch in step 1 can be solved with complexity $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ and then we show that step 2 is polynomially solvable. Consider step 1. We always branch on the vertex j with largest $c_j = c_{\max} \leqslant \Delta$ where $c_j \geqslant 3$ and either we select or discard j. If we select j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their degree (and correspondingly their coefficient) by one unit. Similarly, if we discard j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their coefficient as their degree remains unchanged but their α parameter is increased by one unit. Hence, the recurrence becomes:

$$T(p) \leqslant 2T\left(p - w_{[c_{\max}]} - \sum_{h \in N(j)} \left(w_{[c_h]} - w_{[c_h - 1]}\right)\right)$$

By constraining the weights to satisfy the inequality:

$$w_{[j]} - w_{[j-1]} \leqslant w_{[j-1]} - w_{[j-2]} \quad \forall j = 2, \ldots, c_{\max}$$

the previous recurrence becomes in the worst-case:

$$T(p) \leqslant 2T\left(p - w_{[c_{\max}]} - c_{\max}\left(w_{[c_{\max}]} - w_{[c_{\max}-1]}\right)\right)$$

As $c_{\max} \leqslant \Delta$, where the equality occurs when $\alpha_j = 0$, the above recurrence becomes, in the worst-case, $T(p) \leqslant 2T\left(p - w_{[\Delta]} - \Delta\left(w_{[\Delta]} - w_{[\Delta-1]}\right)\right)$.

Summarizing, to handle graphs with maximum degree Δ, we need to guarantee that the recurrences $T(p) \leqslant 2T(p - w_{[i]} - i(w_{[i]} - w_{[i-1]}))$, $\forall i \in 3, \ldots, \Delta$ (as $c_j \geqslant 3$), and the constraints:

$$\begin{array}{l} w_{[i]} - w_{[i-1]} \leqslant w_{[i-1]} - w_{[i-2]} \;\; \forall i = 2, \ldots, \Delta \\ 0 = w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots \leqslant w_{[\Delta-1]} \leqslant w_{[\Delta]} = 1 \end{array}$$

are satisfied simultaneously. This corresponds to a non linear optimization problem of the form:

$$\begin{aligned} \min\ & \alpha \\ & \alpha^{(w_{[i]} + i(w_{[i]} - w_{[i-1]}))} \geqslant 2 \;\; \forall i = 3, \ldots, \Delta & (1) \\ & w_{[i]} - w_{[i-1]} \leqslant w_{[i-1]} - w_{[i-2]} \;\; \forall i = 2, \ldots, \Delta & (2) \\ & 0 = w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots \leqslant w_{[\Delta-1]} \leqslant w_{[\Delta]} = 1 & (3) \end{aligned}$$

We so get performances 1.4142^n, for $\Delta = 3$, 1.5157^n, for $\Delta = 4$, 1.5866^n, for $\Delta = 5$, 1.6405^n, for $\Delta = 6$, 1.6817^n, for $\Delta = 7$, or 1.7143^n, for $\Delta = 8$.

Interestingly enough, for all these values of Δ, the complexity corresponds to $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$. Indeed, this is not accidental. By setting:

$$\begin{aligned} w_{[i]} &= \frac{(i-1)(\Delta+1)}{(i+1)(\Delta-1)} \;\; \forall i = 2, \ldots, \Delta & (4) \\ w_{[1]} &= \frac{1}{2} w_{[2]} & (5) \\ w_{[0]} &= 0 & (6) \end{aligned}$$

we can see that constraints (2) and (3) are satisfied. To see that inequalities (2) are satisfied, notice that:

$$\begin{aligned} w_{[3]} - w_{[2]} = w_{[2]} - w_{[1]} &= \frac{1}{3} w_{[3]} \\ w_{[2]} - w_{[1]} = w_{[1]} - w_{[0]} &= w_{[1]} \end{aligned}$$

For the general recursion with $i \geqslant 4$, we have to show that $w_{[i]} - w_{[i-1]} \leqslant w_{[i-1]} - w_{[i-2]}$, i.e., that $w_{[i]} - 2w_{[i-1]} + w_{[i-2]} \leqslant 0$. This corresponds to:

$$\begin{aligned}
&\left(\frac{i-1}{i+1} - 2\frac{i-2}{i} + \frac{i-3}{i-1}\right)\left(\frac{\Delta+1}{\Delta-1}\right) \leqslant 0 \\
\Longrightarrow\ & \frac{i-1}{i+1} - 2\frac{i-2}{i} + \frac{i-3}{i-1} \leqslant 0 \\
\Longleftrightarrow\ & i(i-1)^2 - 2(i-2)\left(i^2-1\right) + i(i-3)i + 1 \leqslant 0 \\
\Longleftrightarrow\ & i^3 - 2i^2 + i - 2i^3 + 4i^2 + 2i - 4 + i^3 - 2i^2 - 3i = -4 \leqslant 0, \quad \forall i
\end{aligned}$$

Also, to see that inequalities (3) are satisfied, notice that equations (4) imply:

$$\begin{aligned}
w_{[\Delta]} &= 1 \\
w_{[i]} &> 0 \ \ \forall i = 2, \ldots, \Delta \\
w_{[i]} &> w_{[i-1]} \ \ \forall i = 3, \ldots, \Delta
\end{aligned}$$

while equations (5) and (6) imply $w_{[2]} > w_{[1]} > w_{[0]} = 0$.

Finally, notice that such values of $w_{[j]}s$ satisfy constraints (1) that now correspond to $\Delta - 2$ copies of the inequality $\alpha^{\frac{\Delta+1}{\Delta-1}} \geqslant 2$ where the minimum value of α is obviously given by $2^{\frac{\Delta-1}{\Delta+1}n}$. Consequently, the overall complexity of step 1 is $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

We consider now step 2. For $c_j = c_{\max} \leqslant 2$, MAX k-VERTEX COVER can be seen as a maximum weighted k-vertex cover problem in an undirected graph G where each vertex j has a weight α_j and a degree $d_j = c_j$ and the maximum vertex degree is 2. But this problem has been shown to be solvable in $O(n)$ time by dynamic programming in [36]. ♦

3 MAX k-VERTEX COVER and Fixed-Parameter Tractability

Denote by $(a - \overline{b} - c)$, a branch of the search tree where vertices a and c are selected and vertex b is discarded. Consider the vertex j with maximum degree Δ and neighbors $l_1, \ldots, l_\Delta$. As j has maximum degree, we may assume that if there exists an optimal solution of the problem where all neighbors of j are discarded, then there exists at least one optimal solution where j is selected. Hence, a branching scheme (called *basic branching scheme*) on j of type:

$$\left[l_1, \left(\overline{l_1} - l_2\right), \ldots, \left(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - l_\Delta\right), \left(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_\Delta} - j\right)\right]$$

can be applied. Hence, the following easy but interesting result holds.

Proposition 1. *The* MAX k-VERTEX COVER *problem can be solved to optimality in $O^*(\Delta^k)$.*

Proof. Consider vertex j with maximum degree Δ and neighbors $l_1, \ldots, l_\Delta$ where the basic branching scheme of type $[l_1, (\overline{l_1} - l_2), (\overline{l_1} - \overline{l_2} - l_3), \ldots, (\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - l_\Delta), (\overline{l_1} - \overline{l_2} - \ldots - \overline{l_\Delta} - j)]$ can be applied. Then, the last two branches can be substituted by the branch $(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - j)$ as, if all neighbors of j but one are not selected, any solution including the last neighbor l_Δ but not including j is not better than the solution that selects j.

Now, one can see that the basic branching scheme generates Δ nodes. On the other hand, we know that in each branch of the basic branching scheme at least one vertex is selected. As, at most k nodes can be selected, the overall complexity cannot be superior to $O^*(\Delta^k)$. ♦

Corollary 1. MAX k-VERTEX COVER(k) *in bounded degree graphs is in* **FPT**.

Note that Corollary 1 can also be proved without reference to Proposition 1. Indeed, in any graph of maximum degree Δ, denoting by ℓ the value of an optimal solution for MAX k-VERTEX COVER, $\ell \leqslant k\Delta$. Then, taking ito account that MAX k-VERTEX COVER$(\ell) \in$ **FPT**, immediately derives Corollary 1.

Now, let $V' \subset V$ be a minimum vertex cover of G and let τ be the size of V' that is $\tau = |V'|$. Correspondingly, let $I = V \setminus V'$ be a maximum independent set of G and set $\alpha = |I|$. Notice that V' can be computed, for instance, in $O^*(1.2738^\tau)$ time by means of the fixed-parameter algorithm of [12], and using polynomial space. Let us note that we can assume $k \leqslant \tau$. Otherwise, the optimal value ℓ for MAX k-VERTEX COVER would be equal to $|E|$ and one could compute a minimum vertex cover V' in G and then one could arbitrarily add $k - \tau$ vertices without changing the value of the optimal solution.

Theorem 2. *The following two assertions hold for* MAX k-VERTEX COVER*:*

1. *there exists an $O^*(2^\tau)$-time algorithm that uses polynomial space;*
2. *there exists an algorithm running in time $O^*(\max\{\gamma^\tau, c^k\})$, for two constants $\gamma < 2$ and $c > 4$, and needing polynomial space.*

Proof. For proving item 1, fix some minimum vertex cover V' of G and consider some solution K for MAX k-VERTEX COVER, i.e., some set of k vertices of G. Any such set is distributed over V' and its associated independent set $I = V \setminus V'$. Fix now an optimal solution K^* of MAX k-VERTEX COVER and denote by S' the subset of V' that belongs to K^* (S' can be eventually the empty set) and by I' the part of K^* belonging to I. In other words, the following hold:

$$\begin{aligned} K^* &= S' \cup I' \\ S' &\subseteq V' \\ I' &\subseteq I \;=\; V \setminus V' \end{aligned}$$

Given S' (assume $|S'| = k'$), it can be completed into K^* in polynomial time. Indeed, for each vertex i belonging to I we need simply to compute (in linear time) the total number e_i of edges (i, j) for all $j \in V' \setminus S'$. Then, I' is obtained by selecting the $k - k'$ vertices of I with largest e_i value. So, the following algorithm can be used for MAX k-VERTEX COVER:

1. compute a minimum vertex cover V' (using the algorithm of [11]);
2. for every subset $S' \subseteq V'$ of cardinality at most k, take the $k - |S'|$ vertices of $V \setminus V'$ with the largest degrees to $V' \setminus S'$; denote by I' this latter set;
3. return the best among the sets $S' \cup I'$ so-computed (i.e., the set that covers the maximum of edges).

Step 1 takes time $O^*(1.2738^\tau)$, while step 2 has total running time $O^*(\sum_{i=1}^{k} \binom{\tau}{i})$ that is at most $O^*(2^\tau)$.

Note that, from item 1 of Theorem 2, it can be immediately derived that MAX k-VERTEX COVER can be solved to optimality in $O^*(2^{\frac{\Delta-1}{\Delta}n})$ time. Indeed if a graph G has maximum degree Δ, then for the maximum independent set we have $\alpha \geqslant \frac{n}{\Delta}$. Also, we can assume that G is not a clique on $\Delta + 1$ vertices (note that MAX k-VERTEX COVER is polynomial in cliques). In this case, G can be colored with Δ colors [8]. In such a coloring the cardinality of the largest color is greater than $\frac{n}{\Delta}$ and, a fortiori, so is the cardinality of a maximum independent set (since each color is an independent set). Consequently, $\tau \leqslant \frac{\Delta-1}{\Delta}n$.

In what follows, we improve the analysis of item 1 and prove item 2 that claims, informally, the instances of MAX k-VERTEX COVER that are not fixed-parameter tractable (with respect to k) are those solved with running time better than $O^*(2^\tau)$.

For this observe that the running time of the algorithm in the proof of item 1 is $O^*(\sum_{i=1}^{k} \binom{\tau}{i})$. As mentioned above, k can be assumed to be smaller than, or equal to, τ. Consider some positive constant $\lambda < 1/2$. We distinguish the following two cases: $\tau > k \geqslant \lambda\tau$ and $k < \lambda\tau$.

If $\tau > k \geqslant \lambda\tau$, then $\tau \leqslant k/\lambda$. As $\lambda < 1/2$, $k/\lambda > 2k$ and, since $i \leqslant k$, we get using Stirling's formula:

$$\begin{aligned}\sum_{i=1}^{k}\binom{\tau}{i} \leqslant \sum_{i=1}^{k}\binom{k/\lambda}{i} \quad &\leqslant \quad k\binom{k/\lambda}{k} \quad \sim k \; \frac{\frac{k}{\lambda}^{\frac{k}{\lambda}}}{k^k\left(\frac{k}{\lambda}-k\right)^{\left(\frac{k}{\lambda}-k\right)}} \\ = k\left(\frac{\frac{1}{\lambda}^{\frac{1}{\lambda}}}{\left(\frac{1}{\lambda}-1\right)^{\left(\frac{1}{\lambda}-1\right)}}\right)^{k} \quad &= \quad O^*\left(c^k\right)\end{aligned} \tag{7}$$

for some constant c that depends on λ and it is fixed if λ is so.

If $k < \lambda\tau$, then, by the hypothesis on λ, $2k < \tau$ and, since $i \leqslant k$, expression $\sum_{i=1}^{k}\binom{\tau}{i}$ is bounded above by $k\binom{\tau}{k}$. In all, using also Stirling's formula the following holds:

$$\begin{aligned}\sum_{i=1}^{k}\binom{\tau}{i} \leqslant k\binom{\tau}{k} \quad &\leqslant \quad k\binom{\tau}{\lambda\tau} \quad \sim \quad k\frac{\tau^\tau}{(\lambda\tau)^{(\lambda\tau)}[(1-\lambda)\tau]^{(1-\lambda)\tau}} \\ = k\left(\frac{1}{\lambda^\lambda(1-\lambda)^{(1-\lambda)}}\right)^{\tau} \quad &\overset{\lambda<1/2}{<} \quad O^*\left(2^\tau\right)\end{aligned} \tag{8}$$

In other words, if $k < \lambda\tau$, then MAX k-VERTEX COVER can be solved in time at most $O^*(\gamma^\tau)$, for some γ that depends on λ and is always smaller than 2 for $\lambda < 1/2$.

Table 1. The values of c and γ for some values of λ

λ	0.01	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.40	0.45	0.49
$\frac{\frac{1}{\lambda^{\frac{1}{\lambda}}}}{(\frac{1}{\lambda}-1)^{(\frac{1}{\lambda}-1)}}$	270.47	53.00	25.81	16.74	12.21	9.48	7.66	6.36	5.38	4.61	4.11
$\frac{1}{\lambda^{\lambda}(1-\lambda)^{1-\lambda}}$	1.06	1.22	1.38	1.53	1.65	1.75	1.84	1.91	1.96	1.99	1.9996

Expressions (7) and (8) derive the claim and conclude the proof. In Table 1 the values of c and γ are given for some values of λ. ♦

Let us note that the technique of item 1 of Theorem 2, that consists of determining a decomposition of the input graph into a minimum vertex cover and a maximum independent set and then of taking a subset S' of a minimum vertex cover V' of the input-graph and of completing it into an optimal solution can be applied to several other well-known combinatorial **NP**-hard problems. We sketch here some examples:

- in MIN 3-DOMINATING SET (dominating set in graphs of maximum degree 3), the set S' is completed in the following way:
  - take all the vertices in $I \setminus \Gamma_I(S')$ (in order to dominate vertices in $V' \setminus S'$);
  - if there remain vertices of $V' \setminus S'$ not dominated yet solve a MIN SET COVER problem considering $\Gamma_I(S')$ as the set-system of the latter problem and assuming that a vertex $v \in \Gamma_I(S')$, seen as set, contains its neighbors in $V' \setminus S'$ as elements; since $\Gamma_I(S')$ is the neighborhood of S', the degrees of its vertices to $V' \setminus S'$ are bounded by 2, that induces a polynomial MIN SET COVER problem ([27]);
- in MIN INDEPENDENT DOMINATING SET, S' is completed by the set $I \setminus \Gamma_I(S')$, where $\Gamma_I(S')$ is the set of neighbors of S' that belong to I;
- in EXISTING DOMINATING CLIQUE, MIN DOMINATING CLIQUE (if any), MAX DOMINATING CLIQUE (if any) and MAX CLIQUE, S' can eventually be completed by a single vertex of $\Gamma_I(S')$.

Theorem 3. MIN INDEPENDENT DOMINATING SET*,* EXISTING DOMINATING CLIQUE*,* MIN DOMINATING CLIQUE*,* MAX DOMINATING CLIQUE*,* MAX CLIQUE *and* MIN 3-DOMINATING SET *can be solved in time $O^*(2^{\tau})$ using polynomial space.*

4 Tailoring Measure-and-Conquer to Graphs with Maximum Degree 3

Let us note that, as it is proved in [23], for any $\epsilon > 0$, there exists an integer n_ϵ such that the pathwidth of every (sub)cubic graph of order $n > n_\epsilon$ is at most $(1/6 + \epsilon)n$. Based upon the fact that there exists for MAX k-VERTEX COVER(w) an $O^*(2^w)$-time exponential space algorithm [34], and taking into account that in (sub)cubic graphs $w \leqslant (1/6+\epsilon)n$, the following corollary is immediately derived.

Corollary 2. MAX k-VERTEX COVER *in graphs with maximum degree 3 can be solved in time* $O^*(2^{n/6}) = O^*(1.123^n)$ *using exponential space.*

In this section we tailor the measure-and-conquer approach developed in Section 2 to graphs with $\Delta = 3$, in order to get an improved running-time algorithm for this case needing polynomial space. The following remark holds.

Remark 1. The graph can be cubic just once. When branching on a vertex j of maximum degree 3, we can always assume that it is adjacent to at least one vertex h that has already been selected or discarded. That is, either $d_h \leqslant 2$, or $\alpha_h \geqslant 1$, that is $c_h \leqslant 2$. Indeed, the situation where the graph is 3-regular occurs at most once (even in case of disconnection). Thus, we make only one "bad" branching (where every free vertex of maximum degree 3 is adjacent only to free vertices of degree 3). Such a branching may increase the global running time only by a constant factor.

Lemma 1. *Any vertex* i *with* $d_i \leqslant 1$ *and* $\alpha_i = 0$ *can be discarded w.l.o.g.*

Proof. If $d_i = \alpha_i = 0$, then i can be obviously discarded. If $d_i = 1$ and $\alpha_i = 0$, then i is adjacent to another free vertex h. But then, if h is selected, i becomes of degree 0 and can be discarded. Alternatively, h is discarded, but then any solution with i but not h is dominated by that including h instead of i. ♦

Lemma 2. *Any vertex* i *with* $\alpha_i \geqslant 2$ *and* $d_i = 3$ *can be selected w.l.o.g.*

Proof. If $\alpha_i = 3$, then i can be obviously selected. If $d_i = 3$ and $\alpha_i = 2$, then i is adjacent to another free vertex h. But then, if h is discarded, we have $\alpha_i = 3$ and i can be selected. Alternatively, h is selected, but then any solution with h but not i is dominated by that including i instead of h. ♦

To solve MAX k-VERTEX COVER on graphs with $\Delta = 3$, consider the following algorithm, called `MAXKVC-3`.

> Select j such that c_j is maximum and branch according to the following exhaustive cases:
>
> 1. if $c_j = 3$, assume, w.l.o.g., that j is adjacent to i, l, m free vertices with $c_i \leqslant 2$ (see in [14]) and $c_i \leqslant c_l \leqslant c_m$, and branch on j according to the following exhaustive subcases:
>   (a) $c_i = c_l = c_m = 1$
>   (b) $c_i = c_l = 1$, $c_m = 2$
>   (c) $c_i = c_l = 1$, $c_m = 3$
>   (d) $c_i = 1$, $c_l = c_m = 2$ with l, m adjacent
>   (e) $c_i = 1$, $c_l = c_m = 2$ with l, m non adjacent
>   (f) $c_i = 1$, $c_l = 2$, $c_m = 3$
>   (g) $c_i = c_l = 2$, $c_m = 3$ with i, l adjacent
>   (h) $c_i = c_l = 2$, $c_m = 3$ with i, l non adjacent
>   (i) $c_i = 2$, $c_l = c_m = 3$
> 2. else $c_j \leqslant 2$ and `MAXKVC-3` is polynomially solvable.

The following Theorem 4 holds in graphs with maximum degree 3 (due to space constraints, the proof is omitted; it can be found in [14]).

Theorem 4. *Algorithm* `MAXKVC-3` *solves* MAX k-VERTEX COVER *on graphs with maximum degree 3 with running time* $O^*(1.3339^n)$ *and using polynomial space.*

5 Approximating MAX k-VERTEX COVER by Moderately Exponential Algorithms

We now show how one can get approximation ratios non-achievable in polynomial time using moderately exponential algorithms with worst-case running times better than those required for an exact computation (see [4,5] for more about this issue). Denote by $\text{opt}(G)$ the cardinality of an optimal solution for MAX k-VERTEX COVER in G and by $m(G)$, the cardinality of an approximate solution. Our goal is to study the approximation ratio $m(G)/\text{opt}(G)$.

In what follows, we denote, as previously, by K^* the optimal solution for MAX k-VERTEX COVER. Given a set K of vertices, we denote by $C(K)$, the set of edges covered by K (in other words, the value of a solution K for MAX k-VERTEX COVER is $|C(K)|$; also, according to our previous notation, $\text{opt}(G) = |C(K^*)|$). We first prove the following easy lemma that will be used later.

Lemma 3. *For any* $\lambda \in [0,1]$*, the subset* H^* *of* λk *vertices of* K^* *covering the largest amount of edges covered by* K^**, covers at least* $\lambda\,\text{opt}(G)$ *edges.*

Proof. Indeed, if the λk "best" vertices of K^* cover less than $\lambda\,\text{opt}(G)$ edges, then any disjoint union of k/λ subsets of K^*, each of cardinality λk covers less than $\text{opt}(G)$ edges, a contradiction. ♦

Now, run the following algorithm, called `APPROX` in what follows:

1. fix some $\lambda \in [0,1]$ and optimally solve MAX λk-VERTEX COVER in G (as previously, let H^* be the optimal solution built and $C(H^*)$ be the edge-set covered by H^*);
2. remove H^* and $C(H^*)$ from G and approximately solve MAX $(1-\lambda)k$-VERTEX COVER in the surviving graph (by some approximation algorithm); let K' be the obtained solution;
3. output $K = H^* \cup K'$.

It is easy to see that if $T(p,k)$ is the running time of an optimal algorithm for MAX k-VERTEX COVER, where p is some parameter of the input-graph G (for instance, n, or τ), then the complexity of `APPROX` is $T(p, \lambda k)$. Furthermore, `APPROX` requires polynomial space.

Theorem 5. *If* $T(p,k)$ *is the running time of an optimal algorithm for* MAX k-VERTEX COVER*, then, for any* $\epsilon > 0$*,* MAX k-VERTEX COVER *can be approximated within ratio* $1-\epsilon$ *with worst-case running time* $T(p,(1+2\sqrt{1-3\epsilon})k/3)$ *and polynomial space.*

Proof. Denote by K^* an optimal solution of MAX k-VERTEX COVER in G, by G_2 the induced subgraph $G[V \setminus H^*]$ of G, by $\mathrm{opt}_{(1-\lambda)}(G_2)$, the value of an optimal for MAX $(1-\lambda)k$-VERTEX COVER in G_2. Suppose that E' edges are common between $C(H^*)$ and $C(K^*)$. This means that $C(K^*) \setminus E'$ edges of $C(K^*)$ are in G_2 and are exclusively covered by the vertex-set $L^* = K^* \setminus H^*$ that belongs to G_2. Set $\ell^* = |L^*|$ and note that $\ell^* \leqslant k$ and $\ell^* \geqslant (1-\lambda)k$.

According to Lemma 3, the $(1-\lambda)k$ "best" vertices of L^* cover more than $(1-\lambda)|C(K^*) \setminus E'| = (1-\lambda)(\mathrm{opt}(G) - |E'|)$ edges in G_2 and these vertices constitute a feasible solution for MAX $(1-\lambda)k$-VERTEX COVER in G_2. Hence:

$$\mathrm{opt}_{(1-\lambda)}(G_2) \geqslant (1-\lambda)(\mathrm{opt}(G) - |E'|) \tag{9}$$

Taking into account (9), the fact that K' in step 2 of `APPROX` has been computed by, say, a ρ-approximation algorithm and the fact that $|E'| \leqslant |C(H^*)|$, we get:

$$\begin{aligned} m(G) = C(H^*) + C(K') &\geqslant C(H^*) + \rho(1-\lambda)\mathrm{opt}_{(1-\lambda)}(G_2) \\ &\geqslant C(H^*) + \rho(1-\lambda)(\mathrm{opt}(G) - |E'|)\, C(H^*) \\ &\quad + \rho(1-\lambda)(\mathrm{opt}(G) - C(H^*)) \\ &\geqslant (1-\rho(1-\lambda))C(H^*) + \rho(1-\lambda)\,\mathrm{opt}(G) \end{aligned} \tag{10}$$

Using once more Lemma 3, $|C(H^*)| \geqslant \lambda\,\mathrm{opt}(G)$, and combining it with (10), we get:

$$\frac{m(G)}{\mathrm{opt}(G)} \geqslant \rho(1-\lambda) + \lambda(1-\rho(1-\lambda)) \tag{11}$$

Setting $\rho = \frac{3}{4}$ in (11), in order to achieve an approximation ratio $m(G)/\mathrm{opt}(G) = 1-\epsilon$, for some $\epsilon > 0$, we have to choose an λ satisfying $\lambda = (1+2\sqrt{1-3\epsilon})/3$, that completes the proof of the theorem. ♦

Corollary 3. MAX k-VERTEX COVER *can be approximated within ratio* $1-\epsilon$ *and with running time:*

$$\min\left\{O^*\left(n^{\left(1+2\sqrt{1-3\epsilon}\right)(\omega k)/9}\right), O^*\left(\frac{\tau}{\left(1+2\sqrt{1-3\epsilon}\right)k/3}\right)\right\}$$

and polynomial space.

For Corollary 3, just observe that the running-times claimed for the first two entries are those needed to optimally solve MAX λk-VERTEX COVER (the former due to [9] and the latter due to item 1 of Theorem 2). Note that the second term in the min expression in the corollary is an FPT approximation schema (with respect to parameter τ). Observe also that for the cases where the time needed for solving MAX k-VERTEX COVER is given by the c^k expression of item 1 of Theorem 2, this represents an improvement with respect to the FPT approximation schema of [32]. Note finally that the result of Theorem 5 is indeed a kind of reduction between moderately exponential (or parameterized) approximation and exact (or parameterized) computation for MAX k-VERTEX COVER in the sense that exact solution on some subinstance of the problem derives an approximation for the whole instance.

Finally, let us close this section and the paper by some remarks on what kind of results can be expected in the area of (sub)exponential approximation. All the algorithms given in this section have exponential running time when we seek for a *constant* approximation ratio (unachievable in polynomial time). On the other hand, for several problems that are hard to approximate in polynomial time (like MAX INDEPENDENT SET, MIN COLORING, ...), subexponential time can be easily reached for ratios depending on the input-size (thus tending to ∞, for minimization problems, or to 0, for maximization problems). An interesting question is to determine, for these problems, if it is possible to devise a constant approximation algorithm working in subexponential time. An easy argument shows that this is not always the case. For instance, the existence of subexponential approximation algorithms (within ratio better than 4/3) is quite improbable for MIN COLORING since it would imply that 3-COLORING can be solved in subexponential time, contradicting so the "exponential time hypothesis" [29]. We conjecture that this is true for any constant ratio for MIN COLORING. Anyway, the possibility of devising subexponential approximation algorithms for **NP**-hard problems, achieving ratios forbidden in polynomial time or of showing impossibility of such algorithms is an interesting open question that deserves further investigation.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12(2), 308–340 (1991)
2. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. 39(2), 546–563 (2009)
3. Bläser, M.: Computing small partial coverings. Inform. Process. Lett. 85(6), 327–331 (2003)
4. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient approximation by "low-complexity" exponential algorithms. Cahier du LAMSADE 271, LAMSADE, Université Paris-Dauphine (December 2007), http://www.lamsade.dauphine.fr/cahiers/PDF/cahierLamsade271.pdf
5. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient Approximation of Combinatorial Problems by Moderately Exponential Algorithms. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 507–518. Springer, Heidelberg (2009)
6. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient approximation of MIN COLORING by moderately exponential algorithms. Inform. Process. Lett. 109(16), 950–954 (2009)
7. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient approximation of MIN SET COVER by moderately exponential algorithms. Theoret. Comput. Sci. 410(21-23), 2184–2195 (2009)
8. Brooks, R.L.: On coloring the nodes of a network. Math. Proc. Cambridge Philos. Soc. 37, 194–197 (1941)
9. Cai, L.: Parameter complexity of cardinality constrained optimization problems. The Computer Journal 51, 102–121 (2008)

10. Cai, L., Huang, X.: Fixed-Parameter Approximation: Conceptual Framework and Approximability Results. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 96–108. Springer, Heidelberg (2006)
11. Chen, J., Kanj, I., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms 41, 280–301 (2001)
12. Chen, J., Kanj, I., Xia, G.: Improved upper bounds for vertex cover. Theoret. Comput. Sci. 411(40-42), 3736–3756 (2010)
13. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
14. Croce, F.D., Paschos, V.T.: On the max k-vertex cover problem. Cahier du LAMSADE 307, LAMSADE, Université Paris-Dauphine (2011)
15. Cygan, M., Kowalik, L., Wykurz, M.: Exponential-time approximation of weighted set cover. Inform. Process. Lett. 109(16), 957–961 (2009)
16. Cygan, M., Pilipczuk, M.: Exact and approximate bandwidth. Theoret. Comput. Sci. 411(40–42), 3701–3713 (2010)
17. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
18. Downey, R.G., Fellows, M.R., McCartin, C.: Parameterized Approximation Problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 121–129. Springer, Heidelberg (2006)
19. Feige, U., Langberg, M.: Approximation algorithms for maximization problems arising in graph partitioning. J. Algorithms 41(2), 174–211 (2001)
20. Fellows, M.: Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 2–10. Springer, Heidelberg (2009)
21. Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph Layout Problems Parameterized by Vertex Cover. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 294–305. Springer, Heidelberg (2008)
22. Fomin, F., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. Assoc. Comput. Mach. 56(5), 1–32 (2009)
23. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inform. Process. Lett. 97(5), 191–196 (2006)
24. Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S.: Subexponential algorithms for partial cover problems. Inform. Process. Lett. 111(16), 814–818 (2011)
25. Fomin, F.V., Villanger, Y.: Treewidth Computation and Extremal Combinatorics. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 210–221. Springer, Heidelberg (2008)
26. Fürer, M., Gaspers, S., Kasiviswanathan, S.P.: An Exponential Time 2-Approximation Algorithm for Bandwidth. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 173–184. Springer, Heidelberg (2009)
27. Garey, M.R., Johnson, D.S.: Computers and intractability. W. H. Freeman, San Francisco (1979)
28. 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)
29. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. System Sci. 63(4), 512–530 (2001)
30. Jäger, G., Srivastav, A.: Improved approximation algorithms for maximum graph partitioning problems. J. Comb. Optim. 10(2), 133–167 (2005)

31. 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)
32. Marx, D.: Parameterized complexity and approximation algorithms. The Computer Journal 51(1), 60–78 (2008)
33. Marx, D.: Fixed parameter algorithms. Open lectures for PhD students in computer science (January 2010)
34. Moser, H.: Exact algorithms for generalizations of vertex cover. Master's thesis, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena (2005)
35. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. Comment. Math. Univ. Carolinae, 415–419 (1985)
36. Niedermeier, R., Rossmanith, P.: On efficient fixed-parameter algorithms for weighted vertex cover. J. Algorithms 47(2), 63–77 (2003)
37. Praveen, M.: Logic, Courcelle's theorem and application. IMPECS School on Parameterized and Exact Computation (December 2010)

A Model Theoretic Proof of Completeness of an Axiomatization of Monadic Second-Order Logic on Infinite Words

Colin Riba*

ENS de Lyon, Université de Lyon, LIP
colin.riba@ens-lyon.fr
http://perso.ens-lyon.fr/colin.riba/

Abstract. We discuss a complete axiomatization of Monadic Second-Order Logic (MSO) on infinite words.By using model-theoretic methods, we give an alternative proof of D. Siefkes' result that a fragment with full comprehension and induction of second-order Peano's arithmetic is complete w.r.t. the validity of MSO-formulas on infinite words. We rely on Feferman-Vaught Theorems and the Ehrenfeucht-Fraïssé method for Henkin models of MSO. Our main technical contribution is an infinitary Feferman-Vaught Fusion of such models. We show it using Ramseyan factorizations similar to those for standard infinite words.

1 Introduction

We discuss the completeness of an axiomatization of Monadic Second-Order Logic (MSO) on infinite words. MSO on infinite words is known to be decidable since the celebrated work of Büchi [2]. The usual route is to translate MSO-formulas to finite state automata running on infinite words. Such automata provide an established framework for the specification and verification of non-terminating programs, while MSO is a yardstick language for expressing properties about them. We refer to e.g. [7,6,8] for comprehensive presentations of the subject.

D. Siefkes has shown in [11] that a fragment of second-order Peano's arithmetic containing the comprehension axiom scheme and the induction axiom is complete with respect to the standard model: every MSO-formula true on infinite words is provable. The approach taken there was to formalize the translation of MSO-formulas to Büchi automata. This requires to represent automata in the logic and to formalize the correctness proof of the translation in the corresponding deduction system.

In this paper, we give an alternative proof of Siefkes' completeness result by using model-theoretic tools. This leads to a more abstract proof which does not require explicit manipulation of automata in the logic. To our knowledge, such approaches to MSO have not been much explored compared to the great body of work on automata and corresponding algebraic structures [6,8].

* UMR 5668 CNRS ENS Lyon UCBL INRIA.

We follow the method of [5], where complete axiomatizations of variants of MSO on finite trees are presented. Starting from Henkin completeness, we show that all models of our axiomatization are equivalent w.r.t. the validity of MSO-formulas. As in [5], we use Feferman-Vaught Theorems obtained by the Ehrenfeucht-Fraïssé method [10].

In contrast with [10,7], works like [5] or the present one have to handle non-standards models of second-order arithmetic. As far as Henkin completeness is concerned, a model $\mathcal{M}$ of MSO can be seen as a structure with two domains: a domain $\mathcal{M}^\iota$ of individuals and a domain $\mathcal{M}^o \subseteq \mathcal{P}(\mathcal{M}^\iota)$ of sets of individuals (called *predicates* in this paper). Besides non-standards individuals (whose order type is very different from ω), the main difficulty is that $\mathcal{M}^o$ is in general strictly contained in $\mathcal{P}(\mathcal{M}^\iota)$: there might not be "enough" predicates.

A crucial observation due to K. Doets [3] makes apparent in (possibly non-standard) models a structure similar to standard infinite words. Our main technical contribution is a kind of Feferman-Vaught Infinitary Fusion for such models. Intuitively, it is a model-theoretic counterpart to a run of a Büchi automaton on a standard infinite word. The point is to ensure that such a "run" always exists as a predicate of a given model. For this, we use Ramseyan factorizations similar to those of infinite words (see e.g. [8]).

The paper is organized as follows. In Section 2, we describe our formal system for MSO, as well as the class of models we are interested in. These models are motivated by usual results on Henkin completeness for second-order logic that we briefly recall. We present in Section 3 the notions on the Ehrenfeucht-Fraïssé method that we will need. We use it to prove a Feferman-Vaught Finite Sums Lemma for linearly ordered structures with parameters, which is discussed in Section 4. We then give the main argument for completeness in Section 5. It relies on an infinitary version of the Finite Sums Lemma, that we call "Infinite Fusion" and which is shown in Section 6.

A full version of this paper is available on the author's web page `http://perso.ens-lyon.fr/colin.riba/papers/msofull.pdf`.

2 A Deduction System for Monadic Second-Order Logic on Infinite Words

2.1 Language

We consider a formulation of *Monadic Second-Order Logic* (MSO) based on a two-sorted language: There is one sort ι intended to range over *individuals* and one sort o intended to range over *monadic* (or one-place) *predicates on individuals*. We assume given two countable sets $\mathcal{V}_\iota = \{x, y, z, \dots\}$ and $\mathcal{V}_o = \{X, Y, Z, \dots\}$ of respectively individual and predicate *variables*. The formulas of MSO are then defined by the following grammar:

$$\phi, \psi \in \Lambda \quad ::= \quad Xx \quad | \quad x < y \quad | \quad \neg\phi \quad | \quad \phi \vee \psi \quad | \quad \exists X\, \phi \quad | \quad \exists x\, \phi$$

The set $\mathrm{FV}(\phi)$ of free (individual and predicate) variables of a formula ϕ is defined as usual. A *sentence* (or *closed formula*) is a formula with no free variable, *i.e.* a formula ϕ such that $\mathrm{FV}(\phi) = \emptyset$. Formulas are identified modulo renaming of their bound variables. The capture-avoiding substitution of y for x in ϕ is written $\phi[y/x]$.

Note that there is no primitive equality in Λ. This is discussed in Section 2.4. The other logical connectives are defined as usual:

$$\begin{array}{rcl} \phi \to \psi & := & \neg\phi \vee \psi \\ \phi \wedge \psi & := & \neg(\neg\phi \vee \neg\psi) \\ \phi \longleftrightarrow \psi & := & (\phi \to \psi) \wedge (\psi \to \phi) \end{array} \qquad \begin{array}{rcl} \forall X\, \phi & := & \neg\exists X\, \neg\phi \\ \forall x\, \phi & := & \neg\exists x\, \neg\phi \end{array}$$

2.2 Deduction for Second-Order Logic

We now discuss formal deduction for second-order logic. As usual, the rules for second-order logic are those of the (two-sorted) classical predicate calculus together with the comprehension axiom scheme (see e.g. [9]). There are several different formulations equivalent w.r.t. provability. The following Natural Deduction system is a possible choice.

The *deduction relation* is writen $\Gamma \vdash \phi$, where Γ is a (possibly empty) finite unordered list of (possibly not closed) formula, and ϕ is a (possibly not closed) formula. It is inductively defined by the following rules.

– Rules for propositional logic:

$$\frac{}{\Gamma \vdash \phi \vee \neg\phi} \qquad \frac{}{\Gamma, \phi \vdash \phi} \qquad \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \neg\phi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \varphi \qquad \Gamma, \psi \vdash \varphi}{\Gamma \vdash \varphi}$$

– Rules for predicate logic (where $\mathcal{X}, \mathcal{Y} \in \mathcal{V}_\iota$ or $\mathcal{X}, \mathcal{Y} \in \mathcal{V}_o$):

$$\frac{\Gamma \vdash \phi[\mathcal{Y}/\mathcal{X}]}{\Gamma \vdash \exists \mathcal{X}\, \phi} \qquad \frac{\Gamma \vdash \exists \mathcal{X}\, \phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}\ (\mathcal{X} \notin \mathrm{FV}(\Gamma, \psi))$$

– Comprehension scheme (for all formula ϕ):

$$\frac{}{\Gamma \vdash \exists X\, \forall x\, (Xx \longleftrightarrow \phi)}\ (X \notin \mathrm{FV}(\phi))$$

2.3 Models of Second-Order Logic

We discuss the class of structures (or models) we will use to interpret the language of MSO presented in Section 2.1. These structures are motivated by known results on Henkin completeness that we briefly recall.

Structures, Assignments and Satisfiability. We consider *(Henkin) structures* $\mathcal{M}$ of the form $(\mathcal{M}^\iota, \mathcal{M}^o, <_\mathcal{M})$ where $\mathcal{M}^\iota$ is a *non-empty* set of *individuals*, $\mathcal{M}^o \subseteq \mathcal{P}(\mathcal{M}^\iota)$ is a *non-empty* set of *predicates* and $<_\mathcal{M}$ is a binary relation on $\mathcal{M}^\iota$. We call $\mathcal{M}^\iota$ and $\mathcal{M}^o$ respectively the *individual* and *predicate domains* of $\mathcal{M}$.

An $\mathcal{M}$-*assignment* is a map $\rho : (\mathcal{V}_\iota \cup \mathcal{V}_o) \to (\mathcal{M}^\iota \cup \mathcal{M}^o)$ which respects the sorts, *i.e.* such that $\rho(x) \in \mathcal{M}^\iota$ and $\rho(X) \in \mathcal{M}^o$ if $x \in \mathcal{V}_\iota$ and $X \in \mathcal{V}_o$. Given $x \in \mathcal{V}_\iota$ and $a \in \mathcal{M}^\iota$, we write $\rho[a/x]$ for the assignment which maps x to a and is equal to ρ everywhere else. The assignment $\rho[A/X]$ (where $X \in \mathcal{V}_o$ and $A \in \mathcal{M}^o$) is defined similarly.

Given a structure $\mathcal{M}$, an $\mathcal{M}$-assignment ρ and a formula ϕ, we define the *satisfaction relation* $\mathcal{M}, \rho \models \phi$ by induction on ϕ as usual:

$$\begin{array}{lll}
\mathcal{M}, \rho \models Xx & \text{iff} & \rho(x) \in \rho(X) \\
\mathcal{M}, \rho \models x < y & \text{iff} & \rho(x) <_\mathcal{M} \rho(y) \\
\mathcal{M}, \rho \models \neg\phi & \text{iff} & \mathcal{M}, \rho \not\models \phi \\
\mathcal{M}, \rho \models \phi \vee \psi & \text{iff} & \mathcal{M}, \rho \models \phi \text{ or } \mathcal{M}, \rho \models \psi \\
\mathcal{M}, \rho \models \exists X\, \phi & \text{iff} & \text{there is some } A \in \mathcal{M}^o \text{ such that } \mathcal{M}, \rho[A/X] \models \phi \\
\mathcal{M}, \rho \models \exists x\, \phi & \text{iff} & \text{there is some } a \in \mathcal{M}^\iota \text{ such that } \mathcal{M}, \rho[a/x] \models \phi
\end{array}$$

We say that ϕ is *valid* in $\mathcal{M}$ (notation $\mathcal{M} \models \phi$) if $\mathcal{M}, \rho \models \phi$ for every ρ. A set of formulas Δ is valid in $\mathcal{M}$ (notation $\mathcal{M} \models \Delta$) if $\mathcal{M} \models \phi$ for every $\phi \in \Delta$.

It is sometimes convenient to consider formulas with a fixed assignment of their free variables to some structure $\mathcal{M}$. These formulas are called *formulas with parameters in* $\mathcal{M}$. We define them as pairs of a formula ϕ and a *finite partial* $\mathcal{M}$-assignment $\nu : (\mathcal{V}_\iota \cup \mathcal{V}_o) \rightharpoonup (\mathcal{M}^\iota \cup \mathcal{M}^o)$. The set of free variables of the formula with parameters (ϕ, ν) is $\mathrm{FV}(\phi, \nu) := \mathrm{FV}(\phi) \setminus \mathrm{dom}(\nu)$. We will often write $\phi[\nu(\mathcal{X})/\mathcal{X} \mid \mathcal{X} \in \mathrm{dom}(\nu)]$ for the formula with parameters (ϕ, ν).

The satisfaction of a formula with parameters (ϕ, ν) in a structure $\mathcal{M}$ and assignment ρ (notation $\mathcal{M}, \rho \models (\phi, \nu)$) is defined as the satisfaction of ϕ in $\mathcal{M}$ and assignment $\rho[\nu(\mathcal{X})/\mathcal{X} \mid \mathcal{X} \in \mathrm{dom}(\nu)]$. The corresponding validity relation $\mathcal{M} \models (\phi, \nu)$ holds if $\mathcal{M}, \rho \models (\phi, \nu)$ for every ρ.

Second-Order Henkin Structures. Deduction *without the comprehension scheme* is correct in any structure $\mathcal{M}$: if $\vdash \phi$ is derivable without using the comprehension then ϕ is valid in $\mathcal{M}$. The following notions are useful to handle the comprehension scheme. A set of individuals $A \in \mathcal{P}(\mathcal{M}^\iota)$ is *definable* if there is a formula ϕ and an $\mathcal{M}$-assignment ρ such that

$$A \quad = \quad \{a \in \mathcal{M}^\iota \mid \mathcal{M}, \rho[a/x] \models \phi\}$$

Of course, all $A \in \mathcal{M}^o$ are definable. The converse is more interesting, since $\mathcal{M}$ satisfies every instance of the comprehension scheme if and only if $\mathcal{M}^o$ is the set of all definable $A \in \mathcal{P}(\mathcal{M}^\iota)$. In this case, we call $\mathcal{M}$ a *second-order* (Henkin) structure.

Remark 2.1. (i) We say that $\mathcal{M}$ is *full* if $\mathcal{M}^o = \mathcal{P}(\mathcal{M}^\iota)$. Full structures are second-order.

(ii) Finite boolean combinations of definable predicates are definable. Hence, the predicate domain of a second-order structure is closed under finite boolean operations.

Henkin Completeness. Usual Henkin completeness holds for deduction w.r.t. validity in all second-order Henkin structures (see e.g. [9]):

Theorem 2.2 (Henkin Completeness). *Let Δ be a set of sentences and ϕ be a sentence. Assume that for all second-order Henkin structure $\mathcal{M}$, if $\mathcal{M} \models \Delta$ then $\mathcal{M} \models \phi$. Then there is a finite set $\Gamma \subseteq \Delta$ such that $\Gamma \vdash \phi$.*

2.4 Equality

Monadic Second-Order Logic has a definable equality (see e.g. [9]):

$$(x \doteq y) \ := \ \forall X \, (Xx \to Xy)$$

Thanks to the comprehension scheme, it is an equivalence relation which moreover satisfies Leibniz's scheme:

$$\vdash \forall x \, (x \doteq x) \qquad \vdash \forall xy \, (x \doteq y \to y \doteq x) \qquad \vdash \forall xyz \, (x \doteq y \to y \doteq z \to x \doteq z)$$

$$\vdash \forall xy \, (x \doteq y \to \phi[x/z] \to \phi[y/z]) \qquad \text{(for all formula } \phi\text{)}$$

Remark 2.3. Given a second-order structure $\mathcal{M}$, we have $\mathcal{M}^\iota, \emptyset \in \mathcal{M}^o$ since $\mathcal{M}^\iota$ is definable by the formula $(x \doteq x)$.

Second-Order Structures with Correct Equality. It is well-known that the equality $\doteq$ may not be correct: Given a structure $\mathcal{M}$, it is possible that $\mathcal{M} \models (a \doteq b)$ but $a \neq b$, even if $\mathcal{M}$ is second-order (see e.g. [9]). We say that a structure $\mathcal{M}$ has *correct equality* if $\mathcal{M} \models (a \doteq b)$ implies $a = b$ for all $a, b \in \mathcal{M}^\iota$.

Remark 2.4. (i) Full structures have correct equality.

(ii) Consider an arbitrary structure $\mathcal{M}$ with correct equality. Note that every singleton $\{a\}$ with $a \in \mathcal{M}^\iota$ is definable (by the formula with parameters $(x \doteq y, [a/x])$). According to Remark 2.1.(ii), it follows that if $\mathcal{M}$ is second-order, then $\mathcal{M}^o$ contains all the finite subsets of $\mathcal{M}^\iota$.
In particular, finite second-order structures with correct equality are full.

As far as Henkin completeness is concerned, it is always possible to assume that a second-order structure has correct equality. We in fact have the following strengthening of Henkin completeness (see e.g. [9]):

Corollary 2.5. *Let Δ be a set of sentences and ϕ be a sentence. Assume that for all second-order Henkin structure $\mathcal{M}$ with correct equality, if $\mathcal{M} \models \Delta$ then $\mathcal{M} \models \phi$. Then there is a finite set $\Gamma \subseteq \Delta$ such that $\Gamma \vdash \phi$.*

2.5 Axiomatization

The *standard model* is $\mathsf{N} := (\mathbb{N}, \mathcal{P}(\mathbb{N}), <_{\mathbb{N}})$, where $<_{\mathbb{N}}$ is the usual order on natural numbers. Recall that thanks to the celebrated result of Büchi [2], the *monadic theory* of N (*i.e.* the set of sentences ϕ such that $\mathsf{N} \models \phi$) is decidable.

In this section, we describe a set MSO^ω of sentences which *completely axiomatizes* the monadic theory of N: for all sentence ϕ, if $\mathsf{N} \models \phi$ then $\mathsf{MSO}^\omega \vdash \phi$. The axiomatization we consider is an adaptation of that of [11] to the language of MSO presented in Section 2.1. This is essentially a fragment of second-order Peano's arithmetic with full comprehension and induction.

For the completeness proof of MSO^ω, we shall also discuss variations on Ramsey's theorem and the axiom of choice in Sections 5 and 6.

Definition 2.6 (MSO^ω). *MSO^ω is the set of the following sentences:*

- *Linear Order axioms:*

$$\forall x\, \neg(x < x) \qquad \forall xyz\, (x < y \to y < z \to x < z)$$
$$\forall xy\, (x < y \lor x \doteq y \lor y < x)$$

- *Unboundedness axiom:*

$$\forall x\, \exists y\, (x < y)$$

- *Induction axiom:*

$$\forall X\ [\forall x\, (\forall y\, (y < x \to Xy) \to Xx) \to \forall x\, Xx]$$

- *Predecessor axiom:*

$$\forall x\, (\exists y(y < x) \to \exists y[y < x \land \neg\exists z\, (y < z \land z < x)])$$

A formula ϕ is derivable in MSO^ω *if $\mathsf{MSO}^\omega \vdash \phi$ is derivable using the deduction system of Section 2.2.*

A second-order structure with correct equality $\mathcal{M}$ is a model of MSO^ω *if $\mathcal{M} \models \mathsf{MSO}^\omega$.*

In this paper, we give a model-theoretic proof of Siefkes' completeness result:

Theorem 2.7 (Completeness of MSO^ω [11]). *For all sentence ϕ, if $\mathsf{N} \models \phi$ then $\mathsf{MSO}^\omega \vdash \phi$.*

Following the method of [5], our route to Theorem 2.7 is to use usual Henkin completeness (as formulated in Corollary 2.5), and to show that all models of MSO^ω are equivalent w.r.t. the validity of MSO-formulas. This is the main result of the paper.

Theorem 2.8 (Main Theorem). *Let $\mathcal{M}$ be a model of MSO^ω and ϕ be a sentence. We have $\mathcal{M} \models \phi$ if and only if $\mathsf{N} \models \phi$.*

Theorem 2.8 is proved in Section 5. As [5], we rely on Feferman-Vaught Theorems proved by the Ehrenfeucht-Fraïssé method.

We now discuss some aspects of the different axioms of MSO^ω. All structures considered here are second-order and have correct equality.

Orders. We use the following defined formula:

$$x \leq y \ := \ x < y \vee x \doteq y$$

Hence, in a structure $\mathcal{M}$ with correct equality, given $a, b \in \mathcal{M}^\iota$ we have $\mathcal{M} \models a \leq b$ if and only if ($a = b$ or $a <_\mathcal{M} b$).

A structure $\mathcal{M}$ is *linearly ordered* if is satisfies the Linear Order axioms. The first two sentences say that $<_\mathcal{M}$ is strict and transitive. Note that $<_\mathcal{M}$ is thus antisymmetric: if $a <_\mathcal{M} b$ then $b \not<_\mathcal{M} a$. The third sentence says that $<_\mathcal{M}$ is total. Since $\mathcal{M}$ is assumed to have correct equality, it is equivalent to requiring that for all $a, b \in \mathcal{M}^\iota$ we have either $a <_\mathcal{M} b$ or $a = b$ or $b <_\mathcal{M} a$.

Induction. The induction axiom holds in the standard model N but is false for instance in the full structure of real numbers.[1]

Assume that $\mathcal{M}$ satisfies the induction axiom. The contrapositive of induction says that each non-empty predicate $A \in \mathcal{M}^o$ has minimal elements. If moreover $\mathcal{M}$ is linearly ordered, then A has a unique least element.

Successors and Predecessors. If $\mathcal{M}$ is linearly ordered and satisfies the induction axiom, then every $a \in \mathcal{M}^\iota$ which is not maximal has a *successor*, *i.e.* there is a unique least $b >_\mathcal{M} a$. However, a non minimal $a \in \mathcal{M}^\iota$ may not have a *predecessor*, *i.e.* a greatest $b <_\mathcal{M} a$.[2] The predecessor axiom ensures that every non-minimal individual has a predecessor.

Unboundedness. The axiom of Unboundedness is a kind of infinity axiom. Given a structure $\mathcal{M}$, we say that $U \in \mathcal{M}^o$ is *unbounded in* $\mathcal{M}$ if for all $a \in \mathcal{M}$ there is some $b \in U$ such that $a <_\mathcal{M} b$. If $<_\mathcal{M}$ is strict and transitive, then U must be infinite. Note however that the converse does not hold, even for models of MSO^ω.

Remark 2.9 (Non-Standard Models of MSO^ω). A model $\mathcal{M}$ of MSO^ω can be *non-standard* (*i.e.* non-isomorphic to the standard model N) for two reasons: (i) because its predicate domain $\mathcal{M}^o$ is different from $\mathcal{P}(\mathcal{M}^\iota)$ or (ii) because its individual domain is not isomorphic to $\mathbb{N}$. Let us discuss these two points in view of Theorem 2.8.

(i) It is well-known that if $\mathcal{M}$ is full (*i.e.* $\mathcal{M}^o = \mathcal{P}(\mathcal{M}^\iota)$), then $\mathcal{M}^\iota$ is isomorphic to $\mathbb{N}$ (see e.g. [9]). Hence non-standard models $\mathcal{M}$ have $\mathcal{M}^o \subsetneq \mathcal{P}(\mathcal{M}^\iota)$.

(ii) Thanks to the Löwenheim-Skolem Theorem (see e.g. [1]), we can always assume that an MSO^ω-model $\mathcal{M}$ has a countable individual domain $\mathcal{M}^\iota$. However, the order structure of $\mathcal{M}$ can be very different from that of $\mathbb{N}$. For instance, if $\mathcal{M}$ is a non-standard model of second-order Peano's arithmetic, then it is also a model of MSO^ω. But $\mathcal{M}$ is also a non-standard model of First-Order Peano's Arithmetic, and it is well-known (see e.g. [1]) that its order type is that of: $\mathbb{N}$ followed by $\mathbb{Q}$ copies of $\mathbb{Z}$. In particular, segments of the form $[a, b) = \{c \in \mathcal{M}^\iota \mid a \leq_\mathcal{M} c <_\mathcal{M} b\}$ may be infinite.

[1] The monadic theory of $\mathbb{R}$ is undecidable (see [7] for references).

[2] Besides completeness w.r.t. N, recall that the monadic theory of the ordinal ω_2 is independent from ZFC (see [10]).

3 The Ehrenfeucht-Fraïssé Method

We present the notions on the Ehrenfeucht-Fraïssé method that we will need. They are mostly variations on those used in [5]. See [4] for a standard reference.

For the remaining of the paper, we fix enumerations of the individual and predicate variables. Let $\mathcal{V}_\iota = \{x_1, \dots, x_p, \dots\}$ and $\mathcal{V}_o = \{X_1, \dots, X_q, \dots\}$. We say that ϕ is a *p-q-formula* if $\mathrm{FV}(\phi) \subseteq \{x_1, \dots, x_p, X_1, \dots, X_q\}$.

Unlike the rest of the paper, the results discussed in this section are insensitive on whether we are dealing with Henkin structures, general models, or second-order version thereof. For convenience, we will only consider Henkin structures which are *not necessarily second-order*. In this context, two formulas ϕ and ψ are *logically equivalent* if $(\phi \longleftrightarrow \psi)$ is valid in all such structures.

3.1 Logical Equivalence Up to Bounded Quantifier Depth

The first step is to classify formulas according to their quantifier-depth.

Definition 3.1 (Quantifier-Depth). *The* quantifier depth $\mathrm{qd}(\phi)$ *of a formula ϕ is defined by induction on ϕ as follows:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathrm{qd}(Xx) & := & 0 & \mathrm{qd}(\exists x\, \phi) & := & \mathrm{qd}(\phi) + 1 \\
\mathrm{qd}(x < y) & := & 0 & \mathrm{qd}(\exists X\, \phi) & := & \mathrm{qd}(\phi) + 1 \\
\mathrm{qd}(\neg\phi) & := & \mathrm{qd}(\phi) & \mathrm{qd}(\phi \vee \psi) & := & \max(\mathrm{qd}(\phi), \mathrm{qd}(\psi))
\end{array}$$

We let $\Lambda_n^{p,q}$ be the set of p-q-formulas of q.d. $\leq n$ and write Λ_n for $\Lambda_n^{0,0}$.

A remarkable property of languages without function symbols, such as the language of MSO, is the following standard observation (see e.g. [4]).

Lemma 3.2 (Finiteness Lemma). *Up to logical equivalence, there are only finitely many p-q-formulas of quantifier depth $\leq n$.*

Recall that logical equivalence is defined as validity of equivalence in all (possibly non second-order) structures. Requiring instead validity of equivalence in *all second-order structures* has no impact on finiteness: This amounts to add the comprehension axiom scheme, and adding axioms can only *reduce* the number of equivalence classes.

3.2 Structures with Parameters

A *structure with parameters* is a structure $\mathcal{M}$ together with $a_1, \dots, a_p \in \mathcal{M}^\iota$ and $A_1, \dots, A_q \in \mathcal{M}^o$. We write $\overline{a}$ for a finite sequence of individuals of length $|\overline{a}|$, and similarly for $\overline{A}$. If $|\overline{a}| = p$ and $|\overline{A}| = q$ then we say that $(\mathcal{M}, \overline{a}, \overline{A})$ is a *p-q-structure.*

If ϕ is a p-q-formula, we write $(\mathcal{M}, \overline{a}, \overline{A}) \models \phi$ for $\mathcal{M} \models \phi[\overline{a}/\overline{x}][\overline{A}/\overline{X}]$. Two p-q-structures $(\mathcal{M}, \overline{a}, \overline{A})$ and $(\mathcal{N}, \overline{b}, \overline{B})$ are *n-equivalent* (notation $\equiv_n^{p,q}$) if they satisfy the same p-q-formulas of q.d. $\leq n$. We write $\equiv_n$ instead of $\equiv_n^{p,q}$ when p, q are clear from the context. The Finiteness Lemma allows to characterize the n-equivalence class of a p-q-structure by a single p-q-formula:

Corollary 3.3. *For all $n \in \mathbb{N}$ and all p-q-structure $(\mathcal{M}, \overline{a}, \overline{A})$, there is a formula $\phi \in \Lambda_n^{p,q}$ such that for all p-q-structure $(\mathcal{N}, \overline{b}, \overline{B})$, we have $(\mathcal{N}, \overline{b}, \overline{B}) \models \phi$ if and only if $(\mathcal{M}, \overline{a}, \overline{B}) \equiv_n (\mathcal{N}, \overline{b}, \overline{B})$. Such a ϕ is an n-*characteristic *of $(\mathcal{M}, \overline{a}, \overline{B})$.*

Moreover, there is a finite *set $\Phi_n^{p,q} \subseteq \Lambda_n^{p,q}$ of n-characteristics which contains an n-characteristic of each p-q-structure.*

3.3 Ehrenfeucht-Fraïssé Games

Ehrenfeucht-Fraïssé games are a convenient characterization of $\equiv_n$-equivalence for languages satisfying the Finiteness Lemma. There are different possible formulations for second-order logic. Our presentation is inspired from [5], which is itself that of [4] adapted to non-full models.

Definition 3.4 (Ehrenfeucht-Fraïssé Games). *Given structures $(\mathcal{M}, \overline{a}, \overline{A})$, $(\mathcal{N}, \overline{b}, \overline{B})$ and $n \in \mathbb{N}$, the* Ehrenfeucht-Fraïssé Game $\mathrm{EF}_n((\mathcal{M}, \overline{a}, \overline{A}), (\mathcal{N}, \overline{b}, \overline{B}))$ *is an n-round game played between two players called "Spoiler" and "Duplicator".*

At each round, Spoiler plays first and chooses either an individual or a predicate in one of the two structures. Duplicator then responds in the other structure by choosing an individual if Spoiler chose an individual or a predicate if Spoiler chose a predicate. After n rounds, Spoiler and Duplicator have build a finite relation

$$\{(a'_1, b'_1), \dots, (a'_p, b'_p), (A'_1, B'_1), \dots, (A'_q, B'_q)\}$$

with $n = p + q$, $\overline{a}' \in \mathcal{M}^\iota$, $\overline{b}' \in \mathcal{N}^\iota$, $\overline{A}' \in \mathcal{M}^o$ and $\overline{B}' \in \mathcal{N}^o$. Then Duplicator wins if and only if $(\mathcal{M}, \overline{aa}', \overline{AA}') \equiv_0 (\mathcal{N}, \overline{bb}', \overline{BB}')$.

Our presentation differs from [5,4] on the following point. In these works, Duplicator wins if the finishing tuple is a finite partial isomorphism between the two structures. In our case, equality is not a quantifier-free formula, and we take a coarser wining condition based on $\equiv_0$-equivalence.

Ehrenfeucht-Fraïssé games characterize $\equiv_n$-equivalence:

Theorem 3.5. *Given two structures $(\mathcal{M}, \overline{a}, \overline{A})$ and $(\mathcal{N}, \overline{b}, \overline{B})$ and $n \in \mathbb{N}$, Duplicator has a wining strategy in* $\mathrm{EF}_n((\mathcal{M}, \overline{a}, \overline{A}), (\mathcal{N}, \overline{b}, \overline{B}))$ *if and only if $(\mathcal{M}, \overline{a}, \overline{A})$ and $(\mathcal{N}, \overline{b}, \overline{B})$ are $\equiv_n$-equivalent.*

4 Finite Sums of Segments

We now discuss how to restrict structures into *segments* that can be concatenated. This will be done for second-order linearly ordered structures with correct equality. The Ehrenfeucht-Fraïssé method gives simple proofs that concatenation of segments preserves $\equiv_n$-equivalence. This leads to a partial sum operation on $\equiv_n$-classes. We follow well-known patterns of Feferman-Vaught Theorems [10,7,5].

4.1 Restrictions and Relativizations

Segments will be obtained from structures by restrictions and relativizations. The *restriction* of a structure $\mathcal{M}$ to some *non-empty* predicate $A \in \mathcal{M}^o$ is the structure $\mathcal{M}\restriction A$ defined as expected: its individual domain is $\mathcal{M}^\iota \cap A$, its predicate domain is $\{B \cap A \mid B \in \mathcal{M}^o\}$ and its relation $<_{\mathcal{M}\restriction A}$ is the restriction of $<_{\mathcal{M}}$ to A: $<_{\mathcal{M}\restriction A} := <_{\mathcal{M}} \cap (A \times A)$. It is convenient to write the individual and predicate domains of $\mathcal{M}\restriction A$ respectively as $\mathcal{M}^\iota\restriction A$ and $\mathcal{M}^o\restriction A$.

Restrictions of Structures with Parameters. We shall also need the less usual restriction of *structures with parameters.* Let $p, q \in \mathbb{N}$. Consider a structure $\mathcal{M}$ with individual parameters $\overline{a} = a_1 \ldots a_p$ and predicate parameters $\overline{A} = A_1 \ldots A_q$. Let $A \in \mathcal{M}^o$ be non-empty and such that $a_1, \ldots, a_p \in A$. We define the *restriction of* $(\mathcal{M}, \overline{a}, \overline{A})$ *to* A to be the structure:

$$(\mathcal{M}, \overline{a}, \overline{A})\restriction A \quad := \quad (\mathcal{M}\restriction A, a_1 \ldots a_p, (A_1 \cap A) \ldots (A_q \cap A))$$

Relativization of Formulas. An analogous operation can be defined on formulas. Let ϕ and φ be two formulas with no free variables in common, and let y be a variable not appearing free in ϕ. The *relativization* of ϕ to $\varphi[y]$, notation $\phi\restriction\varphi[y]$, is defined by induction on ϕ as follows:

$$\begin{array}{rcll}
\phi\restriction\varphi[y] & := & \phi & \text{if } \phi \text{ is atomic} \\
(\phi \vee \psi)\restriction\varphi[y] & := & (\phi\restriction\varphi[y]) \vee (\psi\restriction\varphi[y]) & \\
(\neg\phi)\restriction\varphi[y] & := & \neg(\phi\restriction\varphi[y]) & \\
(\exists X\, \phi)\restriction\varphi[y] & := & \exists X\, (\phi\restriction\varphi[y]) & \text{if } X \notin \mathrm{FV}(\varphi) \\
(\exists x\, \phi)\restriction\varphi[y] & := & \exists x\, (\varphi[x/y] \wedge \phi\restriction\varphi[y]) & \text{if } x \notin \mathrm{FV}(\varphi) \cup \{y\}
\end{array}$$

If (ϕ, ν) is a formula with parameters in a structure $\mathcal{M}$, and if $A \in \mathcal{M}^o$ contains all individual parameters of ϕ, then $(\phi, \nu)\restriction A$ is defined as $((\phi\restriction(Xx)[x]), \nu[A/X])$ where $X, x \notin \mathrm{FV}(\phi, \nu)$.

The Transfer Property. We now check that restriction and relativization are equivalent w.r.t. satisfaction. This in particular implies that restriction preserves the comprehension scheme: $\mathcal{M}\restriction A$ is second-order if $\mathcal{M}$ is second-order.

Proposition 4.1 (Transfer). *Let* $p, q \in \mathbb{N}$ *and* $(\mathcal{M}, \overline{a}, \overline{A})$ *be a* p-q*-structure. Let* φ *be a formula with parameters in* $\mathcal{M}$ *and whose free variables are disjoint from* $\{x_1, \ldots, x_p, X_1, \ldots, X_q\}$*. Given* $x_0 \notin \{x_1, \ldots, x_p\}$*, let* $A \in \mathcal{M}^o$ *be non-empty and such that* $(\mathcal{M}, \overline{a}, \overline{A}) \models \forall x\, (Ax \longleftrightarrow \varphi[x/x_0])$*. Assume that* $\overline{a} \in A$*.*

Let ϕ *be a formula with* $\mathrm{FV}(\phi) \subseteq \{x_1, \ldots, x_p, X_1, \ldots, X_q\}$*. Then we have* $(\mathcal{M}, \overline{a}, \overline{A})\restriction A \models \phi$ *if and only if* $(\mathcal{M}, \overline{a}, \overline{A}) \models \phi\restriction\varphi[x_0]$*.*

4.2 Finite Sums of Segments

A *segment* of a structure $\mathcal{M}$ is a predicate of the form

$$[a, b) \ := \ \{c \in \mathcal{M}^\iota \mid a \leq_{\mathcal{M}} c <_{\mathcal{M}} b\} \qquad \text{where } a <_{\mathcal{M}} b$$

We write $[-, b)$ for $[a, b)$ if $\mathcal{M}$ is linearly ordered with least element a. Two consecutive segments $(\mathcal{M}, \overline{a}, \overline{A})\restriction[a, b)$ and $(\mathcal{M}, \overline{b}, \overline{A})\restriction[b, c)$ can be concatenated to $(\mathcal{M}, \overline{a}\overline{b}, \overline{A})\restriction[a, c)$. Using the Ehrenfeucht-Fraïssé method, it is easy to show that concatenation of segments preserves $\equiv_n$-equivalence.

Similar operations have already been defined for full models (see e.g. [10]) as well as for Henkin models [5]. Our operation differs from [5] in the treatment of predicate parameters: since we only need the concatenation of consecutive segments which are restrictions of the same structure $\mathcal{M}$, we can share the predicate parameters in the two components. This simplifies both the statement and the proof of the Lemma.

Lemma 4.2 (Finite Sums of Segments). *Consider two second-order linearly ordered structures $\mathcal{M}$ and $\mathcal{N}$, both with correct equality. Let $n \in \mathbb{N}$.*

$$\begin{array}{rrcl} \textit{If} & (\mathcal{M}, \overline{a}, \overline{A})\restriction[t_0, t_1) & \equiv_n & (\mathcal{N}, \overline{b}, \overline{B})\restriction[u_0, u_1) \\ \textit{and} & (\mathcal{M}, \overline{a}', \overline{A})\restriction[t_1, t_2) & \equiv_n & (\mathcal{N}, \overline{b}', \overline{B})\restriction[u_1, u_2) \\ \textit{then} & (\mathcal{M}, \overline{a}\overline{a}', \overline{A})\restriction[t_0, t_2) & \equiv_n & (\mathcal{N}, \overline{b}\overline{b}', \overline{B})\restriction[u_0, u_2). \end{array}$$

5 Completeness of MSO^ω w.r.t. the Standard Model

In this section, we present a proof of Theorem 2.8. We use an infinitary version of the Finite Sums Lemma which is discussed in Section 6.

We actually prove the following formulation of Theorem 2.8:

Theorem 5.1. *Let $\mathcal{M}$ be a model of MSO^ω. For all $n \in \mathbb{N}$ we have $\mathcal{M} \equiv_n \mathsf{N}$.*

Doets' Lemma. Our way to Theorem 5.1 starts from the simple but crucial observation that bounded segments of models of MSO^ω are $\equiv_n$-equivalent to finite linear orders. To our knowledge, this is due to K. Doets [3] for the Π^1_1-case (first-order logic with universal prenex quantification on predicates). Recall that a bounded segment of an arbitrary model of MSO^ω may not be finite (see Remark 2.9).

In our context, a *finite linear order* is a structure of the form $\mathsf{N}\restriction[m_0, m_1)$ with $m_0 < m_1 \in \mathbb{N}$. Note that if $m_1 - m_0 = k_1 - k_0$ (where $m_0 < m_1$ and $k_0 < k_1$), then $\mathsf{N}\restriction[m_0, m_1) \equiv_n \mathsf{N}\restriction[k_0, k_1)$ for all $n \in \mathbb{N}$.

Lemma 5.2 (Doets' Lemma). *Let $\mathcal{M}$ be a model of MSO^ω and $n \in \mathbb{N}$. For all $a <_{\mathcal{M}} b$, there is a finite linear order $\mathcal{L}$ such that $\mathcal{M}\restriction[a, b) \equiv_n \mathcal{L}$.*

Ramseyan Factorizations. Let $\mathcal{M}$ be a model of MSO^ω. In order to obtain $\mathcal{M} \equiv_n \mathsf{N}$ from Doets' Lemma 5.2, we would like to perform a kind of infinite sum of the $(\mathcal{M}\restriction[a, b))_{a <_{\mathcal{M}} b}$. We rely on a weak form of Ramsey's theorem which is similar to the usual Ramseyan factorizations of infinite words discussed e.g. in [8].

Recall from Corollary 3.3 that if $\mathcal{M}$ is a linearly ordered second-order structure with correct equality, then for all $n \in \mathbb{N}$ and all $a <_{\mathcal{M}} b$, there is a $\phi \in \Phi^{0,0}_n$ such that $\mathcal{M} \models \phi\restriction[a, b)$. We say that $\mathcal{M}$ has Ramseyan factorizations if there is

$\phi \in \Phi_n^{0,0}$ and an unbounded $U \in \mathcal{M}^o$ which is homogeneous for ϕ. We actually need a slightly stronger statement involving formulas with predicate parameters.

Given a structure $\mathcal{M}$ and a predicate $U \in \mathcal{M}^o$, we let $[U]^2 \subseteq \mathcal{M}^\iota \times \mathcal{M}^\iota$ be the set of pairs $(a, b) \in U \times U$ such that $a <_\mathcal{M} b$.

Theorem 5.3 (Ramseyan Factorizations). *Let $\mathcal{M}$ be a model of MSO^ω and let $n, q \in \mathbb{N}$. Given $A_1, \ldots, A_q \in \mathcal{M}^o$ and an unbounded $U \in \mathcal{M}^o$, there is an unbounded predicate $V \subseteq U$ and a $\phi \in \Phi_n^{0,q}$ such that for all $(a, b) \in [V]^2$ we have $(\mathcal{M}, , \overline{A}) \models \phi{\restriction}[a, b)$.*

Theorem 5.3 follows from Theorem I.1.c.3 of [11]. See also the full version of this paper. Together with Doets' Lemma, we obtain:

Corollary 5.4. *Let $\mathcal{M}$ be a model of MSO^ω and $n \in \mathbb{N}$. There is an unbounded $U \in \mathcal{M}^o$ and a finite linear order $\mathcal{L}$ such that for all $(a, b) \in [U]^2$ we have $\mathcal{M}{\restriction}[a, b) \equiv_n \mathcal{L}$.*

Infinite Fusion. Let $\mathcal{M}$ be a model of MSO^ω and $n \in \mathbb{N}$. Using Corollary 5.4 and Doets' Lemma 5.2 we arrive at the following point: There are unbounded $U \in \mathcal{M}^o$ and $V \in \mathcal{P}(\mathbb{N})$ together with $u \in U$ and $v \in V$ such that

$$\mathcal{M}{\restriction}[-, u) \equiv_n \mathsf{N}{\restriction}[-, v)$$

and for all $(u_0, u_1) \in [U]^2$ and all $(v_0, v_1) \in [V]^2$

$$\mathcal{M}{\restriction}[u_0, u_1) \equiv_n \mathsf{N}{\restriction}[v_0, v_1)$$

We can conclude that $\mathcal{M} \equiv_n \mathsf{N}$ from these assumptions thanks to the Infinite Fusion Lemma 6.2. We state and prove it in Section 6, and this will achieve the proof of Theorem 5.1.

6 The Infinite Fusion Lemma

In this section, we state and prove the Infinite Fusion Lemma. Besides Ramseyan factorizations (already discussed in Section 5), we shall also use a weak form of the axiom of choice which is called *Splicing* in [11].

6.1 Splicing

We discuss the Splicing Theorem of [11] and one of its corollary that we actually use in the Infinite Fusion Lemma.

Let $\mathcal{M}$ be a model of MSO^ω and $U \in \mathcal{M}^o$. Individuals $a, b \in \mathcal{M}^\iota$ are *consecutive in U* if $a, b \in U$, $a <_\mathcal{M} b$ and there is no $c \in U$ such that $a <_\mathcal{M} c <_\mathcal{M} b$.

The Splicing Theorem is the following: Given a formula $(\exists X \phi)$ with predicate parameters in $\mathcal{M}$, if for all a, b consecutive in U we have $\mathcal{M} \models \exists X \phi{\restriction}[a, b)$, then there is a predicate $A \in \mathcal{M}^o$ such that for all a, b consecutive in U we have $\mathcal{M} \models \phi[A/X]{\restriction}[a, b)$. This is Theorem I.5.b.1 of [11].

For the Infinite Fusion Lemma, we shall use a variant of Splicing that we call *Idempotent Splicing*. The main difference is that we need to obtain a predicate $A \in \mathcal{M}^o$ which is correct for *all* $(a, b) \in [U]^2$, and not just the *consecutive* ones. On the other hand, we only need it for those $\equiv^q_n$-characteristics which moreover define an idempotent coloring.

Proposition 6.1 (Idempotent Splicing). *Let $\mathcal{M}$ be a model of MSO^ω and let $n, q \in \mathbb{N}$.*

Given and $A_1, \ldots, A_q \in \mathcal{M}^o$, let $\phi \in \Phi^{0,(q+1)}_n$ and $U \in \mathcal{M}^o$ be such that

(i) $(\mathcal{M}, , \overline{A}) \models \exists X\, \phi[X/X_{q+1}]\restriction[a, b)$ for all $(a, b) \in [U]^2$, and
(ii) there is a second-order linearly ordered 0-$(q + 1)$-structure with correct equality $(\mathcal{N}, , \overline{B}B)$ and $b_0 <_{\mathcal{N}} b_1 <_{\mathcal{N}} b_3$ such that ϕ holds in $(\mathcal{N}, , \overline{B}B)\restriction[b_0, b_2)$, $(\mathcal{N}, , \overline{B}B)\restriction[b_0, b_1)$ and $(\mathcal{N}, , \overline{B}B)\restriction[b_1, b_2)$.

Then there is a predicate $A \in \mathcal{M}^o$ such that for all $(a, b) \in [U]^2$ we have $(\mathcal{M}, , \overline{A}A) \models \phi\restriction[a, b)$.

In Proposition 6.1 above, condition (i) is actually the premise of the Splicing Theorem. Condition (ii) intuitively says that ϕ defines an idempotent coloring. We give more details on Splicing in the full version of the paper.

6.2 Infinite Fusion

As usual with the Ehrenfeucht-Fraïssé method, we perform an induction on the quantifier depth of formulas. This to consider structures with parameters.

Lemma 6.2 (Infinite Fusion). *Let $\mathcal{M}$ and $\mathcal{N}$ be models of MSO^ω. Let $n \in \mathbb{N}$.*

Let $U \in \mathcal{M}^o$ and $V \in \mathcal{N}^o$ be unbounded, and assume that their respective least elements u and v are not the least elements of respectively $\mathcal{M}^\iota$ and $\mathcal{N}^\iota$.

Let $\overline{a} \in \mathcal{M}^\iota\restriction[-, u)$, $\overline{b} \in \mathcal{N}^\iota\restriction[-, v)$ both of length $p \in \mathbb{N}$ and $\overline{A} \in \mathcal{M}^o$, $\overline{B} \in \mathcal{N}^o$ both of length $q \in \mathbb{N}$. Assume that

$$(\mathcal{M}, \overline{a}, \overline{A})\restriction[-, u) \equiv_n (\mathcal{N}, \overline{b}, \overline{B})\restriction[-, v)$$

and that for all $(u_0, u_1) \in [U]^2$ and all $(v_0, v_1) \in [V]^2$ we have

$$(\mathcal{M}, , \overline{A})\restriction[u_0, u_1) \equiv_n (\mathcal{N}, , \overline{B})\restriction[v_0, v_1)$$

Then $(\mathcal{M}, \overline{a}, \overline{A}) \equiv_n (\mathcal{N}, \overline{b}, \overline{B})$.

Using Lemma 6.2, we can achieve the proof of Theorem 5.1 as follows. Let $\mathcal{M}$ be a model of MSO^ω and let $n \in \mathbb{N}$. By Corollary 5.4 there is a finite linear order $\mathcal{L}$ and an unbounded predicate $U \in \mathcal{M}^o$ such that $\mathcal{M}\restriction[a, b) \equiv_n \mathcal{L}$ for all $(a, b) \in [U]^2$. Since $\mathcal{M}$ is a second-order linearly ordered structure with correct equality, we can moreover assume that the least element u of U is not the least element of $\mathcal{M}$. By Doets' Lemma 5.2, the segment $\mathcal{M}[-, u)$ is $\equiv_n$-equivalent to $\mathsf{N}\restriction[-, v)$ for some $v > 0$. We thus obtain an unbounded set $V \in \mathcal{P}(\mathbb{N})$ with least element v, and such that $\mathcal{M}\restriction[-, u) \equiv_n \mathsf{N}\restriction[-, v)$ and $\mathcal{M}\restriction[u_0, u_1) \equiv_n \mathsf{N}\restriction[v_0, v_1)$ for all $(u_0, u_1) \in [U]^2$ and all $(v_0, v_1) \in [V]^2$. We conclude $\mathcal{M} \equiv_n \mathsf{N}$ by Lemma 6.2.

The rest of this section is devoted to the proof of Lemma 6.2. We reason by (external) induction on $n \in \mathbb{N}$.

Base Case ($n = 0$). We just have to show that $(\mathcal{M}, \overline{a}, \overline{A})$ and $(\mathcal{N}, \overline{b}, \overline{B})$ agree on atomic formulas ϕ with individual variables in $\{x_1, \dots, x_p\}$ and predicate variables in $\{X_1, \dots, X_q\}$. We only detail the case of $x_i < x_j$, that of $X_i x_j$ being similar.

Since $a_i, a_j <_{\mathcal{M}} u$, the formula $(x_i < x_j)$ holds in $(\mathcal{M}, \overline{a}, \overline{A})$ if and only if it holds in $(\mathcal{M}, \overline{a}, \overline{A}){\restriction}[-, u)$. The same holds for $\mathcal{N}$, and we are done since $(\mathcal{M}, \overline{a}, \overline{A}){\restriction}[-, u) \equiv_0 (\mathcal{N}, \overline{b}, \overline{B}){\restriction}[-, v)$.

Inductive Step. We now consider the inductive step: we show the property for $n+1$ assuming it for n. Using Theorem 3.5, we consider the different possible moves of Spoiler, and then build the answer of Duplicator.

Spoiler plays an individual, say $a \in \mathcal{M}^\iota$. Since U is unbounded, there is $u' \in U$ strictly greater than a. Also using the unboundedness of V, let $v' \in V$ be strictly greater than v.

We have $(\mathcal{M}, \overline{a}, \overline{A}){\restriction}[-, u') \equiv_{n+1} (\mathcal{N}, \overline{b}, \overline{B}){\restriction}[-, v')$ thanks to the Finite Sums Lemma 4.2 applied to the assumptions $(\mathcal{M}, \overline{a}, \overline{A}){\restriction}[-, u) \equiv_{n+1} (\mathcal{N}, \overline{b}, \overline{B}){\restriction}[-, v)$ and $(\mathcal{M}, , \overline{A}){\restriction}[u, u') \equiv_{n+1} (\mathcal{N}, , \overline{B}){\restriction}[v, v')$. Now, by Theorem 3.5 there is some $b \in \mathcal{N}^\iota{\restriction}[-, v')$ such that $(\mathcal{M}, \overline{a}a, \overline{A}){\restriction}[-, u') \equiv_n (\mathcal{N}, \overline{b}b, \overline{B}){\restriction}[-, v')$. The predicates $U' := \{s \in U \mid s \geq_{\mathcal{M}} u'\}$ and $V' := \{t \in V \mid t \geq_{\mathcal{N}} v'\}$ are both unbounded. For all $(u_0, u_1) \in [U']^2$, $(v_0, v_1) \in [V']^2$, we have $(\mathcal{M}, , \overline{A}){\restriction}[u_0, u_1) \equiv_n (\mathcal{N}, , \overline{B}){\restriction}[v_0, v_1)$. Moreover, since $\mathcal{M}$ and $\mathcal{N}$ are both linearly ordered and with correct equality, u' and v' are the least elements of respectively U' and V'. We can thus conclude by induction hypothesis.

Spoiler plays a predicate, say $A \in \mathcal{M}^o$. Since $\mathcal{M}$ has Ramseyan factorizations (Theorem 5.3) we get an unbounded predicate $U' \subseteq U$ and an n-characteristic $\phi \in \Phi_n^{0,q+1}$ such that for all $(u_0, u_1) \in [U']^2$ we have $(\mathcal{M}, , \overline{A}A) \models \phi{\restriction}[u_0, u_1)$, *i.e.* $(\mathcal{M}, , \overline{A}A){\restriction}[u_0, u_1) \models \phi$ thanks to the Transfer Property (Proposition 4.1). Since U' is unbounded and since on the other hand $\mathcal{M}$ is a second-order linearly ordered structure with correct equality, we can assume that U' has a least element u'.

We now claim that for all $(v_0, v_1) \in [V]^2$ we have $(\mathcal{N}, , \overline{B}) \models (\exists X\, \phi){\restriction}[v_0, v_1)$.

– *Proof (of the claim).* Fix $(u_0, u_1) \in [U']^2 \subseteq [U]^2$. For all $(v_0, v_1) \in [V]^2$, since by assumption $(\mathcal{N}, , \overline{B}){\restriction}[v_0, v_1) \equiv_{n+1} (\mathcal{M}, , \overline{A}){\restriction}[u_0, u_1)$, by Theorem 3.5 there is some $B^{v_0,v_1} \in \mathcal{N}^o$ such that $(\mathcal{N}, , \overline{B}B^{v_0,v_1}){\restriction}[v_0, v_1) \equiv_n (\mathcal{M}, , \overline{A}A){\restriction}[u_0, u_1)$, hence $(\mathcal{N}, \overline{B}B^{v_0,v_1}){\restriction}[v_0, v_1) \models \phi$. □

By Proposition 6.1 (Idempotent Splicing), there is a predicate $B \in \mathcal{N}^o$ such that $(\mathcal{N}, , \overline{B}B) \models \phi{\restriction}[v_0, v_1)$ for all $(v_0, v_1) \in [V]^2$. Note that condition (ii) of Proposition 6.1 is satisfied with $(\mathcal{M}, , \overline{A}A)$ and any $u'_0 <_{\mathcal{M}} u'_1 <_{\mathcal{M}} u'_2$ in the unbounded predicate U'.

We now build Duplicator's response to $A \in \mathcal{M}^o$. We have to take care of the initial segment $(\mathcal{M}, , \overline{A}A){\restriction}[-, u')$. Using the unboundedness of V, let $v' \in V$ be strictly greater than v. Reasoning as above, we get $(\mathcal{M}, \overline{a}, \overline{A}){\restriction}[-, u') \equiv_{n+1}$

$(\mathcal{N}, \overline{b}, \overline{B})\restriction[-, v')$ using the Finite Sums Lemma 4.2. Let $V' := \{c \in V \mid v' \leq_{\mathcal{N}} c\}$. By Theorem 3.5, there is some $B' \in \mathcal{N}^o$ such that $(\mathcal{N}, , \overline{B}B')\restriction[-, v') \equiv_n (\mathcal{M}, , \overline{A}A)\restriction[-, u')$. Since $\mathcal{N}$ is second-order, let $B'' := B'\restriction[-, v') \cup B\restriction[v', -)$, where $[v', -) := \{c \in \mathcal{N}^\iota \mid v' \leq_{\mathcal{N}} c\}$. Now, $(\mathcal{N}, \overline{b}, \overline{B}B'')$ (together with V') satisfies the premise of the induction hypothesis and we are done.

7 Conclusion

We gave a model-theoretic proof of Siefkes' completeness result for MSO^ω [11]. It is based on Ramsey's Theorem for additive colorings, with constructions reminiscent from algebraic approaches to ω-rational languages [8]. Further works will begin by clarifying these relationships. An interesting question is the proof-theoretic analysis of MSO^ω. The algebraic approach to parity conditions [8] can be interesting in this perspective. An other direction is the completeness of MSO on infinite trees, and the comparison with Walukiewicz's completeness result for the μ-calculus [12].

References

1. Boolos, G.S., Burgess, J.P., Jeffrey, R.C.: Computability and Logic, 5th edn. Cambridge University Press (2007)
2. Büchi, J.R.: On a Decision Methond in Restricted Second-Order Arithmetic. In: Nagel, E., et al. (eds.) Logic, Methodology and Philosophy of Science. Proc. 1960 Intern. Congr., pp. 1–11. Stanford Univ. Press (1962)
3. Doets, K.: Monadic Π^1_1-Theories of Π^1_1-Properties. Notre Dame Journal of Formal Logic 30(2), 224–240 (1989)
4. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer Monographs in Mathematics. Springer (1999)
5. Gheerbrant, A., ten Cate, B.: Complete Axiomatizations of MSO, $\mathsf{FO}(\mathsf{TC}^1)$ and $\mathsf{FO}(\mathsf{LFP}^1)$ on Finite Trees. In: Artemov, S., Nerode, A. (eds.) LFCS 2009. LNCS, vol. 5407, pp. 180–196. Springer, Heidelberg (2008)
6. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002)
7. Gurevich, Y.: Monadic Second-Order Theories. In: Barwise, J., Feferman, S. (eds.) Model-Theoretical Logics. Perspective in Mathematical Logic, pp. 479–506. Springer (1985)
8. Perrin, D., Pin, J.-É.: Infinite Words: Automata, Semigroups, Logic and Games. Pure and Applied Mathematics. Elsevier (2004)
9. Shapiro, S.: Foundations without Foundationalism: A Case for Second-Order Logic. Oxford University Press (1991)
10. Shelah, S.: The Monadic Theory of Order. The Annals of Mathematics, Second Series 102(3), 379–419 (1975)
11. Siefkes, D.: Decidable Theories I: Büchi's Monadic Second Order Successor Arithmetic. LNM, vol. 120. Springer (1970)
12. Walukiewicz, I.: Completeness of Kozen's Axiomatisation of the Propositional μ-Calculus. Information and Computation 157(1-2), 142–182 (2000)

Compositional Abstraction Techniques for Probabilistic Automata*

Falak Sher and Joost-Pieter Katoen

Software Modeling and Verification Group, RWTH Aachen University, Germany

Abstract. We present aggressive abstraction techniques for probabilistic automata (PA), a state-based model involving discrete probabilistic and nondeterministic branching. Our abstractions yield *abstract* PA in which transitions are typed "possible" or "required"—as in modal transition systems—and have constraint functions as target. The key idea is to focus on identifying common combined-transitions from concrete states and putting them as required ones in the abstract state. We prove the correctness of our abstraction techniques, study their relationship, and show that they are *compositional* w.r.t. parallel composition. We also show the preservation of probabilistic and expected reachability properties for PA.

1 Introduction

Segala's probabilistic automata [16] (PA) are important state-based models for modeling complex systems that involve discrete probabilistic and nondeterministic branching. PA generalize labelled transition systems (LTSs) [14] in that the target of a transition is a probability distribution over the states rather than just a single state. Nondeterminism occurs if several (possibly equally) labelled transitions emanate from a given state. PA are *compositional*—a model of a complex system can be obtained by modeling its components and putting them in parallel, e.g., by synchronizing actions in a CSP-based manner. Thus, PA are an adequate modeling formalism for asynchronously concurrent systems with discrete probabilistic choice such as randomized distributed algorithms. They have been used as semantic model for probabilistic process algebras and the PIOA language.

This paper presents aggressive abstraction techniques for PA that aim at applying abstraction in a component-based,i.e., compositional manner. Rather than focusing on obtaining equivalent behaviour (as for bisimulation), we focus on refinement, a pre-order relation between abstract and concrete models, that assures every *required* transition of an abstract model is mimicked by a combined-transition in the concrete model; and any transition of a concrete model needs to be matched by an *optional* combined-transition in the abstraction. The kernel of our refinement yields a notion of strong bisimulation for PA. It is shown to

* This research is supported by the EU FP7 MoVeS Project (Modeling, Verification and Control of Complex Systems) and the German-Dutch bilateral project ROCKS.

be a pre-congruence w.r.t. parallel composition of abstract PA. This result is a cornerstone for the remaining results in this paper as it facilitates *compositional abstraction*.

As in [8,7], our abstraction techniques yield *abstract* PA in which action-labelled transitions are typed either "possible" or "required" and have constraint functions as targets representing sets of distributions as in constraint Markov chains [5]. As in modal transition systems [9], this distinction is exploited to consider abstract PA as specifications representing a set of concrete PA, its implementations. The key idea of our abstraction techniques is to find out common combined-transitions from concrete states and put them as required transitions in the abstract state. We define two different notions of abstraction and prove their correctness, i.e., show that the concrete models are indeed refinements of the abstract models. Using the fact that our refinement is a pre-congruence for parallel composition, this enables the compositional abstraction of PA.

For the formal analysis and verification of probabilistic systems two quantitative properties are of interest: *probabilistic reachability* (the probability of reaching a set of goal states) and *expected reachability* (the expected number of transitions before reaching a set of goal states). Due to the presence of nondeterminism in PA, it is not always possible to have one unique value for these measures; instead, the *maximum* and the *minimum* values are obtained. As every abstraction technique induces additional nondeterminism, the analysis of the abstract PA gives *lower* and *upper* bounds on the maximum/minimum values for the concrete model. In our abstraction techniques, the distinction of required and possible transitions help achieving this goal of finding bounds on the maximum/minimum values as in [12]. It also helps finding how precise the abstraction is; the better the abstraction is, the tighter the bounds will be. Had there been just one type of transitions in abstract PA, there would have been only single maximum/minimum value for each property. The contributions of the paper are summarized as follows:

– a new notion of a transition relation, called *multi transition*, that is used in defining a refinement relation among PA,
– a new notion of a refinement relation that is a pre-congruence w.r.t. parallel composition,
– a novel compositional abstraction technique, based on common combined-transitions from concrete states, that helps defining *lower* and *upper* bounds on the maximum/minimum values of probabilistic and expected reachability measures, and
– the preservation of probabilistic and expected reachability properties for PA.

Organization. Section 2 sets the ground for this paper and introduces Markov chains, probabilistic automata and abstract PA. Sections 3 and 4 define a satisfaction and a refinement relation respectively. Section 5 discusses two different abstraction techniques whereas section 6 discusses reachability analysis of PA. Section 7 considers parallel composition of abstract PA and presents our compositionality results. Section 8 concludes the paper and provides pointers for future work. The complete paper with proofs of theorems can be found at http://www-i2.informatik.rwth-aachen.de/i2/publications/.

2 Background

Sub-distributions. A function μ is a *sub-distribution* on a finite set S iff $\mu : S \to [0,1]$ and $\sum_{s \in S} \mu(s) \leq 1$. Let $supp(\mu) = \{s \in S \mid \mu(s) > 0\}$ denote the support of μ, and the probability of a set $S' \subseteq S$ w.r.t. μ be given as $\mu(S') = \sum_{s \in S'} \mu(s)$. Let $|\mu| = \mu(S)$ denote the size of the sub-distribution μ. A sub-distribution is a *(full) distribution* iff $|\mu| = 1$. Let $Dist(S)$ denote the set of distributions over S. For $s \in S$, let $\iota_s \in Dist(S)$ denote the *Dirac* distribution, i.e, $\iota_s(s) = 1$. For sub-distributions μ and μ', the point-wise product $(\mu \cdot \mu') : S \times S \to [0,1]$ is given as: $(\mu \cdot \mu')(s, s') = \mu(s) \cdot \mu'(s')$ for $s, s' \in S$. A sub-distribution μ'' can be split into sub-distributions μ and μ', say, represented as $\mu'' = \mu \oplus \mu'$, iff $\mu''(s) = \mu(s) + \mu'(s)$ for $s \in S$. Since $\oplus$ is associative and commutative, we use the notation $\bigoplus$ for finite sums. A sub-distribution is sometimes represented as $\mu = [\![\mu(s) \cdot s \mid s \in supp(\mu)]\!]$, where $[\![$ and $]\!]$ differentiate a set of probabilities from an ordinary set. For $0 \leq c \leq 1$, $c \cdot \mu$ denotes the sub-distribution defined by: $(c \cdot \mu)(s) = c \cdot \mu(s)$.

Constraint Functions. Let φ be an arithmetic expression on variables over S denoting probabilities over S and let $sat(\varphi)$, referred to as the *satisfaction set* of φ, denote the set of distributions that satisfy φ. We call φ a *constraint function* and let $C(S)$ denote the set of constraint functions over S. For simplicity we use φ and $sat(\varphi)$ interchangeably. As every distribution is a constraint function, thus $Dist(S) \subseteq C(S)$. The satisfaction set of the product of two sub-constraint functions φ and φ' is denoted as $sat(\varphi \cdot \varphi') = sat(\varphi) \cdot sat(\varphi') = \{(\mu \cdot \mu') \mid \mu \in sat(\varphi), \mu' \in sat(\varphi')\}$.

Probability Measures and Spaces. Let Ω be a non-empty set and $\mathcal{F} \subseteq 2^{\Omega}$. $\mathcal{F}$ is a σ-field on Ω iff: (1) $\{\} \in \mathcal{F}$ (2) $A \in \mathcal{F} \Rightarrow \Omega \backslash A \in \mathcal{F}$ (3)$A_1, A_2, A_3, ... \in \mathcal{F} \Rightarrow A_1 \cup A_2 \cup A_3 \cup ... \in \mathcal{F}$. The elements of $\mathcal{F}$ are *measurable sets* and $(\Omega, \mathcal{F})$ is a *measurable space*. A function $Prob : \mathcal{F} \to [0,1]$ is a *probability measure* on $(\Omega, \mathcal{F})$ iff $Prob(\Omega) = 1$ and if $A_1, A_2, ...$ are disjoint elements in $\mathcal{F}$, then $Prob(\bigcup_i A_i) = \sum_i Prob(A_i)$. $(\Omega, \mathcal{F}, Prob)$ is called a *measurable space*. For any $\mathcal{A} \subseteq \mathcal{F}$, there exists a unique smallest σ-field that contains $\mathcal{A}$ [2]; and given that $\mathcal{A}$ satisfies certain conditions [2], a *probability measure* defined on $\mathcal{A}$ can be uniquely extended to the σ-field generated from $\mathcal{A}$.

We recap our basic models: *Markov chains* (MC), *probabilistic automata* (PA) [15] and *abstract probabilistic automata* (APA) [8,7]; the first two will act as implementations while the latter as specifications.

Markov Chain (MC). MC extend labelled transition systems (LTS) by annotating transitions with probabilities (instead of actions) that indicate the likelihood of their occurrences.

Definition 1. *A* Markov chain *(MC) is a tuple* $L = (S, \mathbf{P}, AP, \mathbf{V}, s_0)$ *where:*

– *S is a non-empty, finite set of states with initial state s_0,*
– *$\mathbf{P} : S \times S \to [0,1]$ is a transition probability function,*
– *AP is a finite set of valuations, and*
– *$\mathbf{V} : S \to 2^{AP}$ is a state-labeling function.*

The probability of going from state s to s' is given by $\mathbf{P}(s,s')$. We assume that the number of transitions emanating from a state is finite. A *path* of a MC represents an execution of the system it models. It is a non-empty, finite or infinite sequence of states i.e. $\pi = s_0 s_1 s_2 ...$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for $i \geq 0$. For a finite path π_{fin}, let $|\pi_{fin}|$ denote its length (number of transitions) and $last(\pi_{fin})$ its final state. Let $\pi(i)$ denote state s_i of path π; and $\pi^{(i)}$ denote its *prefix* of length i i.e. $\pi(0)...\pi(i)$. The probability that a finite path π_{fin} is executed is given as: $\mathbf{P}(\pi_{fin}) = 1$ if $|\pi_{fin}| = 0$, otherwise $\mathbf{P}(\pi_{fin}) = \mathbf{P}(\pi_{fin}(0), \pi_{fin}(1)) \cdot ... \cdot \mathbf{P}(\pi_{fin}(n-1), \pi_{fin}(n))$, where $|\pi_{fin}| = n$. Finally, let $Path_{fin}(s)$ and $Path(s)$ denote all finite and infinite paths emanating from state s respectively. In the rest of the paper, $L = (S, \mathbf{P}, AP, \mathbf{V}, s_0)$ is assumed to be a MC.

Let a probability measure space $(Paths(s), \mathcal{F}_s, Prob_s)$ over the (infinite) paths emanating from state s be defined as follows. Let the *cylinder set* for finite path $\pi_{fin} \in Paths_{fin}(s)$ be $cyl(\pi_{fin}) = \{\pi \in Paths(s) \mid \pi_{fin}$ is a prefix of $\pi\}$. The probability of the cylinder set $cyl(\pi_{fin})$ is defined to be that of π_{fin}, i.e. $\mathbf{P}(cyl(\pi_{fin})) = \mathbf{P}(\pi_{fin})$. Let $Cyl(s) = \{cyl(\pi_{fin}) \mid \pi_{fin} \in Paths_{fin}(s)\}$ be the set of all cylinder sets defined on finite paths emanating from s; and let $\mathcal{F}_s$ be the smallest σ-field on $Paths(s)$ such that $Cyl(s) \subseteq \mathcal{F}_s$. $\mathbf{P}$ can be uniquely extended for $\mathcal{F}_s$ as $Prob_s$, thus, completing the definition of the probability measure space $(Paths(s), \mathcal{F}_s, Prob_s)$. Based on this construction, we can now define two measures for every MC, *probabilistic reachability* and *expected reachability*, that form the basis of probabilistic model checking [6,3].

Probabilistic reachability refers to the probability of eventually reaching a state in L satisfying an element of $T \subseteq 2^{AP}$ starting from a given state s. Expected reachability refers to the expected number of transitions before reaching a state satisfying an element of T. Let $Paths_{fin}(s,T) = \{\pi_{fin} \in Paths_{fin}(s) \mid \exists i : \mathbf{V}(\pi(i)) \in T \wedge \forall j < i : \mathbf{V}(\pi(j)) \notin T\}$. Formally,

Definition 2. *For MC L, let $s \in S$ and $T \subseteq 2^{AP}$. Then, the reachability probability of a state satisfying an element in T from s is:*

$$p(L,s,T) = \sum_{\pi_{fin} \in Paths_{fin}(s,T)} Prob_s(cyl(\pi_{fin}))$$

and the expected reachability is: if $p(L,s,T) < 1$, then $e(L,s,T) = \infty$; otherwise

$$e(L,s,T) = \sum_{\pi_{fin} \in Paths_{fin}(s,T)} |\pi_{fin}| \cdot Prob_s(cyl(\pi_{fin})).$$

Probabilistic Automata (PA). PA extend labelled transition systems (LTS) by specifying the target of action-labelled transitions as *distributions over states* instead of single states. Let $U\text{-}Act$ be a countable universe of actions ranged over by a, b, etc; and let $Act(s)$ denote the set of enabled actions from state s.

Definition 3. *A* Probabilistic automaton *(PA) is a tuple $M = (S, A, \Delta, AP, \mathbf{V}, s_0)$ where:*

- *S is a non-empty, finite set of states with initial state s_0,*
- *$A \subseteq U\text{-}Act$ is a set of actions,*
- *$\Delta \subseteq S \times A \times Dist(S)$ is a set of transitions,*
- *AP is a finite set of valuations, and*
- *$\mathbf{V} : S \to 2^{AP}$ is a state-labeling function.*

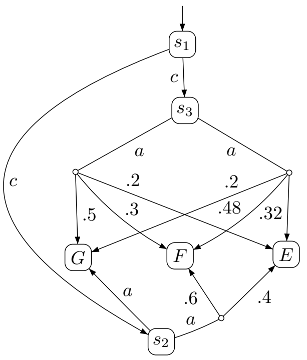


Fig. 1. A PA

We denote $(s, a, \mu) \in \Delta$ by $s \overset{a}{\to} \mu$ and assume that the number of transitions emanating from a state is finite. As $|Act(s)| \geq 1$ for a state $s \in S$, we may have a non-deterministic choice among the enabled actions in s. Therefore, a path in PA represents a particular resolution of non-determinism. Formally, a path from s_0 is given as: $\pi = s_0 \overset{a_0, \mu_0}{\longrightarrow} s_1 \overset{a_1, \mu_1}{\longrightarrow} s_2 ...$ where $a_i \in Act(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geq 0$.

A PA with $|supp(\mu)| = 1$ for every transition $s \overset{a}{\to} \mu$ is an LTS. Similarly, a Markov chain (MC) is a PA in which $|Act(s)| = 1$ for every state $s \in S$. In the rest of the paper, $M = (S, A, \Delta, AP, \mathbf{V}, s_0)$ is assumed to be a PA.

Example 1. Consider the PA given in Fig. 1. Note that the target of the transitions $s_2 \overset{a}{\to} [\![.4E, .6F]\!]$, $s_3 \overset{a}{\to} [\![.2E, .3F, .5G]\!]$ and $s_3 \overset{a}{\to} [\![.32E, .48F, .2G]\!]$ are distributions over states instead of single states. The target of other transitions are Dirac distributions.

To resolve a non-deterministic choice among enabled actions in a state s, a *scheduler* (also known as *policy, strategy or adversary*) is used that makes its decision based on the history of the execution up to that point. In this work, we only consider *deterministic memoryless schedulers* that map a finite path π_{fin} to one pair in $Act(last(\pi_{fin})) \times Dist(S)$. A scheduler κ is called *memoryless* iff $last(\pi_{fin}) = last(\pi'_{fin}) \Rightarrow \kappa(\pi_{fin}) = \kappa(\pi'_{fin})$ for any finite paths π_{fin} and π'_{fin}. A path π under a memoryless scheduler κ is of the form $\pi = s_0 \overset{a_0, \mu_0}{\longrightarrow} s_1 \overset{a_1, \mu_1}{\longrightarrow} s_2 ...$ where $(a_i, \mu_i) = \kappa(s_i)$ for $i \geq 0$. For $s \in S$ and scheduler κ, let $Paths^{\kappa}_{fin}(s)$ and $Paths^{\kappa}(s)$ denote the sets of finite and infinite paths emanating from s under κ. The behaviour of a PA under a scheduler κ is a MC. For PA M and scheduler κ, let M^{κ} be the induced MC. We can construct a measurable space $(Paths^{\kappa}(s_0), \mathcal{F}^{\kappa}, Prob^{\kappa})$ over the (infinite) paths of PA under scheduler κ. To reason about the probabilistic and expected reachability for PA, we consider their *minimum* and *maximum* values under all schedulers κ.

Definition 4. *For PA M, let $T \subseteq 2^{AP}$ and $s \in S$. The minimum and maximum values of probabilistic and expected reachability to a state satisfying an element in T from s are given as:*

- $p^{min}(M, s, T) = \inf_\kappa p(M^\kappa, s, T)$ *and* $p^{max}(M, s, T) = \sup_\kappa p(M^\kappa, s, T)$,
- $e^{min}(M, s, T) = \inf_\kappa e(M^\kappa, s, T)$ *and* $e^{max}(M, s, T) = \sup_\kappa e(M^\kappa, s, T)$.

As proved in [4,1], there exist deterministic memoryless schedulers that optimize the values of probabilistic and expected reachability; and these values can be computed through *value iteration* [4,1,12] which gradually refines them to the required values, or by linear programming.

Abstract PA (APA) [8,7]. Abstract PA act as specifications and aim to represent a set of PA. They extend PA by categorizing transitions/valuations into two types: *required* and *possible* ones. This distinction, adopted from modal transition systems (MTS) [9], is standard for abstract models of labelled transition systems. In addition, the target of transitions are *constraint* functions representing a set of distributions [5]. Altogether, this yields:

Definition 5. *An* abstract PA *is a tuple* $N = (S, A, \Delta_r, \Delta_p, AP, \mathbf{V}_r, \mathbf{V}_p, s_0)$ *with* S *(possibly empty),* A*,* AP *and* s_0 *as before, and:*

- $\Delta_r \subseteq S \times A \times C(S)$ *is a set of* required *transitions,*
- $\Delta_p \subseteq S \times A \times C(S)$ *is a set of* possible *transitions with* $\Delta_r \subseteq \Delta_p$*,*
- $\mathbf{V}_r : S \to 2^{AP}$ *maps a state to a set of* required *valuations, and*
- $\mathbf{V}_p : S \to 2^{AP}$ *maps a state to a set of* possible *valuations with* $\mathbf{V}_r(s) \subseteq \mathbf{V}_p(s)$ *for* $s \in S$*.*

When $(s, a, \varphi) \in \Delta_p$, we write $s \xrightarrow{a}_p \varphi$; and similarly when $(s, a, \mu) \in \Delta_r$, we write $s \xrightarrow{a}_r \varphi$. $s \xrightarrow{a}_p \varphi$ basically represents a set of a-transitions $s \xrightarrow{a}_p \mu$ from s with $\mu \in sat(\varphi)$. In figures, we subscript the actions of required transitions by "r", and those of possible transitions by "p". Each state in an APA is labelled with two sets of atomic propositions: required and possible ones. The required set of atomic propositions is satisfied as a whole by a state in an implementation, where a possible one may be satisfied and that too partially. As the distinction between possible and required propositions is similar to that for transitions, we will ignore propositions in examples. An APA with $\Delta_r = \Delta_p$, $\mathbf{V}_r(s) = \mathbf{V}_p(s)$ for $s \in S$, and $|sat(\varphi)| = 1$ for every required transition $s \to_r \varphi$ is a PA. Similarly, an APA in which for every $s \in S$, $\mathbf{V}_r(s) = \mathbf{V}_p(s)$ and for $s \xrightarrow{a}_p \varphi$, $sat(\varphi)$ is a Dirac distribution, is an MTS [9]. PA and MTS are thus proper sub-models of APA. In fact, PA will be used as implementations and APA as specifications—finite representations of a possibly infinite set of PA. Later on we will see that for the analysis of an APA, a finite set of its implementations suffices. In the rest of the paper, $N = (S, A, \Delta_r, \Delta_p, AP, \mathbf{V}_r, \mathbf{V}_p, s_0)$ is assumed to be an APA.

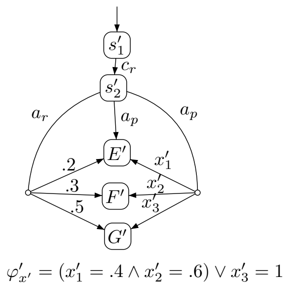


Fig. 2. An APA

Example 2. Fig. 2 represents an APA. Note that state s'_2 has one required transition, $s'_2 \xrightarrow{a}_r [\![.2E', .3F', .5G']\!]$ and two possible transitions, $s'_2 \xrightarrow{a}_p \varphi'_{x'}$ with $sat(\varphi'_{x'}) = \{[\![.4E', .6F']\!], [\![G']\!]\}$ and $s'_2 \xrightarrow{a}_p [\![\iota_{E'}]\!]$.

For the sake of convenience, we introduce the following notations and definitions. The notion of a *combined* transition, denoted $\xrightarrow{a}_C$, arises as a convex combination of a set of transitions with the same label $a \in A$.

Definition 6 (Combined transition). *For PA M, let $s \in S$ and $\mu \in Dist(S)$. We write $s \xrightarrow{a}_C \mu$, if there is a finite indexed set $\{(c_i, \mu_i)\}_{i \in I}$, $c_i \in \mathbb{R}_{\geq 0}$ and $\mu_i \in Dist(S)$, such that $s \xrightarrow{a} \mu_i$ for each $i \in I$, $\sum_{i \in I} c_i = 1$, and $\mu = \bigoplus_{i \in I} c_i \cdot \mu_i$.*

For APA we define $s \xrightarrow{a}_{pC} \varphi$ and $\mu \xrightarrow{a}_{rC} \varphi$ in a similar way for combined possible and required transitions respectively. For a set of distributions $B = \{\mu_1, \mu_2, ..., \mu_n\}$, let $B^C = \{\mu \mid \mu = \bigoplus_{i=1}^n c_i \cdot \mu_i \wedge \sum_i^n c_i = 1\}$. Similarly, we have φ^C for a constraint function φ. Moreover, we write $s \overset{a}{\rightrightarrows}_p \varphi'$, called *multi a-transition* from s, iff $s \xrightarrow{a}_p \varphi_1$, $s \xrightarrow{a}_p \varphi_2$,..., $s \xrightarrow{a}_p \varphi_n$ and $\varphi' = \bigvee_{i=1}^n \varphi_i$. It is important to note that $sat(\bigvee_{s \xrightarrow{a}_{pC} \varphi} \varphi) \subseteq sat(\bigvee_{s \xrightarrow{a}_p \varphi} \varphi)^C$. The multi-transitions extend for PA by default.

Simulation/refinement [10] is a preorder on the state space requiring that whenever state u simulates state s, then u can mimic at least the stepwise behaviour of s. This can be lifted to distributions over states as:

Definition 7 (Simulation). *Let S be a finite, non-empty set of states, and let $\mu, \mu' \in Dist(S)$. For $R \subseteq S \times S$, μ is* simulated *by μ' w.r.t. R, denoted $\mu R \mu'$, iff there exists a weight function $\Delta : S \times S \to [0,1]$ such that for all $u, v \in S$:(1) $\Delta(u,v) > 0 \Rightarrow uRv$, (2) $\sum_{s \in S} \Delta(u,s) = \mu(u)$ and (3) $\sum_{s \in S} \Delta(s,v) = \mu'(v)$.*

3 Satisfaction

A satisfaction relation formally relates a PA, i.e., an implementation, with an APA, i.e., a specification.

Definition 8 (Satisfaction). *Let PA M and APA N' have identical sets of actions A and atomic propositions AP. A relation $R \subseteq S \times S'$ is a* satisfaction *relation if for every sRs':*

1. *$s' \overset{a}{\rightrightarrows}_r \varphi'$ implies $s \overset{a}{\rightrightarrows} \varphi$ and for every $\mu' \in sat(\varphi')$, there exists $\mu \in sat(\varphi)^C$ such that $\mu R \mu'$.*
2. *$s \overset{a}{\rightrightarrows} \varphi$ implies $s' \overset{a}{\rightrightarrows}_p \varphi'$ with $\varphi R \varphi'^C$,*
3. *$\mathbf{V}'_r(s') \subseteq \mathbf{V}(s)$ and $\mathbf{V}(s) \subseteq \mathbf{V}'_p(s')$.*

Here, $\varphi R \varphi'^C$ iff for every $\mu \in sat(\varphi)$, there exists $\mu' \in sat(\varphi')^C$ with $\mu R \mu'$. PA M satisfies *or* implements *APA N', denoted $M \models N'$, iff there exists a satisfaction relation relating s_0 and s'_0. The set of implementations of N' is given as $\wr N' \wr = \{M \mid M \models N'\}$.*

Intuitively, if a state s satisfies a state s', then (1) whenever s' performs a required multi a-transition to a constraint function φ', then s can perform a multi a-transition to a constraint function φ such that every distribution in $sat(\varphi')$ is satisfied by some distribution in $sat(\varphi)^C$; (2) whenever s performs a multi a-transition to a constraint function φ, this can be mimicked by s' by possibly moving to a constraint function φ' such that every $\mu \in sat(\varphi)$ satisfies one distribution in $sat(\varphi')^C$; and (3) s should at least satisfy all required valuations of s' and its all valuations should be derived from that of s'.

Example 3. The PA M in Fig. 1 is an implementation of the AMA N in Fig. 2 as $R = \{(s_1, s_1'), (s_2, s_2'), (s_3, s_2'), (G, G'), (E, E'), (F, F')\}$ is a satisfaction relation. Let us check whether (s_2, s_2') fulfils the conditions of Def. 8. For the required a-transition from s_2' to $[\![.2E', .3F', .5G']\!]$, there is a corresponding multi a-transition from s_2 to a constraint function with satisfaction set $\{[\![.4E, .6F]\!], \iota_G\}$, and $[\![.2E, .3F, .5G]\!]$ is a convex combination of distributions in $\{[\![.4E, .6F]\!], \iota_G\}$. As $[\![.2E, .3F, .5G]\!]$ satisfies $[\![.2E', .3F', .5G']\!]$ ($[\![.2E, .3F, .5G]\!] R [\![.2E', .3F', .5G']\!]$), condition (1) is fulfilled. For the a-transitions from s_2 to ι_G and $[\![.4E, .6F]\!]$, there are corresponding a-transitions from s_2' to $\iota_{G'}$ and $[\![.4E', .6F']\!]$ respectively. Thus, condition (2) is also fulfilled.

Now let us check how the pair (s_3, s_2') fulfils condition (2). Note that there is no simple or combined a-transition from s_2' that satisfies a-transition from s_3 to $[\![.32E, .48F, .2G]\!]$. However, if we consider a multi a-transition from s_2' to $\{[\![.4E, .6F]\!], \iota_E, \iota_G\}$, we get $[\![.32E', .48F', .2G']\!]$, a convex combination of distributions in $\{[\![.4E, .6F]\!], \iota_E, \iota_G\}$, that satisfies $[\![.32E, .48F, .2G]\!]$. Had we just simple transitions in Definition 8, state s_3 would not have been an implementation of state s_2'. (Recall that valuations are not considered in our examples.)

4 Refinement

In this section, we discuss *refinement* that is used to compare APA. Intuitively, a refinement relation compares two APA w.r.t. their sets of implementations. If APA N refines APA N', then we aim at $\wr N \wr \subseteq \wr N' \wr$. Refinement takes a similar view as satisfaction.

Definition 9 (Refinement). *Let APA N and N' have identical sets of actions A and atomic propositions AP. A relation $R \subseteq S \times S'$ is a* refinement *relation if for every sRs':*

1. *$s' \overset{a}{\rightrightarrows}_r \varphi'$ implies $s \overset{a}{\rightrightarrows}_r \varphi$ and for every $\mu' \in sat(\varphi')$, there exists $\mu \in sat(\varphi)^C$ such that $\mu R \mu'$.*
2. *$s \overset{a}{\rightrightarrows}_p \varphi$ implies $s' \overset{a}{\rightrightarrows}_p \varphi'$ with $\varphi R {\varphi'}^C$,*
3. *$\mathbf{V}_r'(s') \subseteq \mathbf{V}(s)$ and $\mathbf{V}(s) \subseteq \mathbf{V}_p'(s')$.*

Let $\preceq$ be the largest refinement relation. N refines *N', denoted $N \preceq N'$, iff there exists a refinement relation relating s_0 and s_0'.*

The above definition is a simple generalization of that of satisfaction. Condition (1) is similar with that of Def. 8 except that the multi a-transition from s now must be a required transition. Condition (2) adds that for every possible multi a-transition from s, there should be a corresponding possible multi a-transition from s'. The similar addition is found in condition (3). Evidently, a satisfaction relation is a special case of a refinement relation.

Definition 10 (Bisimulation). $\sim = \preceq \cap \preceq^{-1}$.

Let us recall the definitions of *Segala's strong probabilistic simulation* and *bisimulation* [15,13] and see how Segala's bisimulation is related to $\sim$.

Definition 11. *Let PA M and M' have identical sets of actions A. A relation $R \subseteq S \times S'$ is a* strong simulation *relation if for every sRs':*

$s \xrightarrow{a} \mu$ implies $s' \xrightarrow{a} \mu'$ with $\mu R \mu'$.

M is simulated by M', denoted $M \sqsubseteq M'$, iff there exists a simulation relation relating s_0 and s'_0.

A relation $R \subseteq S \times S'$ is a strong bisimulation *relation if for every sRs':*

$s \xrightarrow{a} \mu$ implies $s' \xrightarrow{a} \mu'$ with $\mu R \mu'$ and $s' \xrightarrow{a} \mu'$ implies $s \xrightarrow{a} \mu$ with $\mu' R \mu$.

M is strongly bisimilar to M', denoted $N \simeq M'$, iff there exists a strong bisimulation relation relating s_0 and s'_0.

Lemma 1. *$\sim$ coincides with $\simeq$ for any PA.*

Proposition 1. *For PA M,M' and APA N, N':*

– *$N \preceq N'$ implies $\wr N \wr \subseteq \wr N' \wr$.*
– *$M \sim M'$ implies for every N, $M \models N$ iff $M' \models N$.*

5 Abstraction

This section explains two techniques of abstracting an APA that mimics all behaviours of the concrete one. Intuitively, the state space S of APA N is partitioned and each partition is represented by a single state in the abstract APA N'. Formally, let abstraction function $\alpha : S \to S'$ be a surjection and its inverse a concretization function $\gamma : S' \to 2^S$. That is, $\alpha(s)$ is an abstract state of s whereas $\gamma(s')$ is the set of concrete states abstracted by s'. These notions can be lifted to distributions in a simple way: an abstract distribution $\alpha(\mu)$ of μ is given as $\alpha(\mu)(s') = \mu(\gamma(s'))$. α and γ are lifted to sets of states or sets of distributions in a point-wise manner. Thus, $\varphi' = \alpha(\varphi)$ iff $sat(\varphi') = \alpha(sat(\varphi))$.

In the sequel, we assume w.l.o.g. that for APA N, the abstraction function $\alpha : S \to S'$ induces the APA $N' = \alpha(N)$ with the same set of actions A and atomic propositions AP.

Definition 12. (Strong abstraction (SA)) *For APA N, the abstraction function $\alpha : S \to S'$ induces the APA $N' = \alpha(N)$ where for all $s' \in S'$:*

1. *for every $a \in A$ and $\varphi' \in C(S')$,*
   (a) $s' \xrightarrow{a}_r \varphi'$ iff $\forall s \in \gamma(s'), \varphi \in C(S)$: $s \xrightarrow{a}_r \varphi$ and $\varphi' = \alpha(\varphi)$,

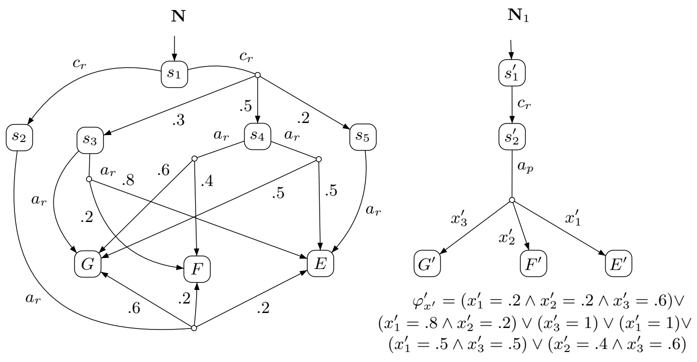

N
N₁
c_r
c_r
s_1
s_2
s_3
s_4
s_5
.3
.5
.2
a_r
a_r
a_r
.8
.6
.4
.5
.5
a_r
.2
a_r
G
F
E
a_r
.6
.2
.2
s_1'
c_r
s_2'
a_p
x_3'
x_2'
x_1'
G'
F'
E'
$\varphi'_{x'} = (x_1' = .2 \wedge x_2' = .2 \wedge x_3' = .6) \vee$
$(x_1' = .8 \wedge x_2' = .2) \vee (x_3' = 1) \vee (x_1' = 1) \vee$
$(x_1' = .5 \wedge x_3' = .5) \vee (x_2' = .4 \wedge x_3' = .6)$

Fig. 3. $N_1 = \alpha(N)$

(b) $s' \xrightarrow{a}_p \varphi'$ *iff* $\exists s \in \gamma(s'), \varphi \in C(S)$*:* $s \xrightarrow{a}_p \varphi$*, and* $\varphi' = \bigvee_{u \in \gamma(s') : u \xrightarrow{a}_p \varphi} \alpha(\varphi)$*.*

2. $\mathbf{V}_r'(s') = \mathbf{V}_r(s)$ *for* $s \in \gamma(s')$*, and* $\mathbf{V}_p'(s') = \bigcup_{s \in \gamma(s')} \mathbf{V}_p(s)$*.*

(1a) Intuitively, if each state in $\gamma(s')$ has a required a-transition, then s' also has a required a-transition and vice versa. Moreover, the abstract behaviour of each concrete transition should be the same as that of the transition from s'. (1b) s' has a possible a-transition iff at least one state in $\gamma(s')$ has a possible a-transition. The accumulative abstract behaviour of all concrete possible transitions is the same as that of the a-transition from s'. (2) The set of required valuations of s' should be the same as that of each concrete state $s \in \gamma(s')$, whereas its set of possible valuations is the union of the sets of possible valuations of concrete states.

Example 4. For $N_1 = \alpha(N)$ (see Fig. 3), $\gamma(s_1') = \{s_1\}$, $\gamma(s_2') = \{s_2, s_3, s_4, s_5\}$, $\gamma(E') = \{E\}$, $\gamma(F') = \{F\}$ and $\gamma(G') = \{G\}$. Note that the abstract behaviour of both required c-transitions from s_1 is the same, and is represented by one required c-transition from s_1'. The required a-transitions from s_2, s_3, s_4 and s_5 behave differently; this is mimicked by the possible a-transition from s_2'.

Consider two states s_1 and s_2 with two required transitions from each: $s_1 \xrightarrow{a}_r \mu_1$, $s_1 \xrightarrow{a}_r \nu_1$ and $s_2 \xrightarrow{a}_r \mu_2$, $s_2 \xrightarrow{a}_r \nu_2$. Let $\alpha(\mu_1) \neq \alpha(\nu_1) \neq \alpha(\mu_2) \neq \alpha(\nu_2)$. It means there is no required a-transition from the abstract state $\alpha(s_1) = \alpha(s_2)$. Let $\varphi_1 = \{\mu_1, \nu_1\}^C$ and $\varphi_2 = \{\mu_2, \nu_2\}^C$ such that $\alpha(\varphi_1) \cap \alpha(\varphi_2) \neq \emptyset$. Then it is possible to have a required a-transition from the abstract state with $\varphi' = \alpha(\varphi_1) \cap \alpha(\varphi_2)$ as the target constraint function. As an example, observe that there are combined a-transitions from s_3 and s_4 (Fig. 3) that have common target distributions e.g. $[\![.2E, .2F, .6G]\!]$. We therefore adapt SA by exploiting these common target distributions of (only) combined required transitions as:

Definition 13. (Common-distribution Abstraction (CDA)) *For APA* N*, the abstraction function* $\alpha_c : S \to S'$ *induces the APA* $N' = \alpha_c(N)$ *where for all* $s' \in S'$*:*

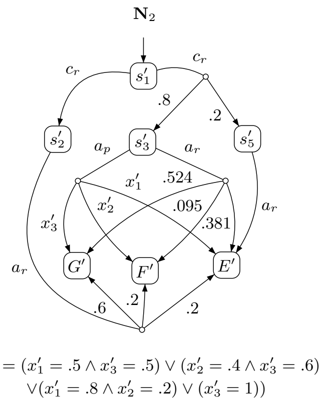

$\mathbf{N}_2$
c_r
c_r
s'_1
.8
.2
s'_2
s'_3
s'_5
a_p
a_r
x'_1
.524
x'_2
.095
x'_3
.381
a_r
a_r
G'
F'
E'
.6
.2
.2
$\varphi'_{x'} = (x'_1 = .5 \wedge x'_3 = .5) \vee (x'_2 = .4 \wedge x'_3 = .6)$
$\vee (x'_1 = .8 \wedge x'_2 = .2) \vee (x'_3 = 1))$

Fig. 4. $N_2 = \alpha_c(N)$

1. *for every* $a \in A$ *and* $\varphi' \in C(S')$,
   (a) $s' \xrightarrow{a}_r \varphi'$ *iff* $\forall s \in \gamma_c(s'), \varphi \in C(S)$: $s \xrightarrow{a}_r \varphi$ *and* $\varphi' = \bigwedge_{u \in \gamma_c(s') : u \overset{a}{\rightrightarrows}_r \varphi} \alpha_c(\varphi)^C$,
   (b) $s' \xrightarrow{a}_p \varphi'$ *iff* $\exists s \in \gamma_c(s'), \varphi \in C(S)$: $s \xrightarrow{a}_p \varphi$ *and* $\varphi' = \bigvee_{u \in \gamma_c(s') : u \xrightarrow{a}_p \varphi} \alpha_c(\varphi)$
2. $\mathbf{V}'_r(s') = \mathbf{V}_r(s)$ *for* $s \in \gamma_c(s')$, *and* $\mathbf{V}'_p(s') = \bigcup_{s \in \gamma_c(s')} \mathbf{V}_p(s)$.

(1a) Like in SA, if each state in $\gamma_c(s')$ has a required a-transition, then s' has a required a-transition and vice versa. However, the common target distributions (after abstraction) among all the combined required a-transitions from the concrete states should be the target of the required a-transition from s'. (1b) and (2) are the same as in SA.

Example 5. Let $N_2 = \alpha_c(N)$ as in Fig. 4 with $\gamma_c(s'_1) = \{s_1\}$, $\gamma_c(s'_2) = \{s_2\}$, $\gamma_c(s'_3) = \{s_3, s_4\}$, $\gamma_c(s'_5) = \{s_5\}$, $\gamma_c(E') = \{E\}$, $\gamma_c(F') = \{F\}$ and $\gamma_c(G') = \{G\}$. The common behaviour of combined a-transitions from s_3 and s_4 is given by the required a-transition from the abstract state s'_3. The possible a-transition from s'_3 represents all a-transitions from s_3 and s_4.

The following results show that the above notions of abstraction yield APA and preserve refinement. Finally, we show that our notions of abstraction are comparable w.r.t. refinement.

Lemma 2. *For APA* N, $\alpha(N)$ *and* $\alpha_c(N)$ *are APA.*

Theorem 1. *For APA* N, $N \preceq \alpha(N)$ *and* $N \preceq \alpha_c(N)$.

Lemma 3. *For APA* N *and* $s \in S$, *if* $\alpha(s) = \alpha_c(s)$ *then* $N \preceq \alpha_c(N) \preceq \alpha(N)$.

6 Reachability

In order to analyse the behaviour of APA, we need to resolve three different types of nondeterminism. The first one is due to possible transitions/valuations that may either be present or absent in an implementation. The second type of nondeterminism is the same as in PA, i.e., the nondeterministic choice among the enabled actions from each state. Like for PA, deterministic schedulers are used for APA to choose among the enabled actions. The third source of nondeterminism is the target constraint function of a transition; one of the distributions in the satisfaction set of the constraint function is nondeterministically chosen. As the satisfaction set of a constraint function may be infinite (and even uncountable), we approximate it by a finite set. We consider the approximation of polynomial constraint functions as they are closed under composition [8].

For $i \in \mathbb{N}^+$, let Θ_i and Φ_i be linear constraint functions in variables over S. Let $\varphi_\iota = \{\iota_s \mid s \in S\}$ be a linear constraint function characterizing Dirac distributions over S, and φ_μ be a linear constraint function characterizing only one distribution, i.e., μ.

Consider a polynomial constraint function ϕ representing a set of distributions over S; therefore, it at least contains a linear constraint $\sum_{s \in S} x_s = 1$ which implies that $\phi^C \subseteq \varphi_\iota^C$. We can now deduce a series of linear constraint functions $\varphi_0 = \varphi_\iota$, $\varphi_1 = \varphi_0 \wedge \Theta_1$, $\varphi_2 = \varphi_1 \wedge \Theta_2$ and so on such that $\phi^C \subseteq \varphi_{i+1}^C \subseteq \varphi_i^C$ for all $i \geq 0$ and $\phi^C = \lim_{i \to \infty} \varphi_i^C$. Every φ_i in the above series over-approximates ϕ, i.e., every μ in ϕ^C also exists in φ_i^C.

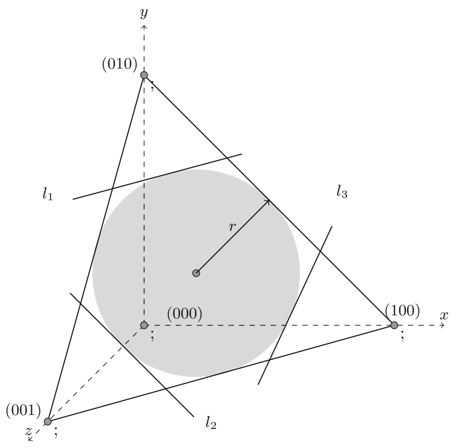
Now we under-approximate ϕ by a linear constraint function. Let $\mu \in \phi$ such that $\varphi_\mu = \mu$ is a linear constraint function. As in the above case there exists a series of constraint functions $\varphi_0 = \varphi_\mu$, $\varphi_1 = \varphi_0 \vee \Phi_1$, $\varphi_2 = \varphi_1 \vee \Phi_2$ and so on such that $\phi^C \supseteq \varphi_{i+1}^C \supseteq \varphi_i^C$ for all $i \geq 0$ and $\phi^C = \lim_{i \to \infty} \varphi_i^C$. Every φ_i in the above series under-approximates ϕ, i.e., every μ in φ_i^C also exists in ϕ^C.

Definition 14. *A constraint-approximating function $\varsigma : C(S) \to C(S)$ over-approximates a polynomial constraint function by a linear one iff for polynomial constraint functions $\phi, \phi_1, \phi_2 \in C(S)$:*

- *$\varsigma(\phi)$ is a linear constraint function and $\phi^C \subseteq \varsigma(\phi)^C$, and*
- *$\phi_1^C \subseteq \phi_2^C \Rightarrow \varsigma(\phi_1)^C \subseteq \varsigma(\phi_2)^C$.*

The inverse function ς^{-1} under-approximates a polynomial constraint function by a linear one, i.e., $\phi^C \supseteq \varsigma^{-1}(\phi)^C$ and $\phi_1^C \supseteq \phi_2^C \Rightarrow \varsigma^{-1}(\phi_1)^C \supseteq \varsigma^{-1}(\phi_2)^C$.

Example 6. Consider a polynomial constraint function $\phi = (x^2 + y^2 + z^2 \leq r^2 \wedge x + y + z = 1)$ representing a set of distributions by a shaded-circular region of radius r within each triangle of Fig. 5 and 6. Let ς_1 and ς_2 be the constraint-approximating functions such that $\varsigma_1(\phi)$ represents the region enclosed among the lines l_1, l_2, l_3 and the sides of the left triangle in Fig. 5; and $\varsigma_2(\phi)$ represents the region among the lines $l_1, \ldots, l_{12}$ in the right triangle. Let $\varsigma_2^{-1}(\phi)$ represent the region among the lines $l_1, \ldots l_6$ in Fig. 6. It is clear that $\varsigma_2(\phi)$ is an over-approximation and $\varsigma_2^{-1}(\phi)$ is an under-approximation of ϕ, i.e., $\varsigma_2^{-1}(\phi) \subseteq \phi \subseteq \varsigma_2(\phi)$. Moreover, $\varsigma_2(\phi)$ gives a better over-approximation of ϕ than $\varsigma_1(\phi)$, i.e., $\phi \subseteq \varsigma_2(\phi) \subseteq \varsigma_1(\phi)$.



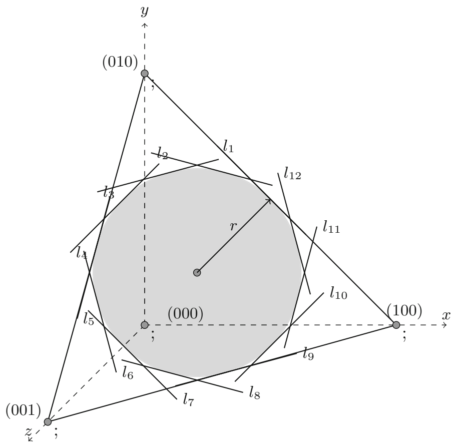


Fig. 5. An example polynomial constraint function (left) and a linear over-approximation (right)

It implies that an APA with polynomial constraint functions can be further abstracted by over/under-approximating its constraint functions with linear ones. The following definition lifts ς from constraint functions to APA.

Definition 15. *For APA N with polynomial constraints, the constraint-approximating function $\varsigma : C(S) \to C(S)$ induces the APA $N' = \varsigma(N)$ with the same set of states S, actions A, atomic propositions AP and state-labeling functions, where for all $s \in S$,*

1. $s \xrightarrow{a}_r \varsigma^{-1}(\phi) \in \Delta'_r$ *iff* $s \overset{a}{\rightrightarrows}_r \phi \in \Delta_r$,
2. $s \xrightarrow{a}_p \varsigma(\phi) \in \Delta'_p$ *iff* $s \overset{a}{\rightrightarrows}_p \phi \in \Delta_p$.

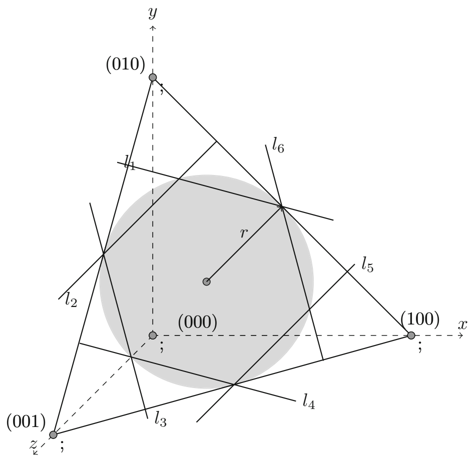


Fig. 6. Linear under-approximation of a polynomial constraint function

(1) As by Definition 14 $\varsigma^{-1}(\phi) \subseteq \phi$, this implies that the set of required a-transitions $s \xrightarrow{a}_r \varsigma^{-1}(\phi)$ from state s in N' is a subset of that of from s in N. (2) However, in the case of possible transitions from s, it is the other way around as $\phi \subseteq \varsigma(\phi)$. This leads to the follow lemma:

Lemma 4. *For APA N, $N \preceq \varsigma(N)$.*

The satisfaction set of a linear constraint function may be infinite, but this can be simplified by just considering its *extreme distributions.*

Definition 16. *The set of* extreme distributions *of a linear constraint function* φ*, denoted* φ_{extr}*, is the smallest finite subset of* φ *such that* $\varphi^C_{extr} = \varphi^C$*.*

The concept of extreme distributions is explained in [11] for interval constraints that can easily be extended for linear constraints. Thus, an APA with linear constraints can be simplified by just considering the extreme distributions of its constraint functions. Let N_{extr} represents an APA in which every transition $s \xrightarrow{a}_p \varphi$ is replaced with concrete transitions $s \xrightarrow{a}_p \mu$ where $\mu \in sat(\varphi_{extr})$. In the following we assume that every APA N with linear constraints is simplified, i.e., $N = N_{extr}$.

Extreme Refinements. Now we consider two refinements of APA N; one in which every possible transition/valuation is converted into a required one and the other in which they are all removed. Let $N^{\uparrow}$ and $N^{\downarrow}$ be the refinements of APA N with $\varDelta^{\uparrow}_r = \varDelta_p$ and $\mathbf{V}^{\uparrow}_r(s) = \mathbf{V}_p(s)$; and $\varDelta^{\downarrow}_p = \varDelta_r$ and $\mathbf{V}^{\downarrow}_p(s) = \mathbf{V}_r(s)$ for $s \in S$ respectively. $N^{\uparrow}$ and $N^{\downarrow}$ are called *extreme refinements* of N.

Consistency. Note that as there do not exist possible transitions/valuations in APA $N^{\downarrow}$, we may have some state, say s, in $N^{\downarrow}$ with $\mathbf{V}^{\downarrow}_p(s) = \emptyset$. Such states are called *inconsistent*. Formally, a state s in $N^{\downarrow}$ is *inconsistent* if either $\mathbf{V}^{\downarrow}_p(s) = \emptyset$ or $s \overset{a}{\rightrightarrows}_r \varphi$ implies either $sat(\varphi) = \emptyset$ or there exists a distribution in $sat(\varphi)$ that assigns a positive mass to at least one state s' with $\mathbf{V}^{\downarrow}_p(s') = \emptyset$. We call such a distribution *inconsistent*. As discussed in [8], such inconsistent states can be iteratively removed from the system by a *pruning operator* β. This process is repeated until there are no more inconsistent states in the system. If the resulting system contains at least one state, we say that it is consistent; otherwise it is inconsistent. (Further details about pruning can be found in [8].)[1ex]

The following lemma tells how Segala's strong probabilistic simulation and bisimulation [15,13] relate implementations of APA N to $\varsigma(N)^{\uparrow}$ and $\varsigma(N)^{\downarrow}$.

Lemma 5. *For APA* N *and for each PA* $M \in \wr N \wr$, $M \sqsubseteq \varsigma(N)^{\uparrow}$ *and* $\varsigma(N)^{\downarrow} \sqsubseteq M$*.*

As the whole behaviour of each implementation M of APA N is derived from N (conditions (2) and (3) of Definition 8), this implies that $M \sqsubseteq \varsigma(N)^{\uparrow}$. As every implementation M of N depicts at least the required behaviour of N (condition (1) of Definition 8), this implies that $\varsigma(N)^{\downarrow} \sqsubseteq \varsigma(N^{\downarrow}) \sqsubseteq M$. Note that $\varsigma(N)^{\downarrow} \neq \varsigma(N^{\downarrow})$. This is because every required transition is also a possible transition and ς over approximates every possible transition in $\varsigma(N^{\downarrow})$.

Moreover, extreme refinements of APA N, and abstractions $\alpha(N)$ and $\alpha_c(N)$ are related by Definition 11 as:

Lemma 6. *For APA* N, $\varsigma(\alpha(N))^{\uparrow} \simeq \varsigma(\alpha_c(N))^{\uparrow}$, $\varsigma(N)^{\downarrow} \sqsubseteq \varsigma(\alpha(N))^{\uparrow}$ *and* $\varsigma(\alpha(N))^{\downarrow} \sqsubseteq \varsigma(\alpha_c(N))^{\downarrow} \sqsubseteq \varsigma(N)^{\downarrow}$*.*

The proof of $\varsigma(\alpha(N))^{\uparrow} \simeq \varsigma(\alpha_c(N))^{\uparrow}$ follows from the fact that the set of possible transitions of $\alpha(N)$ and $\alpha_c(N)$ are the same. As $N \preceq \alpha(N)$, this leads to $\varsigma(N)^{\downarrow} \sqsubseteq \varsigma(\alpha(N))^{\uparrow}$. Moreover, the proof of $\varsigma(\alpha(N))^{\downarrow} \sqsubseteq \varsigma(\alpha_c(N))^{\downarrow} \sqsubseteq \varsigma(N)^{\downarrow}$ follows from the fact that $N \preceq \alpha_c(N) \preceq \alpha(N)$, i.e., the whole required behaviour of $\alpha(N)$ is present in $\alpha_c(N)$ and subsequently in N.

Based on the above lemma we, as a main theorem of this paper, give *lower* and *upper bounds* for maximum/minimum reachability/expected reachability values for PA. In our case as we assume that every PA has only one initial state s_0, therefore, for simplicity we can write $s_0 = \alpha(s_0) = \alpha_c(s_0)$.

Theorem 2. *For PA M and $x \in \{e, p\}$, let $T \subseteq 2^{AP}$, and $M_1 = \varsigma(\alpha(M))^{\uparrow}$, $M_2 = \varsigma(\alpha(M))^{\downarrow}$ and $M_3 = \varsigma(\alpha_c(M))^{\downarrow}$ be PA. Then,*

- $x^{max}(M_2, s_0, T) \leq x^{max}(M_3, s_0, T) \leq x^{max}(M, s_0, T) \leq x^{max}(M_1, s_0, T)$,
- $x^{min}(M_1, s_0, T) \leq x^{min}(M, s_0, T) \leq x^{min}(M_3, s_0, T) \leq x^{min}(M_2, s_0, T)$.

The proof of the above theorem is based on the fact that $M_2 \sqsubseteq M_3 \sqsubseteq M \sqsubseteq M_1$. The bounds given in the above theorem are dependent on the constraint-approximating function ς. The better the ς is, the tighter the bounds will be.

7 Parallel Composition

We define a composition operation that allows to combine two APA. It is defined in a TCSP-like manner, i.e., it is parametrized by a set of actions that need to be performed simultaneously by both APA; other actions can occur autonomously. The following definition is just an extension of the parallel composition definition of APA in [8] with multi transitions.

Definition 17. (Parallel composition) *For APA N and N', the parallel composition w.r.t. synchronization set $\bar{A} \subseteq (A \cap A')$ is given as: $N \|_{\bar{A}} N' = (S \times S', A \cup A', \tilde{\Delta}_r, \tilde{\Delta}_p, AP \cup AP', \tilde{\mathbf{V}}_r, \tilde{\mathbf{V}}_p, (s_0, s'_0))$, where for all $a \in A \cup A'$, $(s, s') \in S \times S'$ and $\tilde{\varphi} \in C(S \times S')$:*

1. *$(s, s') \overset{a}{\rightrightarrows}_r \tilde{\varphi}$ iff one of the following holds:*
   (a) $a \in \bar{A}$, $s \overset{a}{\rightrightarrows}_r \varphi$, $s' \overset{a}{\rightrightarrows}_r \varphi'$ and $\tilde{\varphi} = \varphi \cdot \varphi'$, or
   (b) $a \in A$, $s \overset{a}{\rightrightarrows}_r \varphi$ and $\tilde{\varphi} = \varphi \cdot \iota_{s'}$, or
   (c) $a \in A'$, $s' \overset{a}{\rightrightarrows}_r \varphi'$ and $\tilde{\varphi} = \iota_s \cdot \varphi'$.
2. *$(s, s') \overset{a}{\rightrightarrows}_p \tilde{\varphi}$ iff one of the following holds:*
   (a) $a \in \bar{A}$, $s \overset{a}{\rightrightarrows}_p \varphi$, $s' \overset{a}{\rightrightarrows}_p \varphi'$ and $\tilde{\varphi} = \varphi \cdot \varphi'$, or
   (b) $a \in A$, $s \overset{a}{\rightrightarrows}_p \varphi$ and $\tilde{\varphi} = \varphi \cdot \iota_{s'}$, or
   (c) $a \in A'$, $s' \overset{a}{\rightrightarrows}_p \varphi'$ and $\tilde{\varphi} = \iota_s \cdot \varphi'$.
3. *$\tilde{\mathbf{V}}_r((s, s')) = \mathbf{V}_r(s) \cup \mathbf{V}'_r(s')$ and $\tilde{\mathbf{V}}_p((s, s')) = \mathbf{V}_p(s) \cup \mathbf{V}'_p(s')$.*

In (1a) both s and s' synchronize and perform required multi a-transitions, whereas in (1b) and (1c) they behave independently. Condition (2) considers possible multi transitions from s and s'.

Theorem 3. *For any set $\bar{A}$, $\preceq$ is a pre-congruence w.r.t. $||_{\bar{A}}$.*

The composite APA is exponentially larger in size as compared to the composing ones. This problem could be avoided by applying abstraction prior to composition. The following result shows that the resulting APA is the same as we get by first applying the composition operator to individual APA and then abstracting the monolithic one.

Theorem 4. *For APA N_1 and N_2, synchronization set $\bar{A}$ and abstraction functions α_1, α_2 of the same type: $\alpha_1(N_1) \,||_{\bar{A}}\, \alpha_2(N_2) = (\alpha_1 \times \alpha_2)(N_1 ||_{\bar{A}} N_2)$ up to isomorphism, where $\alpha_1 \times \alpha_2$ is defined as $(\alpha_1 \times \alpha_2)((s, s')) = (\alpha_1(s), \alpha_2(s'))$.*

8 Conclusion

This paper presented novel compositional abstraction techniques for probabilistic automata (PA) as well as a new refinement relation which is pre-congruence w.r.t. parallel composition. The key idea is to find out common combined-transitions from a set of concrete states and put them as required transitions in the abstract state. Moreover, for the analysis and verification of PA, reachability and expected reachability properties are also discussed. We expect the layered composition operator, defined in [17] for PA, can be extended for APA. Future work includes the application of this technique to practical case studies and the development of a counterexample-guided abstraction-refinement framework.

References

1. de Alfaro, L.: Computing Minimum and Maximum Reachability Times in Probabilistic Systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 66–81. Springer, Heidelberg (1999)
2. Ash, R.B., Doléans-Dade, C.A.: Probabilty & Measure Theory, 2nd edn. Academic Press (2000)
3. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distributed Computing 11, 125–155 (1998)
4. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Mathematics of Operations Research 16, 580–595 (1991)
5. Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wasowski, A.: Constraint Markov chains. Theor. Comput. Sci. 412, 4373–4404 (2011)
6. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University, Stanford, CA, USA (1998)
7. Delahaye, B., Katoen, J.-P., Larsen, K., Legay, A., Pedersen, M., Sher, F., Wasowski, A.: New results on abstract probabilistic automata. In: 2011 11th International Conference on Application of Concurrency to System Design (ACSD), pp. 118–127. IEEE CS Press (2011)
8. Delahaye, B., Katoen, J.-P., Larsen, K.G., Legay, A., Pedersen, M.L., Sher, F., Wąsowski, A.: Abstract Probabilistic Automata. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 324–339. Springer, Heidelberg (2011)
9. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal Transition Systems: A Foundation for Three-Valued Program Analysis. In: Sands, D. (ed.) ESOP 2001. LNCS, vol. 2028, pp. 155–169. Springer, Heidelberg (2001)

10. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Logic in Computer Science, pp. 266–277. IEEE CS Press (1991)
11. Katoen, J.-P., Klink, D., Neuhäußer, M.R.: Compositional Abstraction for Stochastic Systems. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 195–211. Springer, Heidelberg (2009)
12. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Formal Methods in System Design 36, 246–280 (2010)
13. Lynch, N.A., Segala, R., Vaandrager, F.W.: Observing branching structure through probabilistic contexts. SIAM J. Comput. 37, 977–1013 (2007)
14. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)
15. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Massachusetts Institute of Technology (1995)
16. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic J. of Computing 2, 250–273 (1995)
17. Swaminathan, M., Katoen, J.-P., Olderog, E.-R.: Layered reasoning for randomized distributed algorithms. Formal Aspects of Computing (to appear, 2012)

Broadcast Abstraction in a Stochastic Calculus for Mobile Networks*

Lei Song and Jens Chr. Godskesen

IT University of Copenhagen, Denmark

Abstract. We introduce a continuous time stochastic broadcast calculus for mobile and wireless networks. The mobility between nodes in a network is modeled by a stochastic *mobility function* which allows to change part of a network topology depending on an exponentially distributed delay and a *network topology constraint*. We allow continuous time stochastic behavior of processes running at network nodes, e.g. in order to be able to model randomized protocols. The introduction of group broadcast and an operator to help avoid *flooding* allows us to define a novel notion of *broadcast abstraction*. Finally, we define a weak bisimulation congruence and apply our theory on a leader election protocol.

1 Introduction

Mobile and wireless networks have become an important part of our life, for instance they have been applied to areas like wireless local area networks, mobile ad-hoc networks, sensor networks, and cellular networks for mobile telephony. Broadcast calculi for this kind of networks have been studied considerably for the last five years, e.g. in [1–5]. A common characteristic for all those calculi is that they deal with mobility and connectivity between nodes abstractly, i.e. a node can move arbitrarily and cause arbitrary change of the network topology, and either a node is connected or disconnected to another node, so none of the calculi address the problem of unreliable links.

In a recent paper [6] we introduced the feature of letting a communication link between two nodes not just be in either 'connected' or 'disconnected' in that we allowed a decoration of connection links with a probability. The meaning being that messages broadcasted along a connection decorated with a probability ρ will be received by that probability. Intuitively this reflects that connection links in wireless networks may not always be reliable. We also enforced restricted mobility by means of a *probabilistic mobility function* saying that a given node with a certain probability may move and thereby change the probability of the connection to another node. The models we obtain are discrete and each network in our calculus in [6] gives rise to a probabilistic automata [7]. A major contribution of this paper is a generalization of the notion of a mobility function. In [6] a mobility function returns the change (the new probability) of just a single connection between two nodes, in this paper we let a mobility function be able to change a number of connections at the same time, i.e. we recognize that mobility of a single node may not just influence the connection to a single neighbor, instead a mobility step

* The research presented in this paper has been supported by MT-LAB, a VKR Center of Excellence for the Modeling of Information Technology.

may change a larger part of the network topology. Moreover, the new kind of mobility functions introduced in this paper makes use of *network topology constraints*. For instance we may specify that the probability for the node l being connected to m must be the same as k being connected to m, i.e. $\rho_{l\longmapsto m} = \rho_{k\longmapsto m}$. Intuitively this may represent that k and l are always within the same distance from m. Another example could be to require that the likelihood one of k and l receiving a broadcast message from m is sufficiently high, we may for instance specify $\rho_{l\longmapsto m} + \rho_{k\longmapsto m} \geq 0.9$, intuitively meaning that m is always sufficiently close to at least one of k and l. We demonstrate the usefulness of topology constraints in Section 5.

Another contribution of this paper is the introduction of stochastically timed behavior for models for mobile and wireless networks, our contribution follows the tradition of having rates for exponential probability distributions, known from say continuous time Markov processes, as part of our calculus. A major motivation for this contribution is that we would like to more realistically being able to model mobility of nodes as time dependent stochastic phenomenon, this is obtained by letting a stochastic mobility function return no longer a discrete probability as in [6] but a rate for an exponential probability distribution. Formally we will write $mf(C, C', \phi) = \lambda$ where C is the current (partial) network configuration, C' is the new configuration reached by a mobility step, ϕ is the network topology constraint the transition from C to C' depends on, and the transition occurs with a delay exponentially distributed by λ. Intuitively the rate signifies how fast the network topology will change, i.e. the higher rate the more likely it is that the topology will change fast. Another reason for introducing continuous time stochastic behavior is that many protocols for mobile and wireless networks make use of time dependent randomized back-off techniques. In order to be able to model such protocols we introduce, in the style of Interactive Markov Chains [8], a prefix construct λ for processes such that we may write e.g. $A = p + \lambda \cdot A$ meaning that A may behave as p or it may after some delay exponentially distributed by λ back off and iterate its behavior. This back-off style encoding is utilized in our model of a leader election protocol for mobile and wireless networks defined in Section 5. By the introduction of the continuous time stochastic behavior it turns out that the semantics of our calculus is a combination of discrete and continuous time probability, non-determinism, and concurrency and thus gives rise to a Markov Automaton (MA) [9]. In [10] a related stochastic restricted broadcast process theory is introduced to model and analyze mobile ad hoc network protocols. Their stochastic model is in PEPA style [11] where the duration of each action is exponentially distributed. After resolving non-determinism a continuous-time Markov chain is derived for each network. Differently our stochastic model is in Interactive Markov Chain [8] style where the rate is used to specify the delay rather than the duration of each action.

A third contribution is that we allow for two novel operators as part of our calculus. To the best of our knowledge these two operators have not before been considered in calculi for mobile and wireless systems. In many broadcast protocols it is quite common for a node to broadcast messages just to a limited number of nodes and hence not to all nodes in the network; to accommodate this feature we introduce a *group broadcast* prefix in our calculus denoted by $\langle x \rhd L\rangle$ where x is the message to be broadcasted and L is the set of intended receivers of x. The other new operator is a kind of a low level protocol that is often used in many wireless broadcast protocols, it is meant to deal with the problem of *flooding*. Flooding occurs when the same message is broadcasted over and over again in the execution of a protocol, but where it is sufficient to have received and dealt with the message just once. Flooding may e.g. occur in a protocol if a node is naively supposed to forward all requests for being part of a protocol, a node receiving similar requests for participating in the same execution of the protocol from multiple neighbors will then forward each of these requests to its neighbors although forwarding just one of these identical requests would ideally be sufficient. The operator is defined by introducing a memory M for each node, formally we write $\lfloor p \rfloor_l^M$ for a node with the processes p running a location l and with memory M. Intuitively the semantics is that whenever the node receives a broadcast message x it is first checked whether x belongs to M, if it does x is discarded and p will remain unchanged, otherwise x is added to M and p is updated accordingly. In this short version of our theory flooding avoidance input is the only broadcast input dealt with.

A fourth and major contribution is that we introduce a novel notion of *broadcast abstraction*. We abstract from the sender of a broadcast message since two broadcast messages should not be distinguished if they can deliver the same message to the same destinations with the same probability, despite that they may originate at different locations. Due to the introduction of group broadcast we can move even further such that one broadcast message can be simulated by several broadcast messages in a row. Intuitively, if a broadcast message α can deliver x to nodes at locations l and k with probability ρ_1 and ρ_2 respectively, and if we have two broadcast messages β_1 and β_2 such that β_1 can only deliver x to l with probability ρ_1 and β_2 can only deliver x to k with probability ρ_2, then β_1 and β_2 together can simulate α. In general we need to assume that the destinations of β_1 and β_2 are disjoint, since otherwise nodes at joint locations may receive x twice with positive probability which will never happen by performing α. The memory M plays a role here, if a node has received x, it will simply ignore it and stay unchanged whenever it receives x again, thus in this case the destinations of β_1 and β_2 may not necessarily be disjoint. For instance, if α can only deliver x to location l with probability ρ, and β_1 and β_2 can only deliver x to l with probability ρ_1 and ρ_2 respectively, then β_1 and β_2 in sequence can simulate α provided that $1 - (1 - \rho_1) \cdot (1 - \rho_2) = \rho$, where $(1 - \rho_1) \cdot (1 - \rho_2)$ equals the probability of x failing to reach l after both β_1 and β_2.

In summary, the main contribution of this paper is a continuous time stochastic broadcast calculus for wireless networks with a stochastic mobility function depending on topology constraints where group broadcast and flooding avoidance are integrated operators. As illustrated above the two operators facilitate abstraction of broadcast messages where several messages may be simulated by one. The paper is organized as follows: the syntax of the calculus is presented in the next section and in Section 3 we give a labeled transition system semantics of our calculus. In Section 4 a weak bisimulation is defined. We apply our calculus on a leader election protocol [12] in Section 5. Finally we end by a conclusion.

2 The Calculus

We presuppose a countable set $\mathcal{N}$ of names, ranged over by x, y, z and a countable set $\mathcal{L}$ of location names, ranged over by k, l, m, and n. Accordingly K, L, M, and N are used to range over finite subsets of $\mathcal{L}$. We also write l directly for a singleton set $\{l\}$. In addition, we also suppose a finite set of probabilities $\wp$ including 0 and 1 ranged over by $\rho, \rho', \rho_1 \ldots$. We define a *location connectivity set*, ranged over by $\mathbb{L}, \mathbb{K} \ldots$, as a finite set $\{(\rho, l) \mid l \in L, \rho \in \wp\}$. We use $l(\mathbb{L}) = \{l \mid (\rho, l) \in \mathbb{L}\}$ to denote all the locations in $\mathbb{L}$.

Let $\mathcal{P}$ denote the set of the processes which is ranged over by $p, q, r \ldots$, and defined by the following grammar:

$$p, q ::= 0 \mid Act \cdot p \mid p + q \mid [x = y]p, q \mid \nu x p \mid A \text{ where } Act ::= \lambda \mid \langle x \rhd L^* \rangle \mid (x)$$

where 0 is the deadlock process. $Act \cdot p$ means that p is prefixed by Act and will behave as p after Act being performed. Specially, $\lambda \cdot p$ means that p is guarded by a delay which is exponentially distributed with rate $\lambda \in \mathbb{Q}^+$.[1] Let $\langle x \rhd L^* \rangle$ and (x) denote (group) broadcast and reception respectively where L^* is either L or $\mathcal{L}$. We usually write $\langle x \rhd \mathcal{L} \rangle$ as $\langle x \rangle$ for simplicity. If $L^* = L$, then $\langle x \rhd L \rangle$ denotes a group broadcast which can deliver the message x only to nodes at locations in L. $p + q$ denotes nondeterministic choices between p and q. $[x = y]p, q$ is a conditional choice, it will evolve into p if $x = y$, otherwise it evolves into q. $\nu x p$ means that x is bounded in p. $A \in \mathcal{A}$ is a process constant where $\mathcal{A}$ is a set of process identifiers. By defining $A \overset{def}{=} p$, A will behave in the same way as p. The set of networks $\mathcal{N}$ is defined by:

$$E, F ::= 0 \mid \lfloor p \rfloor_l^M \mid \{\mathbb{L} \longmapsto l\} \mid \nu x E \mid E \parallel F$$

where node $\lfloor p \rfloor_l^M$ is a process p at location l with memory M which is used to keep track of all the messages having been received. The parameter M is often omitted if it is not important for the discussion. $\nu x E$ and $E \parallel F$ are restriction and parallel composition respectively which have the standard meaning; $\{\mathbb{L} \longmapsto l\}$ denotes connectivity information, i.e. if $(\rho, k) \in \mathbb{L}$, the node at location k is connected to l and can receive messages from l with probability ρ. Let $\mathcal{CN}$ be the set of *connectivity networks* which only contain connectivity information, that is, $C, C' ::= 0 \mid \{\mathbb{L} \longmapsto l\} \mid C \parallel C'$.

A network distribution is a function $\mathbb{E} : \mathcal{N} \to [0, 1]$ satisfying $|\mathbb{E}| = \sum_{E \in \mathcal{N}} \mathbb{E}(E) \leq 1$. Let $\mathcal{ND}$ denote the set of distributions over $\mathcal{N}$, ranged over by $\mathbb{E}, \mathbb{F}, \mathbb{G} \ldots$. The support of $\mathbb{E}$, $Supp(\mathbb{E}) = \{E \mid \mathbb{E}(E) > 0\}$, is the set of networks in $\mathbb{E}$ with positive probability. Sometimes we also write $\{(\rho_i : E_i) \mid \mathbb{E}(E_i) = \rho_i\}$ to denote $\mathbb{E}$. If $\mathbb{E}(E) = 1$, then $\mathbb{E}$ is the *Dirac* distribution δ_E. Given a real number a, $a \cdot \mathbb{E}$ is the distribution such that $(a \cdot \mathbb{E})(E) = a \cdot \mathbb{E}(E)$ for each $E \in Supp(\mathbb{E})$ if $a \cdot |\mathbb{E}| \leq 1$. Moreover $\mathbb{E} = \mathbb{E}_1 + \mathbb{E}_2$ whenever for each E, $\mathbb{E}(E) = \mathbb{E}_1(E) + \mathbb{E}_2(E)$. Parallel composition of network distributions $\mathbb{E} \parallel \mathbb{F}$ is defined as a distribution such that $(\mathbb{E} \parallel \mathbb{F})(E \parallel F) = \mathbb{E}(E) \cdot \mathbb{F}(F)$. Given an equivalence relation $\mathcal{R}$ on networks, $\mathbb{E} \mathcal{R} \mathbb{F}$ iff $\mathbb{E}(S) = \mathbb{F}(S)$ for each $S \in \mathcal{N}/\mathcal{R}$ where $\mathbb{E}(S) = \sum_{E \in S} \mathbb{E}(E)$.

A substitution $\{y/x\}$ can be applied to a process, network, or network distribution. When applied to a network distribution, it means applying the substitution to each network in the support of the distribution. The set of free and bound names in E, denoted by $fn(E)$ and $bn(E)$ respectively, are defined as expected except that $fn(\lfloor p \rfloor_l^M) = fn(p) \cup M$. Structural congruence of processes and networks, $\equiv$, is the least equivalence relation and congruence closed by α-conversion and the rules in Table 1, which can be extended to distributions as usual. Let $loc(E)$ denote the set of locations located in a

[1] $\mathbb{Q}^+$ is the set of all the positive rational numbers.

Table 1. Structural congruence of processes and networks

$p + 0 \equiv p$	$p + q \equiv q + p$	$\nu x \nu y p \equiv \nu y \nu x p$	$(p + q) + r \equiv p + (q + r)$
$E \parallel 0 \equiv E$	$\nu x \nu y E \equiv \nu y \nu x E$	$\{\emptyset \longmapsto l\} \equiv 0$	$\lfloor \nu x p \rfloor_l^M \equiv \nu x \lfloor p \rfloor_l^M, x \notin M$
$E \parallel F \equiv F \parallel E$	$(E \parallel F) \parallel G \equiv E \parallel (F \parallel G)$	$\nu x E \parallel F \equiv \nu x (E \parallel F), x \notin fn(F)$	
$\lfloor p \rfloor_l^M \equiv \lfloor q \rfloor_l^M, p \equiv q$	$\{\mathbb{L}_1 \longmapsto k\} \parallel \{\mathbb{L}_2 \longmapsto k\} \equiv \{\mathbb{L}_1 \cup \mathbb{L}_2 \longmapsto k\}, l(\mathbb{L}_1) \cap l(\mathbb{L}_2) = \emptyset$		

network, i.e. $loc(0) = \emptyset$, $loc(\lfloor p \rfloor_l) = \{l\}$, $loc(\{\mathbb{L} \longmapsto l\}) = \emptyset$, $loc(\nu x E) = l(E)$, and $loc(E \parallel F) = l(E) \cup l(F)$. Differently, $l(E)$ is used to denote all the location names appearing in E including those in connectivity information. The definition of $l(E)$ coincides with $loc(E)$ except that $l(\{\mathbb{L} \longmapsto l\}) = l(\mathbb{L}) \cup \{l\}$.

We use $\rho_{k \longmapsto l}(E)$ to denote the connection probability from k to l in E. When the requested probability does not occur in E the result is $\theta_{k \longmapsto l}$ which denotes an *unknown probability*, i.e. $\rho_{k \longmapsto l}(E) = \rho$ if $E \equiv \{\{(\rho, k)\} \longmapsto l\} \parallel E'$ for some E', otherwise $\rho_{k \longmapsto l}(E) = \theta_{k \longmapsto l}$. We generalize network distributions to contain unknown probabilities. In the following let $\varrho_1, \varrho_2 ::= \rho \mid \theta_{k \longmapsto l} \mid (1 - \theta_{k \longmapsto l}) \mid \varrho_1 \cdot \varrho_2$ be the *generalized probability* which may contain unknown values. The set of *generalized network distribution*, $\mathcal{GND}$, is defined inductively as follows: i) $\mu \in \mathcal{GND}$ if $\mu \in \mathcal{ND}$; ii) $\mu \in \mathcal{GND}$ if there exists ϱ and $\mu_1, \mu_2 \in \mathcal{GND}$ such that $\mu = \varrho \cdot \mu_1 + (1 - \varrho) \cdot \mu_2$. Without causing any confusion, we also use $\mu, \mu', \mu_1, \ldots$ to range over $\mathcal{GND}$. For a generalized network distribution μ, we may substitute unknown probabilities in μ with known probabilities. In order to do so, we introduce the operator $\circ$ such that $\mu \circ \mathcal{D}_l(E)$ is a distribution equal to μ except that an unknown probability $\theta_{k \longmapsto l}$ in μ has been replaced with the probability ρ if $(\rho, k) \in \mathcal{D}_l(E)$. Formally, $(\mu \circ \mathcal{D}_l(E))(F) = (\mu(F)) \circ \mathcal{D}_l(E)$ for each $F \in Supp(\mu)$ where $\circ$ is overloaded to deal with generalized probabilities such that i) $\varrho \circ \mathcal{D}_l(E) = \rho$ if $\varrho = \rho$; ii) $\theta_{k \longmapsto l} \circ \mathcal{D}_l(E) = \rho$ and $(1 - \theta_{k \longmapsto l}) \circ \mathcal{D}_l(E) = 1 - \rho$ if $(k, \rho) \in \mathcal{D}_l(E)$; iii) $(\varrho_1 \cdot \varrho_2) \circ \mathcal{D}_l(E) = (\varrho_1 \circ \mathcal{D}_l(E)) \cdot (\varrho_2 \circ \mathcal{D}_l(E))$.

As mentioned in the introduction we make use of network topology constraints in order to restrict the mobility of nodes. We define the syntax of *topology constraints* Φ, ranged over by ϕ, as follows: $\phi ::= \rho_{k \longmapsto l} = \rho \mid \phi \wedge \phi \mid \phi \vee \phi$ where $\rho_{k \longmapsto l}$ refers to the variable connection probability from k to l, $\rho \in \wp$, and ϕ evaluates to true and false in the obvious way. The above syntax is simple but expressive. For example we can define constraints such as $\rho_{l \longmapsto k} \geq 0.8$ and $\rho_{l \longmapsto m} + \rho_{l \longmapsto n} = 1$ as follows where $\bowtie \in \{<, >, \leq, \geq\}$:

1. $\rho_{l \longmapsto k} \bowtie \rho = \bigvee_{\rho' \in \wp \wedge \rho' \bowtie \rho} \rho_{l \longmapsto k} = \rho'$,
2. $\rho_{l \longmapsto m} + \rho_{l \longmapsto n} \bowtie \rho = \bigvee_{\rho_1, \rho_2 \in \wp \wedge \rho_1 + \rho_2 \bowtie \rho} (\rho_{l \longmapsto m} = \rho_1 \wedge \rho_{l \longmapsto n} = \rho_2)$.

Given a topology constraint ϕ, define operator $E[\phi]$ to evaluate ϕ under a network E by $E[\phi_1 \bowtie \phi_2] = E[\phi_1] \bowtie E[\phi_2]$ with $\bowtie \in \{\wedge, \vee\}$, $E[\rho_{l \longmapsto k} = \rho] = true$ if $\rho_{l \longmapsto k}(E) = \rho$, otherwise $E[\rho_{l \longmapsto k} = \rho] = false$, and boolean operators are evaluated as usual.

Topology constraints together with connectivity networks is the source for defining continuous time stochastic mobility. A *stochastic mobility function* (SMF) $mf : \mathcal{CN} \times \mathcal{CN} \times \Phi \rightarrow \mathbb{Q}^+$ is a partial function where $mf(C, C', \phi)$ returns the mobility rate from C to C' given the topology constraint ϕ. We assume $mf(C, C, true) = 0$ if the connectivity network C is static, i.e. it cannot evolve into other networks. For simplicity

Table 2. Labeled Transition System of Processes

$$\frac{}{\lambda \cdot p \overset{\lambda}{\twoheadrightarrow} p}\ (\text{MAR}) \qquad \frac{}{\langle x \rhd L^* \rangle \cdot p \xrightarrow{\langle x \rhd L^* \rangle} p}\ (\text{PRE}) \qquad \frac{p \overset{\alpha_p}{\rightsquigarrow} p'}{p + q \overset{\alpha_p}{\rightsquigarrow} p'}\ (\text{SUM}) \qquad \frac{p \overset{\alpha_p}{\rightsquigarrow} p' \quad x = y}{[x = y]p, q \overset{\alpha_p}{\rightsquigarrow} p'}\ (\text{IF})$$

$$\frac{}{(x) \cdot p \xrightarrow{(y)} p\{y/x\}}\ (\text{INP}) \qquad \frac{p \overset{\alpha_p}{\rightsquigarrow} p' \quad x \notin fn(\alpha_p)}{\nu xp \overset{\alpha_p}{\rightsquigarrow} \nu xp'}\ (\text{RES}) \qquad \frac{q \overset{\alpha_p}{\rightsquigarrow} q' \quad x \neq y}{[x = y]p, q \overset{\alpha_p}{\rightsquigarrow} q'}\ (\text{ELSE})$$

$$\frac{q \equiv p \overset{\alpha_p}{\rightsquigarrow} p' \equiv q'}{q \overset{\alpha_p}{\rightsquigarrow} q'}\ (\text{STR}) \qquad \frac{p \overset{\alpha_p}{\rightsquigarrow} p' \quad A \overset{def}{=} p}{A \overset{\alpha_p}{\rightsquigarrow} p'}\ (\text{CON}) \qquad \frac{p \xrightarrow{\langle x \rhd L^* \rangle} p' \quad y \notin fn(\nu xp)}{\nu xp \xrightarrow{\nu y \langle y \rhd L^* \rangle} p'\{y/x\}}\ (\text{bOPEN})$$

we let $mf(C, C', \phi) = \bot$ denote that the mobility rule from C to C' under condition ϕ is undefined. An SMF is *valid* if for each C, C' such that $mf(C, C', \phi) \neq \bot$ for some ϕ, then $\rho_{k \longmapsto l}(C) = \theta_{k \longmapsto l}$ iff $\rho_{k \longmapsto l}(C') = \theta_{k \longmapsto l}$ for all k and l. Intuitively, the condition guarantees that when a mobility step from C to C' happens, it only changes the probability of connectivities in C, we can neither obtain information about connectivities not in C, nor lose connectivities in C. For instance let $C = \{\{(0.5, m), (0.9, n)\} \longmapsto l\}$ and $C' = \{\{(0.8, m)\} \longmapsto l\}$, a mobility rule from C to C' is not valid since the connectivity information of $\rho_{n \longmapsto l}$ is lost in C', similarly a mobility rule from C' to C is not valid either. In the following we will only consider valid SMFs, and we assume that there is a given mf throughout the paper.

Since we have infinitely many connectivity networks, it is not reasonable to always define mobility rules for all of them. Instead we allow an mf to be defined for just finitely many pairs C and C' and topology constraints ϕ. We call those rules *explicit* mobility rules. A connection probability $\rho_{l \longmapsto k}$ has an explicit mobility rule if there exists $mf(C, C', \phi) \neq \bot$ with $\rho_{l \longmapsto k}(C) \neq \theta_{l \longmapsto k}$. For any connection probability $\rho_{l \longmapsto k}$ with no explicit mobility rule we assume it has the implicit mobility rule $mf(\{\{(0, l)\} \longmapsto k\}, \{\{(0, l)\} \longmapsto k\}, true) = 0$, that is l is not and will never be connected to k. The default implicit mobility can be changed without affecting our theory.

The structural congruence closed set of *well-formed* networks $\mathcal{N}_{mf}$ under a given SMF mf is inductively defined as follows:

1. $0 \in \mathcal{N}_{mf}$, $\lfloor p \rfloor_l^M \in \mathcal{N}_{mf}$, and $\nu xE \in \mathcal{N}_{mf}$ if $E \in \mathcal{N}_{mf}$,
2. $E \parallel F \in \mathcal{N}_{mf}$ if $E, F \in \mathcal{N}_{mf}$ with $loc(E) \cap loc(F) = \emptyset$ and there does not exist $l, k \in \mathcal{L}$ such that $\rho_{l \longmapsto k}(E) \neq \theta_{l \longmapsto k}$ and $\rho_{l \longmapsto k}(F) \neq \theta_{l \longmapsto k}$,
3. $C \in \mathcal{N}_{mf}$ if there exists C' and ϕ such that $mf(C, C', \phi) \neq \bot$.

Clause 1 is trivial. Clause 2 means that locations are unique and that connectivity information for a single connection can only appear once, while clause 3 (together with clause 2) requires that the connectivity network part of a network can be divided into subnetworks for each of which mobility must be defined by the given mf.

3 Labeled Transition System

We use $\mathcal{A}_p$ to denote the actions of processes, defined as follows:

$$\alpha_p ::= \nu\tilde{x}\langle x \rhd L^*\rangle \mid (x) \mid \lambda,$$

where $\nu\tilde{x}\langle x \rhd L^*\rangle$ denotes broadcasting the message x to nodes at locations in L^*. Whenever x is bounded $\tilde{x} = \{x\}$, otherwise $\tilde{x} = \emptyset$. The (x) means that the process can receive a (group) broadcast message. λ denotes a Markovian action with specified rate. The semantics of processes is given in Table 2 where all the rules are standard, and $\rightsquigarrow\ = (\rightarrow \cup \twoheadrightarrow)$ with $\twoheadrightarrow$ denoting Markovian transitions.

We use $\mathcal{A}$ to denote the actions of networks defined as follows:

$$\alpha ::= \nu\tilde{x}\langle x \rhd L^*, \mathbb{L}\rangle @l \mid (x@L^*, \mathbb{L}) \lhd l \mid \lambda \mid \phi : \lambda \mid \tau.$$

Different from process actions, for the actions of networks connectivity information is attached to any broadcast and reception action. Action $\nu\tilde{x}\langle x \rhd L^*, \mathbb{L}\rangle @l$ denotes that the node at location l can broadcast the message x to the node at location $k \in L^*$ with probability ρ if $(\rho, k) \in \mathbb{L}$. Accordingly $(x@L^*, \mathbb{L}) \lhd l$ means that the node at location $k \in L^*$ can receive the message x from location l with probability ρ if $(\rho, k) \in \mathbb{L}$. The $\phi : \lambda$ is a novel action named *condition guarded Markovian action*. This action is used to model topology constrained mobility where mobility is triggered only when certain conditions are satisfied. The τ and λ are standard.

The semantics of networks is given in Table 3 with $\rightsquigarrow$ (as in Table 2) being the union of $\twoheadrightarrow$ and $\rightarrow$. For readability we also write δ_E directly as E. The behavior of a node is determined by the process in it, but the actions of a node may be enriched with connectivity information as well as the source and destination respectively if the action is either a broadcast or a reception action. Rule (nREC1) says that when a process in a node located at a location can perform a reception, then the node can also perform a reception action, similarly for (nBRD) which deals with broadcast actions. In (nBRD) we remove l from L^* since a node cannot receive messages broadcasted from itself. Note in (nBRD) and (nREC1) there is no connectivity information, so the corresponding connectivity sets in the labels are empty, and furthermore in (nREC1) the node at location l is able to receive a message from location k with unknown probability denoted by $\theta_{l\longmapsto k}$, this is the only rule where unknown probability is added. Two parallel networks E and F communicate by broadcast as shown by (nSYN) where one network can perform a broadcast action while the other one can perform a reception action, similarly in (nREC2) we let two networks in parallel can perform a reception action simultaneously. As shown in both (nSYN) and (nREC2), we require that the destinations of the broadcast and reception actions of the two participants coincide.

Rules (nBRD), (nREC1), (nREC2), and (nSYN) deal with group broadcast when $L^* = L$. Different from broadcast where the broadcast messages can be received by any node in any location, group broadcast has specified destinations, nodes at locations which are not in the set of the destinations will simply ignore the messages and stay unchanged, this is taken care of by rule (nIGN). As explained in the introduction we introduce a low level protocol taking care of flooding assuming that a message can only be received by a node at most once. The parameter M at a node is used to keep track of the messages already received, so only if the coming message is not already in M, it will be dealt with, otherwise it will be simply ignored as explained in rules (nREC1) and (nIGN). On the other hand, if process p at location l cannot perform a reception, it will simply ignore all the coming messages, and stay unchanged as illustrated by (nIGN).

In (nSYN) E and F may obtain new connectivity information $\mathbb{L}$ and $\mathbb{K}$ from each other and update the unknown probabilities that might appear in distributions $\mathbb{E}$ and $\mathbb{F}$ via the operator $\circ$, similarly for (nREC2). In (nSYN) K is the union of the set of locations in F, $loc(F)$, and the set of locations in $\mathbb{K}$ which are not connected to l, $\mathcal{Z}(\mathbb{K}) = \{k \mid (0, k) \in \mathbb{K}\}$. We remove K from the resulting action where $\mathbb{L} \setminus K = \{(\rho, k) \in \mathbb{L} \mid k \notin K\}$. It makes sense to remove $\mathcal{Z}(\mathbb{K})$ from the destination set of the broadcast action since nodes at locations $\mathcal{Z}(\mathbb{K})$ will for sure not receive messages from l. Also we remove locations $loc(F)$ since all the nodes at locations $loc(F)$ in F after the transition will receive the broadcast message.

If an action is not broadcast or reception, networks can execute in parallel without synchronization, this gives the rule (nPAR). Network $\{\mathbb{K} \longmapsto l\}$ only contains connectivity information about l, it can reveal its connectivity information by performing a (group) reception which is shown by (nCONN); it can also, in order to synchronize on broadcast from locations not being l, perform a (group) reception whose source location is different from l with empty connectivity information as illustrated by the rule (nLOS). A broadcast with empty destination has no impact to the outside of the emitting network, therefore it should be seen as an internal action τ which is shown by (nLOC). Due to (nSYN) and (nREC2), (nLOC) can only happen at top level. Rule (nMOB) allows a connectivity network to evolve into another according to the mobility rule defined by the given *mf* carrying out a condition guarded Markovian action $\phi : \lambda$. By (nTRU) if ϕ is evaluated to true, then $\phi : \lambda$ will become a Markovian transition λ. Note in (nREC1) and (nIGN), we require that $l \neq k$ which means that a process at location l cannot receive messages broadcasted from the same location. The rules (nOPEN), (nRES), (nMAR), and (nSTR) are standard.

In our calculus we allow continuous delay, probabilistic choice, and non-deterministic choice, as result each network corresponds to a Markov Automaton [9] which is the integration of probabilistic automata [7] with interactive Markov chains [8]. As usual we assume networks to be free of divergence with probability 1, see e.g. [7], in order to avoid an unrealistic situation where infinitely many actions can happen in finite time. For instance network $E \overset{def}{=} \lfloor A \rfloor_l \parallel \lfloor \lambda \cdot 0 \rfloor_k$ with $A \overset{def}{=} \langle x \rangle \cdot A$ is not free of divergence, since E can perform broadcast from l for infinitely many times, and thus blocks the Markovian transition at k for ever.

4 Weak Bisimulation

In this section we provide a weak bisimulation congruence for our calculus. We say that a network E is *stable*, written $E\downarrow$, if $E \overset{\tau}{\nrightarrow}$ and $E \overset{\langle x \triangleright L^*, \mathbb{L} \rangle @ l}{\nrightarrow}$. Note that broadcasts are considered to be immediate and take no time, since they are non-blocking and will be triggered immediately. Accordingly, a network distribution $\mathbb{E}$ is stable, written $\mathbb{E}\downarrow$, iff $E\downarrow$ for each $E \in Supp(\mathbb{E})$.

Table 3. Labeled Transition System of Networks

$$\frac{p \xrightarrow{(x)} p' \quad (l \in L^* \wedge x \notin M \wedge k \neq l)}{\lfloor p \rfloor_l^M \xrightarrow{(x@L^*,\emptyset)\lhd k} \{(\theta_{l\longmapsto k} : \lfloor p' \rfloor_l^{M\cup\{x\}}), (1-\theta_{l\longmapsto k} : \lfloor p \rfloor_l^M)\}} \text{ (nREC1)}$$

$$\frac{E \xrightarrow{\nu\tilde{y}\langle y \rhd L^*, \mathbb{L}\rangle @l} \mathbb{E} \quad F \xrightarrow{(y@L^*,\mathbb{K})\lhd l} \mathbb{F} \quad \tilde{y} \cap fn(F) = \emptyset \quad K = loc(F) \cup \mathcal{Z}(\mathbb{K})}{E \parallel F \xrightarrow{\nu\tilde{y}\langle y \rhd (L^* \backslash K), (\mathbb{L}\cup\mathbb{K})\backslash K\rangle @l} (\mathbb{E} \circ \mathcal{D}(F)) \parallel (\mathbb{F} \circ \mathcal{D}(E))} \text{ (nSYN)}$$

$$\frac{E \xrightarrow{\langle x \rhd L^*, \mathbb{L}\rangle @l} \mathbb{E} \quad y \notin fn(\nu x E)}{\nu x E \xrightarrow{\nu y\langle y \rhd L^*, \mathbb{L}\rangle @l} \mathbb{E}\{y/x\}} \text{ (nOPEN)} \qquad \frac{E \xrightarrow{(x@L^*,\mathbb{L})\lhd l} \mathbb{E} \quad F \xrightarrow{(x@L^*,\mathbb{K})\lhd l} \mathbb{F}}{E \parallel F \xrightarrow{(x@L^*,\mathbb{L}\cup\mathbb{K})\lhd l} (\mathbb{E} \circ \mathcal{D}(F)) \parallel (\mathbb{F} \circ \mathcal{D}(E))} \text{ (nREC2)}$$

$$\frac{p \overset{\lambda}{\twoheadrightarrow} p'}{\lfloor p \rfloor_l^M \overset{\lambda}{\twoheadrightarrow} \lfloor p' \rfloor_l^M} \text{ (nMAR)} \qquad \frac{p \xrightarrow{\nu\tilde{x}\langle x \rhd L^*\rangle} p'}{\lfloor p \rfloor_l^M \xrightarrow{\nu\tilde{x}\langle x \rhd (L^* \backslash l), \emptyset\rangle @l} \lfloor p' \rfloor_l^M} \text{ (nBRD)} \qquad \frac{E \xrightarrow{\phi:\lambda} \mathbb{E} \quad E[\phi] = true}{E \overset{\lambda}{\twoheadrightarrow} \mathbb{E}} \text{ (nTRU)}$$

$$\frac{F \equiv E \overset{\alpha}{\rightsquigarrow} \mathbb{E} \equiv \mathbb{F}}{F \overset{\alpha}{\rightsquigarrow} \mathbb{F}} \text{ (nSTR)} \qquad \frac{E \overset{\alpha}{\rightsquigarrow} \mathbb{E} \quad \alpha \in \{\lambda, \phi : \lambda\}}{E \parallel F \overset{\alpha}{\rightsquigarrow} \mathbb{E} \parallel F} \text{ (nPAR)} \qquad \frac{E \xrightarrow{\nu\tilde{y}\langle x \rhd \emptyset, \mathbb{L}\rangle @l} \mathbb{E}}{E \xrightarrow{\tau} \mathbb{E}} \text{ (nLOC)}$$

$$\frac{E \overset{\alpha}{\rightsquigarrow} \mathbb{E} \quad x \notin fn(\alpha)}{\nu x E \overset{\alpha}{\rightsquigarrow} \nu x \mathbb{E}} \text{ (nRES)} \qquad \frac{mf(C, C', \phi) = \lambda}{C \xrightarrow{\phi:\lambda} C'} \text{ (nMOB)} \qquad \frac{k \neq l \wedge (l \notin L^* \vee x \in M \vee p \overset{(x)}{\nrightarrow})}{\lfloor p \rfloor_l^M \xrightarrow{(x@L^*,\emptyset)\lhd k} \lfloor p \rfloor_l^M} \text{ (nIGN)}$$

$$\frac{l \neq k}{\{\mathbb{K} \longmapsto k\} \xrightarrow{(x@L^*,\emptyset)\lhd l} \{\mathbb{K} \longmapsto k\}} \text{ (nLOS)} \qquad \frac{}{\{\mathbb{K} \longmapsto l\} \xrightarrow{(x@L^*,\mathbb{K})\lhd l} \{\mathbb{K} \longmapsto l\}} \text{ (nCONN)}$$

In order to evaluate the exit rate of a network we, similar with [8], define the function $\gamma : \mathcal{N}_{mf} \times 2^{\mathcal{N}_{mf}} \mapsto \mathbb{Q}^+$ which returns the exit rate from a given network to a set of networks via Markovian transitions. The formal definition is as follows where $\{\!|\,|\!\}$ denotes multiset: $\gamma(E, S) = \sum\{\!|\lambda \cdot \mathbb{E}(S) \mid E \overset{\lambda}{\twoheadrightarrow} \mathbb{E}|\!\}$. Due to a *race condition* [8, 11] among Markovian transitions they will compete in order to be executed first, this gives us the following natural transitions. Let $E \xrightarrow{\lambda} \mathbb{E}$ if $E\downarrow$ where $\lambda = \gamma(E, \mathcal{N}_{mf})$ and $\mathbb{E}(F) = \frac{\gamma(E,\{F\})}{\lambda}$ for all F. Refer to the following example for an illustration of race condition.

Example 1. Let $E = \lfloor \lambda_1 \cdot p + \lambda_2 \cdot q \rfloor_l$. It is easy to see that E has two Markovian transitions according to Table 2 and 3: $E \overset{\lambda_1}{\twoheadrightarrow} \lfloor p \rfloor_l$ and $E \overset{\lambda_2}{\twoheadrightarrow} \lfloor q \rfloor_l$. The exit rate of E is equal to $\lambda = \lambda_1 + \lambda_2$, and moreover the two Markovian transitions will compete to be executed first. Due to the race condition, the first transition will be executed with probability $\frac{\lambda_1}{\lambda}$, while the second one will be executed with probability $\frac{\lambda_2}{\lambda}$, i.e. $E \xrightarrow{\lambda} \{\frac{\lambda_1}{\lambda} : \lfloor p \rfloor_l, \frac{\lambda_2}{\lambda} : \lfloor q \rfloor_l\}$.

We use $E \overset{\alpha}{\Longrightarrow} \mathbb{E}$ to denote that a distribution $\mathbb{E}$ is reached through a sequence of steps which are internal except one being equal to α. Formally $\overset{\alpha}{\Longrightarrow}$ is the least relation such that, $E \overset{\alpha}{\Longrightarrow} \mathbb{E}$ iff

1. $\alpha = \tau$ and $\mathbb{E} = \delta_E$, or
2. there exists a step $E \xrightarrow{\beta} \mathbb{E}'$ such that $\mathbb{E} = \sum_{E' \in Supp(\mathbb{E}')} \mathbb{E}'(E') \cdot \mathbb{E}_{E'}$, where $E' \overset{\tau}{\Longrightarrow} \mathbb{E}_{E'}$ if $\beta = \alpha$, otherwise $E' \overset{\alpha}{\Longrightarrow} \mathbb{E}_{E'}$ and $\beta = \tau$.

As in [7] we also define the combined transition $\overset{\alpha}{\Longrightarrow}_c$ such that: $E \overset{\alpha}{\Longrightarrow}_c \mathbb{E}$ iff there exists $\{E \overset{\alpha}{\Longrightarrow} \mathbb{E}_i\}_{1\le i\le n}$ and $\{w_i\}_{1\le i\le n}$ with $\sum_{1\le i\le n} w_i = 1$ and $\sum_{1\le i\le n} w_i \cdot \mathbb{E}_i = \mathbb{E}$.

As an abstraction we disregard the emitter of a broadcast message and allow to equate $\nu\tilde{x}\langle x \rhd L^*, \mathbb{L}\rangle@l$ and $\nu\tilde{x}\langle x \rhd L^*, \mathbb{L}\rangle@k$ indicating that in a wireless broadcast setting the sender of a message is not important, that is only the message (and the probability by which it is received), since the receiver of a message may not precisely know whom is the actual emitter of the message. To further enforce what we in the Introduction called broadcast abstraction we will also allow that a broadcast can be simulated by several broadcast messages. In order to do so we define the combination of two broadcast actions such that

$$\nu\tilde{x}\langle x \rhd L_1, \mathbb{L}_1\rangle@l_1 \otimes \nu\tilde{x}\langle x \rhd L_2, \mathbb{L}_2\rangle@l_2 = \nu\tilde{x}\langle x \rhd L, \mathbb{L}\rangle@l$$

where $L = L_1 \cup L_2$, l is any location name, and $\mathbb{L} = \mathbb{M}_1 \cup \mathbb{M}_2$ with

$$\mathbb{M}_1 = \{(\rho, k) \in \mathbb{L}_1 \mid k \in L_1 \setminus L_2\} \cup \{(\rho, k) \in \mathbb{L}_2 \mid k \in L_2 \setminus L_1\},$$

$$\mathbb{M}_2 = \{(1 - (1-\rho_1)\cdot(1-\rho_2), k) \mid k \in L_1 \cap L_2 \wedge (\rho_1, k) \in \mathbb{L}_1 \wedge (\rho_2, k) \in \mathbb{L}_2\}.$$

Intuitively, the resulting combination of two actions has the same effects as the original two. There are three cases to consider. If a location k is only in L_1, then the probability for location k receiving the broadcast message x will not be changed by $\nu\tilde{x}\langle x \rhd L_2, \mathbb{L}_2\rangle@l_2$, similarly for locations only in L_2. For a location k appearing in both L_1 and L_2, the probability for k not receiving x is equal to $(1-\rho_1)\cdot(1-\rho_2)$ if $(\rho_1, k) \in \mathbb{L}_1$ and $(\rho_2, k) \in \mathbb{L}_2$, as a result the probability for a node at location k receiving x is equal to $1 - (1-\rho_1)\cdot(1-\rho_2)$. Obviously, $\otimes$ is associative and commutative. We extend the broadcast transitions in the following way: $E \xrightarrow{\langle x \rhd L^*, \mathbb{L}\rangle@l} \mathbb{E}$ iff $E \overset{\alpha_1}{\Longrightarrow}\overset{\alpha_2}{\Longrightarrow} \ldots \overset{\alpha_n}{\Longrightarrow} \mathbb{E}$ with $\langle x \rhd L^*, \mathbb{L}\rangle@l = (\bigotimes_{1\le i\le n} \alpha_i)$.

According to Table 3 there might occur unknown probabilities during the evolution of networks. Intuitively, to compare two network distributions where unknown probabilities may occur, we consider all the possibilities for substitution of those unknown probabilities by concrete probabilities i.e. two networks are equivalent if they behave equivalently in all possible substitution contexts. In order to do so, we introduce operator $\bullet$ such that $E \bullet C$ denotes a network behaving like E but obtaining new information from C, that is, $E \bullet 0 = E$, $E \bullet (C \parallel C') = (E \bullet C) \bullet C'$, and $E \bullet (\{\{(\rho, k)\} \cup \mathbb{L} \longmapsto l\}) = E \bullet (\{\mathbb{L} \longmapsto l\})$ if $\rho_{k\longmapsto l}(E) \neq \theta_{k\longmapsto l}$, otherwise $E \bullet (\{\{(\rho, k)\} \cup \mathbb{L} \longmapsto l\}) = (E \parallel \{\{(\rho, k)\} \longmapsto l\}) \bullet (\{\mathbb{L} \longmapsto l\})$. Intuitively $\bullet$ is used to supply a network E with auxiliary connection probabilities, information about connections which probability are already known in E will simply be ignored.

In the definition of our bisimulation we make use of the following finite sets of connectivity networks: $\mathcal{CN}_L = \{C \in \mathcal{CN} \mid \forall l, k \in L. \rho_{k\longmapsto l}(C) \neq \theta_{k\longmapsto l}\}$. Intuitively, $\mathcal{CN}_L$ contains all the connectivity networks such that the probability of $\rho_{k\longmapsto l}$ is known for all $l, k \in L$. Below follows the definition of weak bisimulation of networks where we use $C_{E,F,k}$ to range over $\mathcal{CN}_{(l(E)\cup l(F)\cup\{k\})}$, and we let α_k range over all actions including λ except the reception actions from locations l where $l \neq k$.

Definition 1. *An equivalence relation $\mathcal{R} \subseteq \mathcal{N}_{mf} \times \mathcal{N}_{mf}$ is a weak bisimulation iff $E\ \mathcal{R}\ F$ implies that for each k and $C_{E,F,k}$, whenever $E \bullet C_{E,F,k} \xrightarrow{\alpha_k} \mathbb{E}$, there exists $F \bullet C_{E,F,k} \overset{\alpha_k}{\Longrightarrow}_c \mathbb{F}$ such that $\mathbb{E}\ \mathcal{R}\ \mathbb{F}$. Let E and F be weak bisimilar, written as $E \approx_{mf} F$, if there exists a weak bisimulation $\mathcal{R}$ such that $E\ \mathcal{R}\ F$.*

The cases when α_k is τ or λ are standard. When $\alpha_k = (x@L, \mathbb{L}) \lhd k$, any received message must be matched by receiving the same message with the same probabilities from the same sender. Observe that the source of the message cannot appear in $loc(E)$ due to the semantics in Table 3, as a consequence one may prove that $E \approx_{mf} F$ implies $loc(E) = loc(F)$.

Example 2. Given a *mf* such that l and k can always connect to all locations except m with the same probability, and all locations can always connect to l and k with the same probability. Then $\lfloor (x) \cdot \langle x \rangle \rfloor_l \parallel \lfloor 0 \rfloor_k \parallel \lfloor 0 \rfloor_m \approx_{mf} \lfloor (x) \cdot \langle x \rangle \rfloor_k \parallel \lfloor 0 \rfloor_l \parallel \lfloor 0 \rfloor_m$ but since l and k can receive messages from the node at location m with different probabilities $\lfloor (x) \cdot \langle x \rangle \rfloor_l \parallel \lfloor 0 \rfloor_k \not\approx_{mf} \lfloor (x) \cdot \langle x \rangle \rfloor_k \parallel \lfloor 0 \rfloor_l$.

When a network is not stable, then all the Markovian transitions are blocked, and cannot affect the behavior of the network. This is related to the *maximal progress assumption* which is a quite common in time (discrete and continuous) process algebra [8, 13, 14].

Example 3. Consider two networks:$E = \lfloor \langle x \rhd L \rangle \cdot p + \lambda \cdot q \rfloor_l$ and $F = \lfloor \langle x \rhd L \rangle \cdot p \rfloor_l$, since E is not stable due to $E \xrightarrow{\langle x \rhd L, \emptyset \rangle @ l}$, therefore the Markovian transition $E \xrightarrow{\lambda}$ can be omitted, obviously $E \approx_{mf} F$.

When $\alpha_k = \nu\tilde{x}\langle x \rhd L^*, \mathbb{L} \rangle @ l$ any broadcast message x must be matched by a broadcast action containing the same x, and x must be received at the same locations with the same probability, but the emitter need not be the same.

Example 4. Given a *mf* where l is disconnected from k forever, then $\lfloor \langle x \rhd l \rangle \rfloor_k \approx_{mf} \lfloor 0 \rfloor_k$. If $\rho_{l \longmapsto k}$ is not always 0 then $\lfloor \langle x \rhd l \rangle \rfloor_k \not\approx_{mf} \lfloor 0 \rfloor_k$, but if reception at the node at l has no effect then e.g. $\lfloor \langle x \rhd l \rangle \rfloor_k \parallel \lfloor 0 \rfloor_l \approx_{mf} \lfloor 0 \rfloor_k \parallel \lfloor 0 \rfloor_l$.

Additionally when $\alpha_k = \nu\tilde{x}\langle x \rhd L^*, \mathbb{L} \rangle @ l$, we also allow that a broadcast can be simulated by a series of broadcasts whose combination is equivalent to the original broadcast. This relies on the assumption that each message can only be received by a node at most once.

Example 5. Given a *mf* such that location l can receive messages from location k with probability either 1 or 0. Then $\lfloor \langle x \rhd l \rangle \cdot \langle x \rhd l \rangle \rfloor_k \parallel \lfloor p \rfloor_l^M \approx_{mf} \lfloor \langle x \rhd l \rangle \rfloor_k \parallel \lfloor p \rfloor_l^M$ for any p. The reason is that after the process at location k receives the message x, it will remember it, and if it receives the same message for the second time, it will simply ignore it and stay unchanged.

In all cases in Definition 1 we use $C_{E,F,k}$ to eliminate all the possible unknown probabilities during the evolution of both E and F. Observe that unknown probabilities can only appear in derivatives on networks in case of broadcast and reception actions. The reason to include k is because k might be any location not appearing in either E or F, thus when E or F performs a reception from k, an unknown probability $\theta_{l \longmapsto k}$ with

$l \in l(E) \cup l(F)$ may arise. Such an unknown probability may be eliminated by applying any $C_{E,F,k}$. When performing broadcasts the only possible unknown probability in a derivative from E and F is of the form $\theta_{m \longmapsto n}$ with $m, n \in l(E) \cup l(F)$, thus it can also be removed by applying any $C_{E,F,k}$.

Example 6. Suppose a *mf* such that $\rho_{m \longmapsto n}$ is always equal to 0.5 and two networks: $E = \{\{(0.5, m)\} \longmapsto n\}$ and $F = 0$. Without applying a $C_{E,F,k}$, we will conclude that $E \not\approx_{mf} F$ since $E \xrightarrow{(x@L^*,\{(0.5,m)\}) \lhd n} \delta_E$ which cannot be simulated by F. This is against our intuition since we know that $\rho_{m \longmapsto n}$ is always equal to 0.5, thus F should be able to exploit this fact from the given *mf*. By applying any $C_{E,F,k}$ it is easy to check that $E \approx_{mf} F$.

The following theorem shows that the weak bisimulation is a congruence.

Theorem 1. $\approx_{mf}$ *is a congruence.*

The definition of our bisimulation depends on a given SMF *mf*, the more restricted the *mf* the more bisimilar networks we can obtain. For instance, if we consider the extreme case where all the nodes are disconnected from each other all the time, that is, they cannot influence each other's behaviors, we then have $\lfloor p \rfloor_l \approx_{mf} \lfloor q \rfloor_l$ for any p, q.

5 A Leader Election Protocol

We illustrate the application of our calculus by modeling an adaption of the leader election protocol in [12]. Before giving the model we first explain how this protocol works. It is assumed that each node has a unique ID i. A node may regularly initiate an election of a new leader; it will start the process of building a spanning tree by broadcasting a message *Election* to its neighbors and then wait for acknowledgement messages, *Ack*, from its children in the tree. An *Ack* message will contain the information about the node with the highest ID the child has found. When a node j receives an *Election* from another node i, it will set i as its parent and then propagate *Election* to its neighbors and then wait for the acknowledgements *Ack* from its children. In a state waiting for *Ack* messages a node keeps track of the highest ID received before it times out after a certain time limit. When timing out a node (not being the root of the spanning tree) reports the highest ID found to its parent via an *Ack* message and enters a state where it waits to be informed about the new leader found. When the initiator of the run of the protocol times out waiting for *Ack* messages it broadcasts the new leader, i.e. the node with the highest ID found, to its neighbors via the message *Leader*. Notice that due to node mobility a child may disconnect from its parent before it sends the acknowledgement, the time out in this case prevents the parent getting stuck waiting for the acknowledgement forever. Similarly for a node waiting for announcements of a new leader, it will either receive the announcement in time, or it will time out and announce the node with highest ID it has found so far as the new leader.

The state of a node is represented by $Node(i, l, m, p)$ where i is the ID, l is the ID of its leader, m is the maximum ID known in a protocol run, and p is the ID of its parent.

To model this protocol we define three types of messages (names) where I is a finite set of all the possible ID numbers: $\{E\_i \mid i \in I\}$ is the set of *Election* messages, $\{A\_m \mid$

Model 1. The model of the leader election protocol

$$
\begin{array}{lll}
\mathit{Node}(i,l,m,p) & = & \lambda_{init} \cdot \langle E\_i \rhd I \rangle \cdot \mathit{Init}(i,l,m,p) \\
 & + & \sum_{x \neq i}(E\_x) \cdot \langle E\_i \rhd I \rangle \cdot \mathit{waitAck}(i,l,m,x) \\
\mathit{Init}(i,l,m,p) & = & \sum_{x \neq i}(A\_x) \cdot ([x > m]\mathit{Init}(i,l,x,p), \mathit{Init}(i,l,m,p)) \\
 & + & \lambda_{exp} \cdot \langle L\_m \rhd I \rangle \cdot \mathit{Node}(i,m,m,p) \\
\mathit{waitAck}(i,l,m,p) & = & \sum_{x \neq i}(A\_x) \cdot ([x > m]\mathit{waitAck}(i,l,x,p), \mathit{waitAck}(i,l,m,p)) \\
 & + & \lambda_{exp} \cdot \langle A\_m \rhd p \rangle \cdot \mathit{waitLeader}(i,l,m,p) \\
\mathit{waitLeader}(i,l,m,p) & = & \sum_{x \neq i}(L\_x) \cdot \mathit{Node}(i,x,m,p) \\
 & + & \lambda_{par} \cdot \langle L\_m \rhd I \rangle \cdot \mathit{Node}(i,m,m,p)
\end{array}
$$

$m \in I\}$ is the set of *Ack* messages, and $\{L\_l \mid l \in I\}$ is the set of *Leader* messages which announces the elected leader. In [12] the messages in a given election are all assigned a unique index used to distinguish the protocol run from other runs. For simplicity we omit these details in the model of the protocol in this paper.

To make the model more compact we extend the match operator in the following way: $[x > m]p, q$ denotes that the process will evolve into p if $x > m$, otherwise it will evolve into q, this operator can be defined using the standard operators in a straightforward way. The operator $\sum_{x \neq i}(E\_x)$ means that the input only accepts *Election* messages not from i, and ignores all the other messages, the operator can easily be encoded by a sequence of conditional operators prefixed by (x). We introduce similar operators for accepting just one type of protocol messages. The model of the protocol is given in Model 1 where λ_{init} and λ_{exp} denote the rate of initializing a new run of the protocol and the rate of timeout from waiting for the acknowledgements from children respectively. If a node is not involved in any election, it will be at state *Node*. The node with ID i can initialize an election by broadcasting the message E_i to its neighbors, and evolve into *Init*. When the neighbor nodes receive the message E_i, they will join the election and evolve into *waitAck* after forwarding the *Election* message to their neighbors. While at *Init* or *waitAck*, a node will wait for the acknowledgements from its neighbors. In order not to get stuck and wait for the acknowledgements forever, we let each node stop waiting with rate λ_{exp}. When the node at *Init* stops waiting for the acknowledgements, it will announce m, the maximal ID found so far, as the new leader. Differently, when timing out nodes at *waitAck* will send an acknowledgement together with the parameter m to their parents, and then evolve into *waitLeader* waiting for the announcement of the new leader. It may happen that a node will timeout when waiting for the announcement from its parent while at *waitLeader*, in this case it will simply announce m as its leader and terminate the election. Each node at *waitLeader* will timeout with a certain delay by rate λ_{par}.

Next we will show how to define mobility rules for our example. For simplicity we assume that there are four locations in the network: l, k, m, and n where all the nodes are stationary except the node at l. Suppose that nodes at location k and l are always disconnected, thus the move of node at l will not affect the value of $\rho_{k \longmapsto l}$ and $\rho_{l \longmapsto k}$. There are two possible positions Pos_1 and Pos_2 for the node at location l such that when in Pos_1 it will be closer to the node at location m than the node at location n i.e. $\rho_{m \longmapsto l} > \rho_{n \longmapsto l}$ while in Pos_2 we have $\rho_{m \longmapsto l} < \rho_{n \longmapsto l}$. When the node at location l is at Pos_1, it will move to Pos_2 with rate 2, while in Pos_2 it will move to Pos_1 with rate 5. Moreover

Model 2. An simplified model of the leader election protocol

$$\begin{array}{ll}
Node'(i) & = \lambda_{init} \cdot \langle E\_i \rhd I\rangle \cdot Init'(i) + (E\_x) \cdot \langle E\_i \rhd I\rangle \cdot waitAck'(i) \\
Init'(i) & = \lambda_{exp} \cdot \langle L\_i \rhd I\rangle \cdot Node'(i) \\
waitAck'(i) & = \lambda_{exp} \cdot waitLeader'(i) \\
waitLeader'(i) & = (L\_x) \cdot Node'(i) + \lambda_{par} \cdot \langle L\_i \rhd I\rangle \cdot Node'(i)
\end{array}$$

no matter how the node at location l moves, we guarantee that $\rho_{m\longmapsto l} + \rho_{n\longmapsto l} = 1$ as long as $\rho_{m\longmapsto n} = 1$ and $\rho_{n\longmapsto m} = 1$. Since $\rho_{m\longmapsto l}$ and $\rho_{n\longmapsto l}$ may both change when l moves, their mobility rules should be defined together in our SMF. Suppose that $\rho_{m\longmapsto l} = 0.8$ and $\rho_{n\longmapsto l} = 0.2$ when the node at location l moves to Pos_1, and $\rho_{m\longmapsto l} = 0.3$ and $\rho_{n\longmapsto l} = 0.7$ when it is at Pos_2. By letting $mf(C_1, C_2, \phi) = 2$ and $mf(C_2, C_1, \phi) = 5$ we complete the definition of the mobility rules with $C_1 = \{\{(0.8, m), (0.2, n)\} \longmapsto l\}$, $C_2 = \{\{(0.3, m), (0.7, n)\} \longmapsto l\}$, and $\phi = (\rho_{m\longmapsto n} = 1 \wedge \rho_{n\longmapsto m} = 1)$. Note that more complicated rules can be defined, for instance when the condition ϕ does not hold i.e. m and n are not close enough, we can let the $\rho_{m\longmapsto l}$ and $\rho_{n\longmapsto l}$ evolve into other values such that $\rho_{m\longmapsto l} + \rho_{n\longmapsto l} \neq 1$. For simplicity we will omit the details.

It is not hard to see that in this example we use group broadcast often between nodes internally in the network, as a result we can abstract from the concrete execution of the model. Suppose we only care whether each node in a network has a leader or not, then the model can be simplified as Model 2 where the node which initializes the election always chooses itself as the new leader.

In Model 2, the acknowledgement messages $\langle A\_i \rhd I\rangle$ can be abstracted totally, and we can establish that: $\underset{i\in I}{\|} \lfloor Node'(i)\rfloor_i \approx_{mf} \underset{i\in I}{\|} \lfloor Node(i, l, m, p)\rfloor_i$. Intuitively, this equivalence holds because all the group broadcasts will become internal. In Model 2 the group broadcasts dealing with acknowledgements used to find the node with the highest ID are abstracted away, since we do not care about the specific ID of the leader. Essentially in Model 2 the node which initializes the election simply commutes between two states depending on whether it has a valid leader or not, while the nodes participating in an election simply commutes between three states depending on whether they have a valid leader, are part of an election waiting for acknowledgements from children, or are part of an election waiting for the announcement of the leader.

6 Conclusion

In this paper we have introduced a novel continuous time stochastic broadcast calculus for mobile and wireless broadcasting networks, which is able to model stochastic phenomena in mobile networks, like e.g. random back off protocols. We also allow for simultaneous mobility of several nodes due to a stochastic mobility model, and the mobility of nodes may be limited due to network constraints. Also, in order to minimize the state space of our models we have introduced an operator to avoid flooding in networks, and we allow for group broadcast, these two operators facilitate a novel notion of abstraction of broadcast messages where several broadcast messages may be simulated by just one broadcast message or simply be abstracted and become an internal message.

A weak bisimulation congruence $\approx_{mf}$ is defined and applied on the example of a leader election protocol for wireless networks.

References

1. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theor. Comput. Sci. 367, 203–227 (2006)
2. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A Process Calculus for Mobile Ad Hoc Networks. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 296–314. Springer, Heidelberg (2008)
3. Merro, M.: An observational theory for mobile ad hoc networks. Electron. Notes Theor. Comput. Sci. 173, 275–293 (2007)
4. Ghassemi, F., Fokkink, W., Movaghar, A.: Restricted broadcast process theory. In: Proceedings of the 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods, pp. 345–354. IEEE Computer Society (2008)
5. Godskesen, J.C.: A Calculus for Mobile Ad Hoc Networks. In: Murphy, A.L., Dell'Acqua, P. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 132–150. Springer, Heidelberg (2007)
6. Song, L., Godskesen, J.C.: Probabilistic Mobility Models for Mobile and Wireless Networks. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 86–100. Springer, Heidelberg (2010)
7. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic J. of Computing 2, 250–273 (1995)
8. Hermanns, H.: Interactive Markov chains: and the quest for quantified quality. Springer, Heidelberg (2002)
9. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, pp. 342–351. IEEE Computer Society (2010)
10. Ghassemi, F., Talebi, M., Movaghar, A., Fokkink, W.: Stochastic Restricted Broadcast Process Theory. In: Thomas, N. (ed.) EPEW 2011. LNCS, vol. 6977, pp. 72–86. Springer, Heidelberg (2011)
11. Hillston, J.: A compositional approach to performance modelling. Cambridge University Press, New York (1996)
12. Vasudevan, S., Kurose, J., Towsley, D.: Design and analysis of a leader election algorithm for mobile ad hoc networks. In: Proceedings of the 12th IEEE International Conference on Network Protocols, pp. 350–360. IEEE Computer Society (2004)
13. Nicollin, X., Sifakis, J.: An Overview and Synthesis on Timed Process Algebras. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 376–398. Springer, Heidelberg (1992)
14. Yi, W.: CCS + Time = An Interleaving Model for Real Time Systems. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 217–228. Springer, Heidelberg (1991)

An Intersection Type System for Deterministic Pushdown Automata

Takeshi Tsukada[1] and Naoki Kobayashi[2]

[1] Tohoku University
[2] University of Tokyo

Abstract. We propose a generic method for deciding the language inclusion problem between context-free languages and deterministic context-free languages. Our method extends a given decision procedure for a subclass to another decision procedure for a more general subclass called a refinement of the former. To decide $\mathcal{L}_0 \subseteq \mathcal{L}_1$, we take two additional arguments: a language $\mathcal{L}_2$ of which $\mathcal{L}_1$ is a refinement, and a proof of $\mathcal{L}_0 \subseteq \mathcal{L}_2$. Our technique then refines the proof of $\mathcal{L}_0 \subseteq \mathcal{L}_2$ to a proof or a refutation of $\mathcal{L}_0 \subseteq \mathcal{L}_1$. Although the refinement procedure may not terminate in general, we give a sufficient condition for the termination. We employ a type-based approach to formalize the idea, inspired from Kobayashi's intersection type system for model-checking recursion schemes. To demonstrate the usefulness, we apply this method to obtain simpler proofs of the previous results of Minamide and Tozawa on the inclusion between context-free languages and regular hedge languages, and of Greibach and Friedman on the inclusion between context-free languages and superdeterministic languages.

1 Introduction

The language inclusion problem, which asks whether $\mathcal{L}_0 \subseteq \mathcal{L}_1$ for languages $\mathcal{L}_0$ and $\mathcal{L}_1$, is a fundamental problem in the field of formal language theory. We are interested in its decidability, mainly motivated by applications to program verification [1,7,12]. We consider the case that $\mathcal{L}_0$ and $\mathcal{L}_1$ range over context-free languages. It is well known that the inclusion $\mathcal{L}_0 \subseteq \mathcal{L}_1$ is undecidable for context-free languages $\mathcal{L}_0$ and $\mathcal{L}_1$. For some subclasses of context-free languages, however, the inclusion is decidable [3].

In the present paper, we propose a generic method for deciding the inclusion problem. Our method extends a decision procedure for a subclass of context-free languages to another decision procedure for a more general subclass. For example, consider the languages consisting of open and close tags, like XML documents. It is known to be decidable whether a given context-free language is included in the Dyck language, which is the set of all words consisting of correctly nested tags. Using our method, we can extend this result to obtain a new proof of the decidability of inclusion between context-free languages and regular hedge languages [12].

Our method can be outlined as follows. Suppose that a decision procedure is given, which takes a language $\mathcal{L}_0$ and decides whether $\mathcal{L}_0 \subseteq \mathcal{L}_2$ for a fixed language $\mathcal{L}_2$ (in the example above, the language of all correctly nested tags). We assume that the procedure returns a "proof" of $\mathcal{L}_0 \subseteq \mathcal{L}_2$ if it is the case. By using this procedure, our method provides a way of deciding whether $\mathcal{L}_0 \subseteq \mathcal{L}_1$, where $\mathcal{L}_1$ is a subset of $\mathcal{L}_2$, called a *refinement* [19] of $\mathcal{L}_2$ (in the above example, a regular hedge language). To decide $\mathcal{L}_0 \subseteq \mathcal{L}_1$, we first decide whether $\mathcal{L}_0 \subseteq \mathcal{L}_2$, using the decision procedure. If $\mathcal{L}_0 \nsubseteq \mathcal{L}_2$, we conclude $\mathcal{L}_0 \nsubseteq \mathcal{L}_1$. If $\mathcal{L}_0 \subseteq \mathcal{L}_2$, the procedure returns a "proof" of it, and we decide the inclusion $\mathcal{L}_0 \subseteq \mathcal{L}_1$ by refining the "proof" of $\mathcal{L}_0 \subseteq \mathcal{L}_2$.

To formalize the idea, we employ a type-based approach inspired by Kobayashi's intersection type system [7] for the model checking of higher-order recursion schemes. For each deterministic context-free language $\mathcal{L}_i$, we develop a type system characterizing context-free grammars $\mathcal{G}$ such that $\mathcal{L}_\mathcal{G} \subseteq \mathcal{L}_i$, i.e., a type system $\mathcal{T}_i$ such that $\mathcal{G}$ is typable in $\mathcal{T}_i$ if and only if $\mathcal{L}_\mathcal{G} \subseteq \mathcal{L}_i$. Then, the inclusion problem $\mathcal{L}_\mathcal{G} \subseteq \mathcal{L}_i$ is reduced to the typability of $\mathcal{G}$ in $\mathcal{T}_i$. We check it by (i) first checking whether $\mathcal{G}$ is typable in a "simpler" type system $\mathcal{T}_2$, and (ii) if $\mathcal{G}$ is typable in $\mathcal{T}_1$, enumerating "refinements" of the type derivation of $\mathcal{T}_2 \vdash \mathcal{G}$ and checking whether there exists a type derivation for $\mathcal{G}$ in $\mathcal{T}_1$ among them. (We will substantiate the meaning of "simpler type system" and "refinements" in later sections.)

We demonstrate the usefulness of the method by giving simpler proofs of two previous decidability results: (1) The result of Minamide and Tozawa [12] on the inclusion between context-free languages and regular hedge languages; (2) The result of Greibach and Friedman [5] on the inclusion between context-free languages and superdeterministic languages, which is, to our knowledge, one of the strongest results about the inclusion problems.

The rest of the paper is organized as follows. In Section 2, we define some notions and notations about context-free grammars and pushdown automata. In Section 3, we construct an intersection type system characterizing the inclusion problem. In Section 4, we develop a procedure which refines a type derivation and we give a sufficient condition for the termination of the procedure. In Section 5, we apply our method to prove some decidability results. In Section 6, we discuss the related work and we conclude in Section 7. Omitted proofs can be found in the full version, available from the authors' web pages.

2 Preliminaries

Context-Free Grammars. We present context-free grammars for words in the form of (a special case of) context-free tree grammars generating monadic trees (i.e., trees of the form $a_1(a_2(\dots(a_n(\$))\dots)))$. The definition is consistent with the standard definition of the context-free grammars.

We use a special letter \$, which can occur only at the end of a word, and distinguish between two kinds of words: those that end with \$, called *terminating words*, and those that end with a normal letter, called *normal words* (or simply, words). A *sort* κ is o describing terminating words, or $o \to o$ describing normal words. A normal word w can be considered as a function that takes a terminating word $w'\$$ and returns the terminating word $ww'\$$; that is why we assign a function sort to normal words. A *context-free grammar* (CFG, for short) is a quadruple $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$, where:

1. $\mathcal{N}$ is a finite set of symbols called *non-terminals.* They have the sort $o \to o$. Non-terminals are ranged over by F.
2. Σ is a finite set of symbols called *terminals.* We use metavariables a and b for terminals. They also have the sort $o \to o$.
3. $\mathcal{R}$ is a set of rewriting rules of the form $F\ x \to t$, where x is a variable of the sort o and t is a term of the form $\alpha_1(\alpha_2(\dots(\alpha_n(x))\dots))$ with $\alpha_i \in \Sigma \cup \mathcal{N}$. There can be more than one rule for the same non-terminal.
4. S is a distinguished non-terminal, called the *initial symbol.*

We use t and s as metavariables of terms and α as a metavariable ranging over $\Sigma \cup \mathcal{N}$. The *rewriting relation* $\Rightarrow_{\mathcal{R}}$ is defined by:

$$F\ s \Rightarrow_{\mathcal{R}} t[s/x] \text{ if } (F\ x \to t) \in \mathcal{R} \qquad \alpha\, t \Rightarrow_{\mathcal{R}} \alpha\, t' \text{ if } t \Rightarrow_{\mathcal{R}} t'$$

Here $t[s/x]$ is the term obtained by substituting s for x in t. We write $\Rightarrow_{\mathcal{R}}^*$ for the reflexive and transitive closure of $\Rightarrow_{\mathcal{R}}$. We often omit $\mathcal{R}$ if it is clear from the context. For a given non-terminal F, we define the *language generated by* F as $\mathcal{L}_{\mathcal{G}}(F) = \{a_1a_2\dots a_n \in \Sigma^* \mid F\ \$ \Rightarrow^* a_1(a_2(\dots(a_n(\$))\dots))\}$. The *language generated by* $\mathcal{G}$, written $\mathcal{L}_{\mathcal{G}}$, is $\mathcal{L}_{\mathcal{G}}(S)$.

Example 1. For a given alphabet Σ, we define the set of open tags $\acute{\Sigma} = \{\acute{a} \mid a \in \Sigma\}$ and close tags $\grave{\Sigma} = \{\grave{a} \mid a \in \Sigma\}$. Let $\mathcal{G}_0 = (\{S, F_a, F_b\}, \acute{\Sigma}_0 \cup \grave{\Sigma}_0, \mathcal{R}, S)$, where $\Sigma_0 = \{\mathbf{a}, \mathbf{b}\}$ and $\mathcal{R} = \{S\, x \to x,\ S\, x \to \acute{\mathbf{a}}(F_a(x)),\ S\, x \to F_b(\grave{\mathbf{b}}(x)),\ F_a\, x \to S(\grave{\mathbf{a}}(x)),\ F_b\, x \to \acute{\mathbf{b}}(S(x))\}$. The language $\mathcal{L}_{\mathcal{G}}$ consists of words of the form $\acute{a}_1\acute{a}_2\dots\acute{a}_n\grave{a}_n\dots\grave{a}_1$, where $a_i \in \{\mathbf{a}, \mathbf{b}\}$ for all $1 \le i \le n$.

The rules of this CFG can be written in the standard notation as:

$$S \to \varepsilon \mid \acute{\mathbf{a}}\, F_a \mid F_b\, \grave{\mathbf{b}}, \qquad F_a \to S\, \grave{\mathbf{a}}, \qquad F_b \to \acute{\mathbf{b}}\, S,$$

where ε denotes the empty word. □

Pushdown Automaton. A *pushdown automaton* (PDA, for short) is a quadruple $M = (Q, \Sigma, \Gamma, \delta)$, where (1) Q is a finite set of *states*; (2) Σ is an alphabet; (3) Γ is a finite set of *stack symbols* (we use metavariables A and B for stack symbols), and (4) $\delta \subseteq Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ is a *transition relation.* We use $\widetilde{A}$ and $\widetilde{B}$ to denote (possibly empty) sequences of stack symbols. For $q \in Q$, $A \in \Gamma$ and $a \in \Sigma \cup \{\varepsilon\}$, we define $\delta(q, A, a) = \{(q', \widetilde{A}') \mid (q, A, a, q', \widetilde{A}') \in \delta\}$. A pushdown automaton is *deterministic* if for any $q \in Q$, $A \in \Gamma$ and $a \in \Sigma$, the set $\delta(q, A, a) \cup \delta(q, A, \varepsilon)$ has exactly one element. In the rest of the paper, we consider only deterministic pushdown automata.

We call an element of $Q \times \Gamma^*$ a *configuration.* If $(q, A, a, q', \widetilde{A}') \in \delta$ (here $a \in \Sigma \cup \{\varepsilon\}$), we write $(q, \widetilde{B}A) \Vdash_M^a (q', \widetilde{B}\widetilde{A}')$. We say a configuration c is in *reading mode* if c has no ε-transition, i.e., there is no configuration c' such that $c \Vdash_M^\varepsilon c'$. For configurations c and c' in reading mode and $a \in \Sigma$, we write $c \vDash_M^a c'$ if

$$c \Vdash_M^a d_1 \Vdash_M^\varepsilon d_2 \Vdash_M^\varepsilon \cdots \Vdash_M^\varepsilon d_n \Vdash_M^\varepsilon c' \nVdash_M^\varepsilon .$$

For $w = a_1 a_2 \ldots a_n \in \Sigma^*$, we write $c \vDash_M^w c'$ if $c \vDash_M^{a_1} d_1 \vDash_M^{a_2} d_2 \vDash_M^{a_3} \cdots \vDash_M^{a_n} c'$.

For a given configuration c in reading mode and a given set $\mathcal{F}$ of configurations in reading mode, we define $\mathcal{L}_M(c, \mathcal{F}) = \{w \in \Sigma^* \mid \exists c' \in \mathcal{F}.\ c \vDash_M^w c'\}$. Here c indicates the initial configuration and $\mathcal{F}$ the set of accepting configurations.

Example 2. Recall Σ_0 and $\mathcal{G}_0$ defined in Example 1. We define $\mathcal{A}_2 = \langle \{q\}, \acute{\Sigma}_0 \cup \grave{\Sigma}_0, \{\star\}, \delta_{\mathcal{A}_2} \rangle$, where $\delta_{\mathcal{A}_2} = \{(q, \star, \acute{a}, q, \star\star), (q, \star, \grave{a}, q, \varepsilon) \mid a \in \Sigma_0\}$. The automaton $\mathcal{A}_2$ counts and records the difference between the numbers of open tags and close tags, ignoring their labels. Let $L = \mathcal{L}_{\mathcal{A}_2}((q, \star), \{(q, \star)\})$. Then L is the set of all balanced tags, e.g., $\acute{\mathbf{a}}\grave{\mathbf{b}} \in L$ but $\acute{\mathbf{a}}\grave{\mathbf{a}}\grave{\mathbf{b}}\acute{\mathbf{b}} \notin L$. It is obvious that $\mathcal{L}_{\mathcal{G}_0} \subseteq \mathcal{L}_{\mathcal{A}_2}((q, \star), \{(q, \star)\})$.

We define a different PDA $\mathcal{A}_1 = \langle \{q_1, q_2\}, \acute{\Sigma}_0 \cup \grave{\Sigma}_0, \Sigma \cup \{\bot\}, \delta_{\mathcal{A}_1} \rangle$, where $\delta_{\mathcal{A}_1} = \{(q_1, A, \acute{a}, q_1, Aa) \mid A \in \Sigma_0 \cup \{\bot\}, a \in \Sigma_0\} \cup \{(q_1, a, \grave{a}, q_2, \varepsilon), (q_2, a, \grave{a}, q_2, \varepsilon) \mid a \in \Sigma_0\}$. In addition to counting the difference of open tags and close tags, $\mathcal{A}_1$ records labels of open tags on its stack, and checks if end tags are already read, by using its state. Let $L' = \mathcal{L}_{\mathcal{A}_1}((q_1, \bot), \{(q_1, \bot), (q_2, \bot)\})$. Then L' is the set of all words of the form $\acute{a}_1 \acute{a}_2 \ldots \acute{a}_n \grave{a}_n \ldots \grave{a}_2 \grave{a}_1$, where $a_i \in \Sigma_0$. Thus $\mathcal{L}_{\mathcal{G}_0} = L'$. □

3 Type System

We construct a type system $\mathcal{T}_M$ for each PDA M which characterizes the CFGs generating languages accepted by M. In the rest of this section, we fix a PDA M and discuss the definition and properties of the type system $\mathcal{T}_M$.

The syntax of types is defined by: $\tau ::= c \mid \bigwedge \Theta \to c$, where c ranges over configurations of M in reading mode and Θ is a (possibly infinite) set of configurations in reading mode. We often abbreviate $\bigwedge\{d\} \to c$ as $d \to c$. We say a type c has the sort o (written as $c :: o$) and a type $\bigwedge \Theta \to c$ has the sort $o \to o$ (written as $\bigwedge \Theta \to c :: o \to o$). Intuitively, the type c is for terminating words accepted from c (by ignoring \$ at the end). Interpretations of $\to$ and $\bigwedge$ are standard: $d \to c$ describes functions from d to c and $c_1 \bigwedge c_2$ describes terminating words accepted from the both of c_1 and c_2. Thus a normal word $w = a_1 \ldots a_n$, which can be considered as a function $\lambda x.\, a_1(a_2(\ldots(a_n(x))\ldots))$, has a type $d \to c$ if $c \vDash_M^{a_1 a_2 \ldots a_n} d$.

A *type environment* is a (possible infinite) set of bindings of the form $x : \tau$ or $F : \tau$. We allow multiple bindings for the same variable (or the same non-terminal), as in $\{x : \tau_1, x : \tau_2\}$. We often omit curly brackets, and simply write $x_1 : \tau_1, \ldots, x_n : \tau_n$ for $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$. We abbreviate $\{x : c \mid c \in \Theta\}$ as $x : \bigwedge \Theta$. We define $\Delta(x) = \{\tau \mid x : \tau \in \Delta\}$. A type environment Δ is *well-formed* if it respects the sort, i.e., $x : \tau \in \Delta$ implies $\tau :: o$ and $F : \tau \in \Delta$ implies $\tau :: o \to o$. We assume that all type environments appearing in the sequel are well-formed.

The typing rules are listed as follows.

$$\frac{x : \tau \in \Delta}{\Delta \vdash_M x : \tau} \qquad \frac{F : \tau \in \Delta}{\Delta \vdash_M F : \tau} \qquad \frac{\begin{array}{c}\Delta \vdash_M t_1 : \bigwedge \Theta \to c \\ \Delta \vdash_M t_2 : d \ (\text{for all } d \in \Theta)\end{array}}{\Delta \vdash_M t_1\, t_2 : c} \qquad \frac{c \vDash_M^a c'}{\Delta \vdash_M a : c' \to c}$$

These are standard rules for intersection type systems except for the last rule for constants, which is inspired by Kobayashi's type system [7]. Types of constants depend on the transition rule of the automaton, as explained below. Assume $c \vDash^a_M c'$. Then for any (normal) word w accepted from c', aw is accepted from c. By using type-based notations, for any (terminated) word $w(\$) : c'$, we have $a(w(\$)) : c$. Thus a can be considered as a function of type $c' \to c$.

We say that a type environment Δ is an *invariant* of the rules $\mathcal{R}$, written $\Delta \vdash_M \mathcal{R}$, if $\Delta, x : \bigwedge \Theta \vdash_M t : c$ holds for all $F : \bigwedge \Theta \to c \in \Delta$ and $F\,x \to t \in \mathcal{R}$. We write $\Delta \vdash_M (\mathcal{R}, S) : \bigwedge \Theta \to c$ if $\Delta \vdash_M \mathcal{R}$ and $\Delta, \$: \bigwedge \Theta \vdash_M S\$: c$ (in the type system, \$ is treated as a variable).

Theorem 1. *Let $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG, M be a PDA, c be a configuration of M and $\mathcal{F}$ be a set of configurations of M. Then $\mathcal{L}_\mathcal{G}(S) \subseteq \mathcal{L}_M(c, \mathcal{F})$ if and only if $\Delta \vdash_M (\mathcal{R}, S) : \bigwedge \mathcal{F} \to c$ for some type environment Δ.*

Proof. The "if" direction follows from the facts that typing is preserved by reductions of $S\$$, and that $\$: \bigwedge \mathcal{F} \vdash_M w\$: c$ implies $w \in \mathcal{L}_M(c, \mathcal{F})$. For the other direction, let $\Delta = \{F : \bigwedge \Theta \to d \mid \mathcal{L}_\mathcal{G}(F) \subseteq \mathcal{L}_M(d, \Theta)\}$. □

By Theorem 1, the pair of the initial configuration c and the set $\mathcal{F}$ of accepting configurations can be identified with the type $\bigwedge \mathcal{F} \to c$. We call the type $\iota = \bigwedge \mathcal{F} \to c$ the *initial type* and write $\mathcal{L}_M(\iota)$ for $\mathcal{L}_M(c, \mathcal{F})$. When $\Delta \vdash_M (\mathcal{R}, S) : \tau$, the environment Δ is called a *witness of* $\vdash_M (\mathcal{R}, S) : \tau$.

We introduce a partial order on witnesses and show the existence of the minimum witness.

Definition 1. *The* refinement ordering $\sqsubseteq$ *is the smallest partial order that satisfies: (1) $\Theta_1 \sqsubseteq \Theta_2$ if $\Theta_1 \subseteq \Theta_2$, (2) $(\bigwedge \Theta_1 \to c_1) \sqsubseteq (\bigwedge \Theta_2 \to c_2)$ if $c_1 = c_2$ and $\Theta_1 \sqsubseteq \Theta_2$, and (3) $\Delta_1 \sqsubseteq \Delta_2$ if $\Delta_1(x) \sqsubseteq \Delta_2(x)$ for every x.* □

Lemma 1. *Let $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG, M be a PDA and ι be its initial type. Assume that $\mathcal{L}_\mathcal{G}(S) \subseteq \mathcal{L}_M(\iota)$. Then the set of witnesses of $\vdash_M (\mathcal{R}, S) : \iota$, i.e., $\{\Delta \mid \Delta \vdash_M (\mathcal{R}, S) : \iota\}$, has the minimum element with respect to $\sqsubseteq$.*

Proof. Let $\iota = \bigwedge \Theta \to c$. For a non-terminal F, we define $\mathrm{pre}(F) = \{w \mid S\$ \Rightarrow^*_\mathcal{R} wFv\$\}$. Let $\Delta_0 = \{F : \bigwedge \Theta' \to c' \mid \exists w \in \mathrm{pre}(F).\ c \vDash^w_M c' \text{ and } \Theta' = \{d' \mid \exists u \in \mathcal{L}_\mathcal{G}(F).\ c' \vDash^u_M d'\}\}$. Then $\Delta_0 \vdash_M (\mathcal{R}, S) : \iota$ and Δ_0 is minimum: See the full version for more details. □

Example 3. Let $\mathcal{G}_0$ be the CFG defined in Example 1, $\mathcal{A}_2$ be the PDA defined in Example 2 and $\iota_2 = (q, \star) \to (q, \star)$. Since $\mathcal{L}_{\mathcal{G}_0} \subseteq \mathcal{L}_{\mathcal{A}_2}(\iota_2)$, by Theorem 1, there is Δ such that $\Delta \vdash_{\mathcal{A}_2} (\mathcal{R}, S) : \iota_2$. The minimum witnesses is given by $\{S : (q, \widetilde{A}) \to (q, \widetilde{A}), F_a : (q, \widetilde{A}) \to (q, \widetilde{A}\star), F_b : (q, \widetilde{A}\star) \to (q, \widetilde{A}) \mid \widetilde{A} \in \{\star\}^+\}$, where $\{\star\}^+$ is the set of non-empty sequences of $\star$. □

Note that a minimum type environment may be infinite as in Example 3. In the rest of this section, we develop a way to finitely describe (some of) infinite type environments.

An important property of pushdown automata is that only the top of the stack affects its transition. Especially, we can add any stack symbols to the bottom, preserving the transition. For example, let $\mathcal{A}_1$ be the automaton defined in Example 2 and $w = \acute{\mathbf{a}}\grave{\mathbf{a}}\grave{\mathbf{b}}$. Then we have a transition $(q_1, \mathbf{bbb}) \vDash^w_{\mathcal{A}_1} (q_2, \mathbf{bb})$. By adding $\bot\mathbf{aa}$ to the bottom of the stack, we obtain $(q_1, \bot\mathbf{aabbb}) \vDash^w_{\mathcal{A}_1} (q_2, \bot\mathbf{aabb})$. More generally, for any sequence $\widetilde{A}$ of stack symbols, we have $(q_1, \widetilde{A}\mathbf{bbb}) \vDash^w_{\mathcal{A}_1} (q_2, \widetilde{A}\mathbf{bb})$. This does not depend on the choice of w, i.e., for any w such that $(q_1, \mathbf{bbb}) \vDash^w_{\mathcal{A}_1} (q_2, \mathbf{bb})$, we have $(q_1, \widetilde{A}\mathbf{bbb}) \vDash^w_{\mathcal{A}_1} (q_2, \widetilde{A}\mathbf{bb})$.

We will formally state this fact in terms of intersection types (see Lemma 2).

Definition 2. *For a given (possible empty) sequence $\widetilde{B}$ of stack symbols and a given configuration $(q, \widetilde{A})$, we define the* stack extension $(q, \widetilde{A}) \Uparrow \widetilde{B}$ *as $(q, \widetilde{B}\widetilde{A})$. We define $(\Theta \Uparrow \widetilde{B}) = \{c \Uparrow \widetilde{B} \mid c \in \Theta\}$ for the set of configurations, $(\bigwedge \Theta \to c) \Uparrow \widetilde{B} = \bigwedge(\Theta \Uparrow \widetilde{B}) \to (c \Uparrow \widetilde{B})$ for the type, $\Delta \Uparrow \widetilde{B} = \{x : (\tau \Uparrow \widetilde{B}) \mid x : \tau \in \Delta\}$ for the type environment and $(\Delta \vdash t : \tau) \Uparrow \widetilde{B} = (\Delta \Uparrow \widetilde{B}) \vdash t : (\tau \Uparrow \widetilde{B})$ for the judgement. We define $\Delta^{\Uparrow} = \cup_{\widetilde{B}}(\Delta \Uparrow \widetilde{B})$.* □

Lemma 2. *If $\Delta \vdash_M t : \tau$, then for any $\widetilde{B}$, we have $(\Delta \vdash_M t : \tau) \Uparrow \widetilde{B}$.*

Proof. Easy induction on $\Delta \vdash_M t : \tau$. □

We write $\Delta \vdash^{\Uparrow}_M \mathcal{R}$, read "$\Delta$ is an invariant of $\mathcal{R}$ up-to stack extensions", if for every $F : \bigwedge \Theta \to c \in \Delta$ and $F\,x \to t \in \mathcal{R}$, we have $(\Delta^{\Uparrow}), x : \bigwedge \Theta \vdash_M t : c$. Note that while $F : \bigwedge \Theta \to c$ is chosen from Δ, the environment to type the body of F is $\Delta^{\Uparrow}$. The judgement $\Delta \vdash^{\Uparrow}_M (\mathcal{R}, S) : \bigwedge \Theta \to c$ is defined as $\Delta \vdash^{\Uparrow}_M \mathcal{R}$ and $(\Delta^{\Uparrow}), \$: \bigwedge \Theta \vdash_M S\$: c$.

By using this up-to technique, we can sometimes (but not always) finitely describe a witness type environment as shown in the example below.

Example 4. Recall Example 3. We have $\Delta \vdash^{\Uparrow}_{\mathcal{A}_2} (\mathcal{R}, S) : \iota_2$, where $\Delta = \{S : (q, \star) \to (q, \star), F_a : (q, \star) \to (q, \star\star), F_b : (q, \star\star) \to (q, \star)\}$. Note that Δ is a finite set. □

This up-to technique is sound in the sense that if a CFG is typable up-to stack expansions, then it is typable without using the up-to technique.

Theorem 2. $\Delta \vdash^{\Uparrow}_M (\mathcal{R}, S) : \iota$ *implies* $(\Delta^{\Uparrow}) \vdash_M (\mathcal{R}, S) : \iota$.

Proof. We should show that $(\Delta^{\Uparrow}) \vdash_M \mathcal{R}$ and $(\Delta^{\Uparrow}), \$: \bigwedge \Theta \vdash_M S\$: c$, where $\iota = \bigwedge \Theta \to c$. The latter comes from the assumption. To show the former, assume $F : \tau \in (\Delta^{\Uparrow})$ and $F\,x \to t \in \mathcal{R}$. Then we have $F : \sigma \in \Delta$ and $\tau = (\sigma \Uparrow \widetilde{A})$ for some σ and $\widetilde{A}$. Let $\sigma = \bigwedge \Xi \to d$. Then $\tau = \bigwedge(\Xi \Uparrow \widetilde{A}) \to (d \Uparrow \widetilde{A})$. We should show that $(\Delta^{\Uparrow}), (x : \bigwedge \Xi \Uparrow \widetilde{A}) \vdash_M t : (d \Uparrow \widetilde{A})$. By the assumption, $(\Delta^{\Uparrow}), x : \bigwedge \Xi \vdash_M t : d$. By the previous lemma, we have $((\Delta^{\Uparrow}) \Uparrow \widetilde{A}), (x : \bigwedge \Xi \Uparrow \widetilde{A}) \vdash_M t : (d \Uparrow \widetilde{A})$. Because $((\Delta^{\Uparrow}) \Uparrow \widetilde{A}) \subseteq (\Delta^{\Uparrow})^{\Uparrow} = \Delta^{\Uparrow}$, we conclude $(\Delta^{\Uparrow}), (x : \bigwedge \Xi \Uparrow \widetilde{A}) \vdash_M t : (d \Uparrow \widetilde{A})$. □

4 Refining Witnesses

It is in general difficult (in fact undecidable) to check whether a given CFG $\mathcal{G}$ is typable in $\mathcal{T}_{M_1}$ for a given PDA M_1, so that we first consider a simpler PDA M_2 and check whether G is typable in $\mathcal{T}_{M_2}$. If we choose M_2 so that (i) we have a witness of typability of $\mathcal{G}$ in $\mathcal{T}_{M_2}$ and (ii) M_1 is a refinement of M_2, then $\mathcal{G}$ is typable in $\mathcal{T}_{M_1}$ if and only if there is a witness that is a refinement of the witness in $\mathcal{T}_{M_2}$ (Section 4.1). Moreover, if a witness in $\mathcal{T}_{M_2}$ is finite, then the set of its refinements is a finite set. Thus, we can decide the typability in $\mathcal{T}_{M_1}$ by exhaustively searching a witness from the (finite) set of refinements of the witness in $\mathcal{T}_{M_2}$ (Section 4.2).

4.1 Refinements of Automata

We first define the notion of *refinements* of automata. As we will see below, if M_1 is a refinement of M_2, then M_2 is a good over-approximation of M_1.

Definition 3 (Refinement of Automata). *Let $M_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1\rangle$ and $M_2 = \langle Q_2, \Sigma, \Gamma_2, \delta_2\rangle$ be pushdown automata. A* homomorphism $f : M_1 \to M_2$ *is a pair of mappings $f^Q : Q_1 \to Q_2$ and $f^\Gamma : \Gamma_1 \to \Gamma_2$ such that for any $(q, A, a, q', \widetilde{B}) \in \delta_1$, $(f^Q(q), f^\Gamma(A), a, f^Q(q'), f^\Gamma(\widetilde{B})) \in \delta_2$, where $f^\Gamma(B_1B_2 \dots B_n) = f^\Gamma(B_1)f^\Gamma(B_2)\dots f^\Gamma(B_n)$. We often omit superscripts Q and Γ, and simply write $f(q)$ and $f(\widetilde{A})$.* □

The homomorphism $f : M_1 \to M_2$ can be naturally extended to mappings on configurations, types, type environments and judgements, e.g., the mapping on configurations is defined by $f(\ (q, \widetilde{A})\ ) = (f^Q(q), f^\Gamma(\widetilde{A}))$.

When there is a homomorphism $f : M_1 \to M_2$, we say M_2 is an *approximation* of M_1 and M_1 is a *refinement* of M_2. A type τ_1 in $\mathcal{T}_{M_1}$ is a refinement of τ_2 in $\mathcal{T}_{M_2}$ if $f(\tau_1) \sqsubseteq \tau_2$. Refinements of type environments are defined similarly. We can always find a homomorphism $f : M_1 \to M_2$ if it exists, since both of $Q_1 \to Q_2$ and $\Gamma_1 \to \Gamma_2$ are finite. We write $f : (M_1, \iota_1) \to (M_2, \iota_2)$ if $f : M_1 \to M_2$ and $f(\iota_1) = \iota_2$. The next lemma justifies to say that M_2 is an (over-)approximation of M_1.

Lemma 3. *If $f : (M_1, \iota_1) \to (M_2, \iota_2)$, then $\mathcal{L}_{M_1}(\iota_1) \subseteq \mathcal{L}_{M_2}(\iota_2)$.* □

Example 5. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be automata defined in Example 2. Then $\mathcal{A}_1$ is a refinement of $\mathcal{A}_2$ by a homomorphism $(h^Q, h^\Gamma) : \mathcal{A}_1 \to \mathcal{A}_2$ given by $h^Q(q_1) = h^Q(q_2) = q$ and $h^\Gamma(\mathbf{a}) = h^\Gamma(\mathbf{b}) = h^\Gamma(\bot) = \star$. □

In the following, we fix two pushdown automata (with their initial types) (M_1, ι_1) and (M_2, ι_2) and a homomorphism $f : (M_1, \iota_1) \to (M_2, \iota_2)$ between them. For readability, we write $\mathcal{T}_1$ instead of $\mathcal{T}_{M_1}$, $\mathcal{L}_1$ instead of $\mathcal{L}_{M_1}$ and so on.

Validity of type judgements and minimality of a witness are preserved by f.

Theorem 3. *Let $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG, M_1 and M_2 be PDAs, ι_1 and ι_2 be their initial types and $f : (M_1, \iota_1) \to (M_2, \iota_2)$ be a homomorphism.*

1. *If $\Delta \vdash_{M_1} (\mathcal{R}, F) : \iota_1$, then $f(\Delta) \vdash_{M_2} (\mathcal{R}, F) : \iota_2$.*
2. *If Δ is the minimum witness of $\vdash_{M_1} (\mathcal{R}, F) : \iota_1$, then $f(\Delta)$ is the minimum witness of $\vdash_{M_2} (\mathcal{R}, F) : \iota_2$.*

Proof. It is easy to prove that $\Delta \vdash_{M_1} t : \tau$ implies $f(\Delta) \vdash_{M_2} t : f(\tau)$ by induction on t. The first part of the claim is an easy consequence of this proposition. The second part is clear from the construction of the minimum witness in the proof of Lemma 1. $\square$

A witness Δ_2 in $\mathcal{T}_2$ ensures the existence of a "smaller" witness in $\mathcal{T}_1$.

Theorem 4. *Let $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG, M_1 and M_2 be PDAs, ι_1 and ι_2 be their initial types and $f : (M_1, \iota_1) \to (M_2, \iota_2)$ be a homomorphism. Assume that $\Delta_2 \vdash^{\Uparrow}_{M_2} (\mathcal{R}, S) : \iota_2$. If $\Delta_1 \vdash^{\Uparrow}_{M_1} (\mathcal{R}, S) : \iota_1$, then there exists Δ'_1 such that $\Delta'_1 \vdash^{\Uparrow}_{M_1} (\mathcal{R}, S) : \iota_1$ and $f(\Delta'_1) \sqsubseteq \Delta_2$.*

Proof. Here, we give a proof sketch. Since $\Delta_1 \vdash^{\Uparrow}_{M_1} (\mathcal{R}, S) : \iota_1$, there is the minimum witness type environment by Lemma 1. Let Δ^0_1 be the minimum witness of $\vdash_{M_1} (\mathcal{R}, S) : \iota_1$. Note that $f(\Delta^0_1) \sqsubseteq \Delta^{\Uparrow}_2$ by Theorem 3.

We shorten the types in Δ^0_1, appropriately. We define $(q, A_1 A_2 \dots A_m) \Downarrow n = (q, A_{n+1} \dots A_m)$ if $m > n$ (and undefined otherwise). This operation is extended to types by $(\bigwedge \Theta \to c) \Downarrow n = \bigwedge \{ d \Downarrow n \mid d \in \Theta \} \to (c \Downarrow n)$. Let $(F, \tau^0_1, \tau_2, \widetilde{A}_2)$ be a quadruple such that $F : \tau^0_1 \in \Delta^0_1$, $F : \tau_2 \in \Delta_2$ and $f(\tau^0_1) \sqsubseteq (\tau_2 \Uparrow \widetilde{A}_2)$. The corresponding type binding $F : \tau'_1$ of the quadruple is defined by $\tau'_1 = \tau^0_1 \Downarrow n$, where n is the length of $\widetilde{A}_2$. Let Δ'_1 be the set of all such bindings $F : \tau'_1$. Then Δ'_1 satisfies the above conditions: See the full version for a more detailed proof. $\square$

4.2 Procedure and Sufficient Condition for Termination

Recall the overall picture of our method to understand the role of the procedure developed here. The final goal is to decide whether $\mathcal{G}$ is typable in $\mathcal{T}_1$. To solve the problem, we first check whether $\mathcal{G}$ is typable in $\mathcal{T}_2$, and if so, use the derivation for $\mathcal{T}_2$ and Theorem 4 to check whether $\mathcal{G}$ is typable in $\mathcal{T}_1$. The procedure developed here takes care of this last step.

Before describing the procedure, we define the notion of *finiteness*. We say that any base type q is *finite* and a type $\bigwedge \Theta \to c$ is finite if Θ is a finite set. A type environment Δ is finite if Δ is a finite set and for every type binding $x : \tau \in \Delta$, τ is finite.

Figure 1 shows the procedure that refines a *finite* witness in $\mathcal{T}_2$ to one in $\mathcal{T}_1$. Here for a given grammar $\mathcal{G}$ and its rewriting relation $\mathcal{R}$, the function $\mathcal{H}$ on type environments in $\mathcal{T}_1$ is defined by

$$\mathcal{H}(\Delta_1) = \{ F : \bigwedge \Theta \to c \in \Delta_1 \mid \forall (F\, x \to t) \in \mathcal{R}.\ \Delta_1, x : \bigwedge \Theta \vdash^{\Uparrow}_{M_1} t : c \}.$$

The procedure takes five arguments: a grammar $\mathcal{G}$, two PDAs with the initial types (M_1, ι_1) and (M_2, ι_2), a homomorphism $f : (M_1, \iota_1) \to (M_2, \iota_2)$ and a finite

```
Refine(𝒢, (M_1, ι_1), (M_2, ι_2), f, Δ_2).
  1. Let $n := 0$ and $\Delta_1^0 := \{F : \tau_1 \mid \exists \tau_2.\ F : \tau_2 \in \Delta_2 \text{ and } f(\tau_1) \sqsubseteq \tau_2\}$.
  2. Compute a fixed-point $\Delta_1$ of $\mathcal{H}$ starting from $\Delta_1^0$ as follows:
     (a) Let $\Delta_1^{n+1} := \mathcal{H}(\Delta_1^n)$.
     (b) If $\Delta_1^n = \Delta_1^{n+1}$, then $\Delta_1^n$ is a fixed-point of $\mathcal{H}$.
     (c) Otherwise, let $n := n + 1$ and goto (a).
  3. Check whether $S:\iota_1 \in \Delta_1$. If so, return $\Delta_1$. Otherwise, return untypable.
```

Fig. 1. The procedure to refine a witness

type environment Δ_2 in $\mathcal{T}_2$ such that $\Delta_2 \vdash^{\Uparrow}_{M_2} (\mathcal{R}, S){:}\iota_2$. The finiteness of the type environment ensures the termination of the procedure. The procedure returns a witness if it exists, and otherwise returns **untypable**.

Example 6. Let $\mathcal{G}_0$ be the CFG defined in Example 1, $\mathcal{A}_1$ and $\mathcal{A}_2$ be PDAs defined in Example 2, Δ' be the finite witness of $\vdash_{\mathcal{A}_2} (\mathcal{R}, S) : \iota_{\mathcal{A}_2}$ defined in Example 4, $f : \mathcal{A}_1 \to \mathcal{A}_2$ be the homomorphism defined in Example 5 and $\iota_{\mathcal{A}_1} = (q_1, \bot) \wedge (q_2, \bot) \to (q_1, \bot)$. We compute a witness of $\vdash_{\mathcal{A}_1} (\mathcal{R}, S) : \iota_{\mathcal{A}_1}$ by our procedure **Refine**.

The starting point Δ_1^0 for computing a fixed-point of $\mathcal{H}$ is the set of all refinements of type bindings in Δ'. For example, $\Delta_1^0(S)$ is given by

$$\left\{\begin{array}{llllll}
\bigwedge\emptyset & \to (q_1, \mathbf{a}), & \bigwedge\emptyset & \to (q_1, \mathbf{b}), & \bigwedge\emptyset & \to (q_1, \bot) \\
\bigwedge\emptyset & \to (q_2, \mathbf{a}), & \bigwedge\emptyset & \to (q_2, \mathbf{b}), & \bigwedge\emptyset & \to (q_2, \bot) \\
(q_1, \mathbf{a}) & \to (q_1, \mathbf{a}), & (q_1, \mathbf{a}) & \to (q_1, \mathbf{b}), & (q_1, \mathbf{a}) & \to (q_1, \bot) \\
(q_1, \mathbf{a}) & \to (q_2, \mathbf{a}), & (q_1, \mathbf{a}) & \to (q_2, \mathbf{b}), & (q_1, \mathbf{a}) & \to (q_2, \bot) \\
(q_1, \mathbf{b}) & \to (q_1, \mathbf{a}), & (q_1, \mathbf{b}) & \to (q_1, \mathbf{b}), & (q_1, \mathbf{b}) & \to (q_1, \bot) \\
\vdots & & & & & \\
\multicolumn{3}{l}{(q_1, \mathbf{a}) \wedge (q_1, \mathbf{b}) \to (q_1, \mathbf{a}),} & \cdots & & \\
\multicolumn{3}{l}{(q_1, \mathbf{a}) \wedge (q_1, \mathbf{b}) \wedge (q_2, \mathbf{a}) \to (q_1, \mathbf{b}),} & \cdots & &
\end{array}\right\}$$

since $\Delta'(S) = \{(q, \star) \to (q, \star)\}$. The type $\tau = (q_1, \mathbf{a}) \to (q_2, \mathbf{ab})$ does not belong to $\Delta_1^0(S)$, since $f(\tau) = (q, \star) \to (q, \star\star) \not\sqsubseteq (q, \star) \to (q, \star)$. The set $\Delta_1^0(S)$ contains $2^6 \times 6$ elements, because there are 6 refinements of $(q, \star)$. Similarly, $\Delta_1^0(F_a)$ contains $2^6 \times 18$ elements and $\Delta_1^0(F_b)$ contains $2^{18} \times 6$ elements.

Then we filter out wrong type bindings such as $S : \bigwedge\emptyset \to (q_1, \mathbf{b}) \in \Delta_1^0$ by iteratively applying $\mathcal{H}$. For example, $S : \bigwedge\emptyset \to (q_1, \mathbf{b}) \notin \mathcal{H}(\Delta_1^0)$ because $S\,x \to x \in \mathcal{R}$ and $\Delta_1^0, x : \bigwedge\emptyset \nvdash_{\mathcal{A}_1} x : (q_1, \mathbf{b})$.

Be repeated applications of $\mathcal{H}$, we obtain the following fixed-point:

$$\Delta_1 = \left\{\begin{array}{ll|c}
S & : \bigwedge\Big(\{(q_1, B), (q_2, B)\} \cup \Theta_1\Big) \to (q_1, B) & B \in \{\mathbf{a}, \mathbf{b}, \bot\} \\
F_a & : \bigwedge\Big(\{(q_2, B)\} \cup \Theta_1\Big) \to (q_1, B\mathbf{a}) & f(\Theta_1) \subseteq \{(q, \star)\} \\
F_b & : \bigwedge\Big(\{(q_1, B\mathbf{b}), (q_2, B\mathbf{b})\} \cup \Theta_2\Big) \to (q_1, B) & f(\Theta_2) \subseteq \{(q, \star\star)\}
\end{array}\right\}.$$

Δ_1 is an invariant of $\mathcal{R}$ and contains $S : \iota_{\mathcal{A}_1}$. So Δ_1 is a witness and returned by **Refine**. □

We show the correctness and termination of **Refine**.

Lemma 4. *Let M_1 be a PDA. Given a finite environment Δ_1, a term t and a finite type τ, whether $\Delta_1 \vdash^{\Uparrow}_{M_1} t : \tau$ is decidable.*

Proof. Induction on the structure of t. □

Lemma 5. *Let (M_1, ι_1) and (M_2, ι_2) be PDAs with the initial symbols, $f : (M_1, \iota_1) \to (M_2, \iota_2)$ be a homomorphism and Δ_2 be a finite type environment in $\mathcal{T}_2$. Then the type environment Δ_1^0 defined in Fig. 1 is finite.*

Proof. We first show that the following two propositions hold for any finite type τ_2 by induction on τ_2: (i) for any type τ_1 in $\mathcal{T}_1$, $f(\tau_1) \sqsubseteq \tau_2$ implies finiteness of τ_1 and (ii) the set $\{\tau_1 \mid f(\tau_1) \sqsubseteq \tau_2\}$ is a finite set. Since there are finitely many type bindings in Δ_2, propositions (i) and (ii) imply finiteness of Δ_1^0. □

Theorem 5. *Let $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG, (M_1, ι_1) and (M_2, ι_2) be PDAs with the initial types, $f : (M_1, \iota_1) \to (M_2, \iota_2)$ be a homomorphism and Δ_2 be a finite witness of $\vdash^{\Uparrow}_{M_2} (\mathcal{R}, S) : \iota_2$. Then* **Refine**$(\mathcal{G}, (M_1, \iota_2), (M_2, \iota_2), f, \Delta_2)$ *always terminates, and returns a witness of $\vdash^{\Uparrow}_{M_1} (\mathcal{R}, S) : \iota_1$ if and only if it exists.*

Proof. First, we show the termination of the step 2 in Figure 1. It is easy to show that Δ_1^n is a finite type environment by induction on n (for the base case, we use Lemma 5). Thus Lemma 4 implies that we can compute $\mathcal{H}(\Delta_1^n)$. Since $\mathcal{H}$ is decreasing with respect to the set inclusion ordering, i.e., $\mathcal{H}(\Delta_1) \subseteq \Delta_1$ for any environment Δ_1, and Δ_1^0 is a finite set, the fixed-point iteration must terminate. So the procedure **Refine** terminates.

Let Δ_1' be a witness of $\vdash^{\Uparrow}_{M_1} (\mathcal{R}, S) : \iota_1$. Theorem 4 ensures that we can assume without loss of generality that $f(\Delta_1') \sqsubseteq \Delta_2$. Thus $\Delta_1' \subseteq \Delta_1^0$ because Δ_1^0 is the set of all refinement type bindings. By induction on n, we have $\Delta_1' = \mathcal{H}^n(\Delta_1') \subseteq \mathcal{H}^n(\Delta_1^0) = \Delta_1^n$, since Δ_1' is a fixed-point of $\mathcal{H}$ and $\mathcal{H}$ is monotonic. So $S : \iota_1 \in \Delta_1^n$ for any n, especially $S : \iota_1 \in \Delta_1$. □

5 Applications: Some Decidability Results

5.1 Balanced Parenthesis and Regular Hedge Languages

Let Σ be an alphabet. We define a PDA $\mathcal{B} = (\{q\}, \acute{\Sigma} \cup \grave{\Sigma}, \Sigma \cup \{\bot\}, \delta)$, where $\delta = \{(q, A, \acute{a}, q, Aa) \mid A \in \Sigma \cup \{\bot\}, a \in \Sigma\} \cup \{(q, a, \grave{a}, q, \varepsilon) \mid a \in \Sigma\}$ with the initial type $\iota_{\mathcal{B}} = (q, \bot) \to (q, \bot)$. Then $\mathcal{L}_{\mathcal{B}}(\iota_{\mathcal{B}})$ is the set of all balanced tags. For example, $\acute{a}\acute{b}_1\grave{b}_1\acute{b}_2\grave{b}_2\grave{a} \in \mathcal{L}_{\mathcal{B}}(\iota_{\mathcal{B}})$ and $\acute{b}_1\grave{b}_2 \notin \mathcal{L}_{\mathcal{B}}(\iota_{\mathcal{B}})$, where $a, b_1, b_2 \in \Sigma$. It is known that, for a given CFG $\mathcal{G}$, whether $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_{\mathcal{B}}$ is decidable. Moreover, if $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_{\mathcal{B}}$, we can construct a finite type environment Δ such that $\Delta \vdash^{\Uparrow}_{\mathcal{B}} (\mathcal{R}, S) : \iota_{\mathcal{B}}$.

Assume that (M, ι) is a refinement of $(\mathcal{B}, \iota_{\mathcal{B}})$, i.e., there is $f : (M, \iota) \to (\mathcal{B}, \iota_{\mathcal{B}})$. Then we can decide $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_M$ in the following way. First, we decide whether $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_{\mathcal{B}}$. If not, then $\mathcal{L}_{\mathcal{G}} \not\subseteq \mathcal{L}_M$ by Lemma 3. If $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_{\mathcal{B}}$, we construct a finite witness Δ and call **Refine**$(\mathcal{G}, (M, \iota), (\mathcal{B}, \iota_{\mathcal{B}}), f, \Delta)$.

This argument leads to the following decidability result.

Theorem 6. *Let $\mathcal{G}$ be a CFG and M be a refinement of $\mathcal{B}$. Then $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_M(\iota)$ is decidable.* $\square$

We have the following theorem for the class of refinements of $\mathcal{B}$.

Theorem 7. *A language is accepted by a refinement of $\mathcal{B}$ if and only if it is a regular hedge language [14].*

Proof. It is easy to prove using an algebraic representation of a regular hedge language, called binoid [12,18]. $\square$

The above argument therefore gives a new definition of the class of regular hedge languages and a new decidability proof of the inclusion problem between CFLs and regular hedge languages.

5.2 Counting Automata and Superdeterministic Languages

We define the class of PDAs named $\mathcal{C}$-machines.

Definition 4. *A PDA (M, ι_M) with the initial type is called a* $\mathcal{C}$-machine *if its stack alphabet is singleton and ι_M is finite.* $\square$

A configuration of a $\mathcal{C}$-machine is expressed by a pair (q, n) of a state q and a natural number n representing the length of the stack sequence. We define the stack extension $\Uparrow m$ for $\mathcal{C}$-machines by $(q, n) \Uparrow m = (q, n + m)$ and $(\bigwedge \Theta \to c) \Uparrow m = \bigwedge\{d \Uparrow m \mid d \in \Theta\} \to (c \Uparrow m)$.

Theorem 8. *For a given CFG $\mathcal{G}$ and $\mathcal{C}$-machine (M, ι_M), whether $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_M(\iota_M)$ is decidable. Moreover, when $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_M(\iota_M)$, we can construct a finite type environment Δ such that $\Delta \vdash^{\Uparrow}_M (\mathcal{R}, S) : \iota_M$.*

Proof. We give a proof sketch: See the full version for more detail. For simplicity, we assume that $\iota_M = c_E \to c_S$. Let $c_E = (q_E, n_E)$ and $c_S = (q_S, n_S)$. Let N be a finite-state automaton obtained by removing the counter of M, i.e., $q \vDash^a_N p$ if and only if $(q, n) \vDash^a_M (p, m)$ for some n and m. Roughly speaking, N is an "approximation" of M. So we can "refine" a witness in $\mathcal{T}_N$ to a witness in $\mathcal{T}_M$. Since N is finite-state, we can decide whether $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_N(q_E \to q_S)$. If not, then $\mathcal{L}_{\mathcal{G}} \nsubseteq \mathcal{L}_M(\iota_M)$. Assume $\mathcal{L}_{\mathcal{G}} \subseteq \mathcal{L}_N(q_E \to q_S)$ and let Δ_N be the minimum witness of $\vdash_N (\mathcal{R}, S) : q_E \to q_S$ (here $\mathcal{T}_N$ is the type system whose base types are states of N, instead of configurations).

For a given type binding $F : \bigwedge\{q_1, \ldots, q_m\} \to q \in \Delta_N$, we construct a corresponding type binding in $\mathcal{T}_M$. Since Δ_N is minimum, from the construction of the minimum witness (see the proof of Lemma 1), we have $w \in \mathrm{pre}(F)(= \{v \mid \exists u.\ S\$ \Rightarrow^* vFu\$\})$ and $w_i \in \mathcal{L}_{\mathcal{G}}(F)$ $(1 \leq i \leq m)$ such that $q_S \vDash^w_N q$ and $q \vDash^{w_i}_N q_i$ for all i (a different choice of w and w_i gives a different upper-bound of witnesses). We define n and n_i by $(q_E, n_E) \vDash^w_M (q, n)$ and $(q, n) \vDash^{w_i}_M (q_i, n_i)$. Then the corresponding type binding is $F : \bigwedge\{(q_1, n_1), \ldots, (q_m, n_m)\} \to (q, n)$.

Let Δ'_M be the type environment collecting such type bindings. We define $\Delta_M = \{F : \tau \mid \exists \sigma, k.\ F : \sigma \in \Delta'_M \text{ and } \tau \Uparrow k = \sigma\}$. Then Δ_M gives an upper-bound in the sense that if a witness of $\vdash^{\Uparrow}_M (\mathcal{R}, S) : \iota_M$ exists, then a witness included by Δ_M exists. □

Similarly to the argument in the previous subsection, Theorem 8 leads to the following decidability result.

Theorem 9. *For a given context-free grammar $\mathcal{G}$ and a pushdown automaton M which is a refinement of a $\mathcal{C}$-machine N, whether $\mathcal{L}_\mathcal{G} \subseteq \mathcal{L}_M$ is decidable.* □

The class of refinements of $\mathcal{C}$-machines is closely related to the class of *superdeterministic pushdown automata* proposed by Greibach and Friedman [5].

Definition 5 (Superdeterministic PDAs [5]). *A pushdown automaton M is of* delay *d if for any series of one-step transitions by ε, its length is less than or equal to d, i.e., if $c_0 \Vdash^{\varepsilon}_M c_1 \Vdash^{\varepsilon}_M \cdots \Vdash^{\varepsilon}_M c_n$ then $n \leq d$. A pushdown automaton $M(\iota)$ is* superdeterministic *if it satisfies the following properties: (1) M is of delay d for some finite number d, (2) if $(q, \widetilde{A}_1) \vDash^w_M (p_1, \widetilde{B}_1)$ and $(q, \widetilde{A}_2) \vDash^w_M (p_2, \widetilde{B}_2)$, then $p_1 = p_2$ and $|\widetilde{B}_1| - |\widetilde{A}_1| = |\widetilde{B}_2| - |\widetilde{A}_2|$, here $|\widetilde{A}|$ is the length of A, and (3) ι is finite. A language $\mathcal{L}$ is superdeterministic if $\mathcal{L} = \mathcal{L}_M$ for some superdeterministic pushdown automaton M.* □

The class of refinements of $\mathcal{C}$-machines and of superdeterministic PDAs are incomparable as classes of PDAs. However, they are equally expressive in the sense that the class of languages accepted by refinements of $\mathcal{C}$-machines is equivalent to the one accepted by superdeterministic PDAs.

Theorem 10. *A language is superdeterministic if and only if it is accepted by a refinement of a $\mathcal{C}$-machine.*

Proof. We give a proof sketch. We first prove the right-to-left direction. A state q of $\mathcal{C}$-machine C has a ε-loop if there is a sequence of ε-transitions starting from and ending with q, i.e., $(q, n) \Vdash^{\varepsilon}_C \cdots \Vdash^{\varepsilon}_C (q, m)$ for some n and m. By removing states which have ε-loops, we can construct an equivalent $\mathcal{C}$-machine that is of finite delay. Similarly, we can assume without loss of generality that any refinement of a $\mathcal{C}$-machine is of finite delay. Consider condition (2) in Definition 5. The condition on the stack length must be satisfied by all refinements of $\mathcal{C}$-machines, but the condition on the state may not in general. However we can always construct another refinement that satisfies the condition by moving the refined state information to the stack top, i.e., instead of refining a configuration of the $\mathcal{C}$ machine (q, n) to $(q', A_1 \ldots A_n)$, refining it to $(q, \langle A_1, q_1\rangle \ldots \langle A_n, q'\rangle)$. So for all refinements of $\mathcal{C}$-machines, we can construct another refinement which is superdeterministic and accepts the same language.

For the other direction, let M be a superdeterministic PDA and d be its delay. Note that for any configuration $(q, \widetilde{B} A_{d+1} \ldots A_1)$, only $d + 1$ stack symbols at the top (i.e., $A_{d+1} \ldots A_1$) affect a transition $(q, \widetilde{B}\widetilde{A}) \vDash^a_M (q', \widetilde{B}\widetilde{C})$. So we can construct another superdeterministic PDA M', whose transition coincides with the transition of M and is normalized as follows:

$$\begin{aligned}(q, \widetilde{B}\widetilde{A}) &\Vdash^{a}_{M'} (\langle q, a\rangle, \widetilde{B}\widetilde{A}) \\ &\Vdash^{\varepsilon}_{M'} (\langle q, a, A_1\rangle, \widetilde{B}A_{d+1} \dots A_2) \\ &\quad\vdots \\ &\Vdash^{\varepsilon}_{M'} (\langle q, a, \widetilde{A}\rangle, \widetilde{B}) \\ &\Vdash^{\varepsilon}_{M'} (q', \widetilde{B}\widetilde{C}).\end{aligned}$$

In the first stage of the transition, M' records a on its state, pops its stack d times and records them on the state. Then the state is a triple of the form $\langle q, a, \widetilde{A}\rangle$. In the last stage, M' computes q' and $\widetilde{C}$ from its state $\langle q, a, \widetilde{A}\rangle$. See the full version for more details about the construction of M'.

Let $\natural(\cdot)$ be a mapping which forgets stack symbols such as

$$\natural((\langle q, a, A_n \dots A_1\rangle, B_m \dots B_1)) = (\langle q, a, n\rangle, m).$$

The mapping $\natural(\cdot)$ and the transition relation δ of M induces a transition relation $\natural(\delta)$ of some $\mathcal{C}$-machine, which is an approximation of M. Condition (2) in Definition 5 ensures that $\natural(\delta)$ is deterministic. □

The decidability of the inclusion problem between context-free languages and superdeterministic languages has been proved by Greibach and Friedman [5]. The proof of Theorem 9 with Theorem 10 is an alternative and arguably simpler proof of the result.

6 Related Work

There have been a number of studies on the inclusion problems for subclasses of context-free languages (see [3] for a survey).

One of the strongest decidability results is about the inclusion between context-free languages and superdeterministic languages, proved by Greibach and Friedman [5]. Nguyen and Ogawa [15] gave a new proof by simplifying the technique used in [5]. Greibach and Friedman [5] reduced the problem to the emptiness problem for a pushdown automaton and Nguyen and Ogawa [15] gave simpler construction of a pushdown automaton.

Minamide and Tozawa [12] have proposed an algorithm for inclusion between context-free languages and regular hedge languages, motivated by the validation of dynamically generated HTML documents. As demonstrated in Section 5.1, our method gives an alternative algorithm for the same problem, although our algorithm may not be as efficient as Minamide and Tozawa's. Møller and Schwarz [13] have developed an algorithm to validate a context-free grammar against SGML DTDs, dealing with tag omissions and exceptions. It is not clear whether our method can provide a similar result.

The subclass of the context-free languages named *visibly pushdown languages* [1,2] has many good properties such as boolean closure and decidability of the emptiness problem in polynomial time. Some researchers have extended the class preserving such properties. Caucal [4] has introduced a notion of *synchronized pushdown automata* and Nowotka and Srba [16] have proposed *height-deterministic pushdown automata*. The refinement of a counter machine is similar to those notions. Since the class of visibly pushdown automata can be defined as the class of refinements of a certain automaton, our notion of refinements may give an extension of them.

Recently, type-based approaches to model-checking, verification and language inclusion problems have been extensively studied [7,8,9,11,19,20]. Kobayashi and Ong [7,9] have proposed a type system for recursion schemes that is equivalent to the modal μ-calculus model-checking of recursion schemes (the decidability of the model-checking problem has been proved by Ong [17]). These type systems have been applied to verification of higher-order programs [7,11,10], and practically effective typability checkers have been developed [6,8]. The present work extends type systems to deal with infinite state systems, namely deterministic pushdown automata. Types are now configurations of pushdown automata, rather than states of automata, which are finite a priori.

In our previous work [20], we gave a type-based proof for the inclusion problem between context-free languages and superdeterministic languages. But the proof is specific to superdeterministic languages, and difficult to generalize.

7 Conclusion and Future Work

We have proposed an intersection type system characterizing the inclusion by a deterministic context-free language, and given a sufficient condition of decidability of its typability. Future work includes extensions in two directions, extending grammars and automata. A naive extension to higher-order recursion schemes fails to establish the counterpart of Theorem 4. That is because the up-to technique used in this paper is too crude to deal with them. To extend automata is easier than grammars. For example, we can develop a framework for higher-order pushdown automata. So what we should do is to find a language accepted by a higher-order pushdown automaton which has decidable inclusion problem and a practical use.

Acknowledgement. The authors would like to thank the anonymous reviewers for their valuable comments. This work is partially supported by Kakenhi 23220001 and 22·3842.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) STOC, pp. 202–211. ACM (2004)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3), 1–43 (2009)

3. Asveld, P.R.J., Nijholt, A.: The inclusion problem for some subclasses of context-free languages. Theor. Comput. Sci. 230(1-2), 247–256 (2000)
4. Caucal, D.: Synchronization of Pushdown Automata. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 120–132. Springer, Heidelberg (2006)
5. Greibach, S.A., Friedman, E.P.: Superdeterministic PDAs: A subcase with a decidable inclusion problem. J. ACM 27(4), 675–700 (1980)
6. Kobayashi, N.: Model-checking higher-order functions. In: Porto, A., López-Fraguas, F.J. (eds.) PPDP, pp. 25–36. ACM (2009)
7. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: Shao, Z., Pierce, B.C. (eds.) POPL, pp. 416–428. ACM (2009)
8. Kobayashi, N.: A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order Recursion Schemes. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 260–274. Springer, Heidelberg (2011)
9. Kobayashi, N., Ong, C.H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: LICS, pp. 179–188. IEEE Computer Society (2009)
10. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Hall, M.W., Padua, D.A. (eds.) PLDI, pp. 222–233. ACM (2011)
11. Kobayashi, N., Tabuchi, N., Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification. In: Hermenegildo, M.V., Palsberg, J. (eds.) POPL, pp. 495–508. ACM (2010)
12. Minamide, Y., Tozawa, A.: XML Validation for Context-Free Grammars. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 357–373. Springer, Heidelberg (2006)
13. Møller, A., Schwarz, M.: HTML Validation of Context-Free Languages. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 426–440. Springer, Heidelberg (2011)
14. Murata, M.: Hedge automata: a formal model for XML schemata (1999), http://www.xml.gr.jp/relax/hedge_nice.html
15. Nguyen, V.T., Ogawa, M.: Alternate stacking technique revisited: Inclusion problem of superdeterministic pushdown automata. IPSJ Transactions on Programming 1(1), 36–46 (2008)
16. Nowotka, D., Srba, J.: Height-Deterministic Pushdown Automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 125–134. Springer, Heidelberg (2007)
17. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS, pp. 81–90. IEEE Computer Society (2006)
18. Pair, C., Quéré, A.: Définition et etude des bilangages réguliers. Information and Control 13(6), 565–593 (1968)
19. Tsukada, T., Kobayashi, N.: Untyped Recursion Schemes and Infinite Intersection Types. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 343–357. Springer, Heidelberg (2010)
20. Tsukada, T., Kobayashi, N.: A type-theoretic proof of the decidability of the language containment between context-free languages and superdeterministic languages. IPSJ Transactions on Programming 4(2), 31–47 (2011) (in Japanese)

An Output-Based Semantics of $\Lambda\mu$ with Explicit Substitution in the π-Calculus
Extended Abstract

Steffen van Bakel and Maria Grazia Vigliotti

Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, UK
{s.vanbakel,maria.vigliotti}@imperial.ac.uk

Abstract. We study the $\Lambda\mu$-calculus, extended with explicit substitution, and define a compositional *output*-based translation into a variant of the π-calculus with pairing. We show that this translation preserves single-step explicit head reduction with respect to contextual equivalence. We use this result to show operational soundness for head reduction, adequacy, and operational completeness. Using a notion of implicative type-context assignment for the π-calculus, we also show that assignable types are preserved by the translation. We finish by showing that termination is preserved.

1 Introduction

Over the last two decades, the π-calculus [24] and its dialects have proven to give an interesting and expressive model of computation. Encodings of variants of the pure λ-calculus [14, 11] started with [24], which quickly led to more thorough investigations in [29, 31, 10] and also in the direction of object oriented calculi [21, 31].

For these encodings, over the years strong properties have been shown like soundness, completeness, termination, and full abstraction. The strength of these results has encouraged the investigation of encodings into the π-calculus of calculi that have their foundation in classical logic, as done in, for example, [22, 8, 15]. From these papers it might seem that the encoding of such calculi comes at a great price; for example, to encode typed $\lambda\mu$ [25], [22] needs to consider a version of the π-calculus that is not only strongly typed, but, moreover, allows reduction under guard and under replication; [8] shows preservation of reduction in $\mathcal{X}$ [9] only with respect to $\sqsubseteq_{\mathrm{C}}$, the contextual ordering; [15] defines a non-compositional encoding of $\overline{\lambda}\mu\tilde{\mu}$ [17] that strongly depends on recursion, and does not regard the logical aspect at all.

In this paper, we will show that it *is* possible to define a intuitive, natural, logical encoding of $\lambda\mu$ into the pure π-calculus that satisfies all the good properties. Although one could justifiably argue that calculi like $\mathcal{X}$ and $\overline{\lambda}\mu\tilde{\mu}$ are more expressive and, through their direct link to Gentzen's LK [18], more elegantly deal with negation and classical logic, they are also both symmetric in nature, which makes an accurate treatment in the π-calculus more intricate, as can be observed in [8, 15]. Moreover, as argued in [6, 5, 7], only for $\lambda\mu$ is it possible to define a filter semantics, which seems to strengthen the case for that calculus even more.

Reduction in $\lambda\mu$ is confluent and non-symmetric; in fact, the main reduction rule (and the only cause for non-termination, for example) is the β-reduction rule of the λ-calculus. In addition to that rule, $\lambda\mu$ has *structural* rules, where elimination takes place for a type that is not the type of the term itself, but rather for one that appears in one of the alternative conclusions of the shape $\alpha{:}A$, where the Greek variable is the name given to a sub-term. For the naming feature, $\lambda\mu$ adds $[\alpha]M$ to the syntax which expresses that α serves as a pointer to the term M, and pairs this with a notion of μ-abstraction $\mu\alpha.M$, which is used to redirect operands (terms) to those called α. It is this naming feature, together with the structural rules, that make $\lambda\mu$ difficult to reason over; this is reflected in [20] and [9], where the encoding of $\lambda\mu$ into $\overline{\lambda}\mu\tilde{\mu}$ and $\mathcal{X}$, respectively, does not respect normal reduction. In contrast, through our translation we will show that it is possible to give a process semantics for $\lambda\mu$ that very clearly shows that the *context switch* $\mu\alpha.[\beta]M$ is, essentially, just a variant of application.

For the construction of our translation, we will start with that defined in [10], that interprets terms under *output* rather than under *input*, by giving a name to the anonymous output of λ-terms; we will combine this with the inherent naming mechanism of $\lambda\mu$. To accurately define the notion of reduction that is modelled by our translation, we will define *untyped* $\Lambda\mu\mathbf{x}$, a version with explicit substitution [1, 12] of the $\Lambda\mu$-calculus [19], itself a variant of $\lambda\mu$, together with a notion of *explicit head reduction*[1], where reduction is also allowed under abstraction. We will define a new compositional semantic translation of $\Lambda\mu\mathbf{x}$ into the π-calculus, and show that it fully respects each individual explicit head reduction step.

Perhaps surprisingly, we do not need to extend the kind of process calculus at all to accomodate our translation, but can build that directly on the standard π-calculus; in particular, the naming and μ-binding features of $\lambda\mu$ are dealt with by the naming feature of the translation, and renaming, respectively. The only noteworthy change is that, when representing application MN, the communication needs to be replicated; the translation of application and structural substitution is almost identical.

The advantage of considering explicit substitution rather than the standard implicit substitution as considered in [31] has been strongly argued in [10]. That paper showed that communication in the π-calculus has a fine semantic level of granularity that 'faithfully mimics' explicit substitution, and not the implicit one; we stress this point again with the results presented in this paper.

2 The $\Lambda\mu$ Calculus

The $\lambda\mu$-calculus is a proof-term syntax for classical logic, expressed in Natural Deduction, defined as an extension of the Curry type assignment system for the λ-calculus; we focus on de Groote's $\Lambda\mu$, a variant that splits the naming from the μ-binding. We will define in particular $\Lambda\mu\mathbf{x}$, a variant of $\Lambda\mu$ with explicit substitution *à la* $\lambda\mathbf{x}$ [12], and show our results for $\Lambda\mu\mathbf{x}$; since $\Lambda\mu\mathbf{x}$ implements $\Lambda\mu$-reduction, this implies that we also show some of our results for normal reduction (with implicit substitution).

[1] Called *spine reduction* in [10], and *head spine-reduction* in [32]; we prefer to use the terminology *head reduction* from [33].

Definition 1 (Syntax of $\Lambda\mu$). The *$\Lambda\mu$-terms* we consider are defined over the set of *term variables* represented by Roman characters, and *names*, or *context* variables, represented by Greek characters, through the grammar:

$$\begin{array}{rccccccccc} M,N ::= & x & \mid & \lambda x.M & \mid & MN & \mid & \mu\alpha.M & \mid & [\beta]M \\ & \textit{variable} & & \textit{abstraction} & & \textit{application} & & \textit{context abstraction} & & \textit{naming} \end{array}$$

The notion of free and bound names is defined as can be expected, taking both λ and μ as binders, and we assume Barendregt's convention.

Simple type assignment for $\Lambda\mu$ is defined as follows:

Definition 2 (Types, Contexts, and Typing). 1. Types are defined by:

$$A,B ::= \varphi \mid \bot \mid A{\to}B \quad (A \neq \bot)$$

where φ is a basic type of which there are infinitely many.

2. A *context of inputs* Γ is a mapping from term variables to types, denoted as a finite set of *statements* $x{:}A$, such that the *subjects* of the statements (x) are distinct. We write Γ_1,Γ_2 for the *compatible* union of Γ_1 and Γ_2 (if $x{:}A_1 \in \Gamma_1$ and $x{:}A_2 \in \Gamma_2$, then $A_1 = A_2$), and write $\Gamma,x{:}A$ for $\Gamma,\{x{:}A\}$.
3. Contexts of *outputs* Δ, and the notions Δ_1,Δ_2 and $\alpha{:}A,\Delta$ are defined similarly.
4. Type assignment for $\Lambda\mu$ is defined by the following natural deduction system.

$$(Ax): \frac{}{\Gamma,x{:}A \vdash x:A \mid \Delta} \quad (\mu): \frac{\Gamma \vdash M:\bot \mid \alpha{:}A,\Delta}{\Gamma \vdash \mu\alpha.M:A \mid \Delta} \quad (\bot): \frac{\Gamma \vdash M:A \mid \beta{:}A,\Delta}{\Gamma \vdash [\beta]M:\bot \mid \beta{:}A,\Delta}$$

$$({\to}I): \frac{\Gamma,x{:}A \vdash M:B \mid \Delta}{\Gamma \vdash \lambda x.M : A{\to}B \mid \Delta} \quad ({\to}E): \frac{\Gamma \vdash M:A{\to}B \mid \Delta \quad \Gamma \vdash N:A \mid \Delta}{\Gamma \vdash MN:B \mid \Delta}$$

In $\Lambda\mu$, reduction of terms is expressed via implicit substitution; as usual, $M[N/x]$ stands for the substitution of all occurrences of x in M by N, and $M[N{\cdot}\gamma/\alpha]$, the *structural substitution*, stands for the term obtained from M in which every sub-term of the form $[\alpha]M'$ is replaced by $[\gamma](M'N)$.

We have the following rules of computation in $\lambda\mu$:

Definition 3 ($\Lambda\mu$ reduction). $\Lambda\mu$ has two *computational rules*:

$$\begin{array}{rrcll} \textit{logical}\ (\beta): & (\lambda x.M)N & \to & M[N/x] & \\ \textit{structural}\ (\mu): & (\mu\alpha.M)N & \to & \mu\gamma.M[N{\cdot}\gamma/\alpha] & \gamma\ \textit{fresh} \end{array}$$

as well as the *simplification rules*:

$$\begin{array}{rrcl} \textit{renaming}: & \mu\alpha.[\beta]\mu\gamma.M & \to & \mu\alpha.M[\beta/\gamma] \\ \textit{erasing}: & \mu\alpha.[\alpha]M & \to & M \quad \textit{if } \alpha \textit{ does not occur in } M. \end{array}$$

(which are added mainly to simplify the presentation of results), and the contextual rules. We use $\to_{\beta\mu}$ for this reduction, and $\to^*_{\beta\mu}$ for its reflexive and transitive closure.

[26] has shown that typeable terms are strongly normalisable. It also defines extensional rules, that we do not consider here: the model we present through our translation is not extensional, and we can therefore not show that those rules are preserved by the translation. That this notion of reduction is confluent was shown in [28].

3 The Synchronous π-Calculus with Pairing

The notion of π-calculus that we consider in this paper is similar to the one used also in [2], and is different from other systems studied in the literature [21] in that it adds pairing, and uses a *let*-construct to deal with inputs of pairs of names that get distributed.

As already argued in [10], the main reason for the addition of pairing [2] lies in preservation of (implicate, or functional) type assignment; therefore *data* is introduced as a structure over names, such that not only names but also pairs of names can be sent.

Definition 4 (Processes). *Channel names* and *data* are defined by:

$$a,b,c,d,x,y,z \quad \textit{names} \qquad\qquad p ::= a \mid \langle a,b\rangle \quad \textit{data}$$

Notice that pairing is *not* recursive. Processes are defined by:

$$P,Q ::= 0 \mid P|Q \mid\, !P \mid (\nu a)\,P \mid a(x).P \mid \overline{a}\langle p\rangle\,.\,P \mid \mathit{let}\,\langle x,y\rangle = p \;\mathit{in}\; P$$

A *context* $\mathsf{C}[\cdot]$ is a process with a hole $[\,]$; we call $a\,(x)$ and $\overline{a}\langle p\rangle$ *guards*, and call P in $a(x).\,P$ and $\overline{a}\langle p\rangle\,.\,P$ a process *under guard*.

We abbreviate $a\,(x).\,\mathit{let}\,\langle y,z\rangle = x \;\mathit{in}\; P$ by $a\,(y,z).\,P$, as well as $(\nu m)\,(\nu n)\,P$ by $(\nu mn)\,P$, and write $\overline{a}\langle p\rangle$ for $\overline{a}\langle p\rangle\,.\,0$, and $\overline{a}\langle c,d\rangle.P$ for $\overline{a}\langle\langle c,d\rangle\rangle\,.\,P$. Notice that $\mathit{let}\,\langle x,y\rangle = a \;\mathit{in}\; P$ (where a is not a variable) is *stuck*.

Definition 5 (Congruence). The structural congruence is the smallest equivalence relation closed under contexts defined by the following rules:

$$\begin{array}{rcl@{\qquad}rcll}
P \mid 0 & \equiv & P & (P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & \\
P \mid Q & \equiv & Q \mid P & (\nu n)\,0 & \equiv & 0 & \\
!P & \equiv & P \mid\, !P & (\nu m)\,(\nu n)\,P & \equiv & (\nu n)\,(\nu m)\,P & \\
!P & \equiv & !P \mid\, !P & (\nu n)\,(P \mid Q) & \equiv & P \mid (\nu n)\,Q & \textit{if } n \notin \mathit{fn}(P) \\
 & & & \mathit{let}\,\langle x,y\rangle = \langle a,b\rangle \;\mathit{in}\; P & \equiv & P[a/x,b/y] &
\end{array}$$

As usual, we will consider processes modulo congruence and modulo α-convergence: this implies that we will not deal explicitly with the process $\mathit{let}\,\langle x,y\rangle = \langle a,b\rangle \;\mathit{in}\; P$, but rather with $P[a/x,b/y]$. We write $a \rightarrow b$ for the *forwarder* [31] $a\,(x).\,\overline{b}\langle x\rangle$.

Computation in the π-calculus with pairing is expressed via the exchange of *data*.

Definition 6 (Reduction). The *reduction relation* over the processes of the π-calculus is defined by the following (elementary) rules:

$$\begin{array}{rcl}
\overline{a}\langle p\rangle\,.\,P \mid a(x).Q & \to_\pi & P \mid Q[p/x] \\
P \to_\pi P' & \Rightarrow & (\nu n)\,P \to_\pi (\nu n)\,P' \\
P \to_\pi P' & \Rightarrow & P \mid Q \to_\pi P' \mid Q \\
P \equiv Q \;\&\; Q \to_\pi Q' \;\&\; Q' \equiv P' & \Rightarrow & P \to_\pi P'
\end{array}$$

As usual, we write $\to_\pi^+$ for the transitive closure of $\to_\pi$, and $\to_\pi^*$ for its reflexive and transitive closure; we write $\to_\pi(a)$ if we want to point out that a synchronisation took place over channel a, and write $\to_\pi(=_\alpha)$ if we want to point out that α-conversion has taken place during the synchronisation.

Notice that $\overline{a}\langle b,c\rangle \mid a(x,y).Q \;\to_\pi\; Q[b/x,c/y]$.

Definition 7. *1. We write $P \downarrow n$ and say that P* outputs on *n (or P exhibits an output barb on n) if $P \equiv (\nu b_1) \ldots b_m(\overline{n}\langle p\rangle \mid Q)$ for some Q, where $n \neq b_1 \ldots b_m$.*

2. We write $P \Downarrow n$ (P may output on *n) if there exists Q such that $P \rightarrow^*_\pi Q$ and $Q \downarrow n$.*

3. We write $P \sim_C Q$ (P and Q are contextually equivalent*) if, for all $C[\cdot]$, and for all n, $C[P] \Downarrow n$ if and only if $C[Q] \Downarrow n$.*

4. We write $\sim_G$ (called garbage collection*) when we ignore a process because it is contextually equivalent to 0; notice that $\sim_G \subset \sim_C$.*

The following is a well-known result.

Proposition 8. Let P, Q not contain a, then

$$\begin{array}{lll} (\nu a)\,(\overline{a}\langle b\rangle.P \mid a(x).Q) & \sim_C & P \mid Q[b/x] \\ (\nu a)\,(!\overline{a}\langle b\rangle.P \mid a(x).Q) & \sim_C, \sim_G & Q[b/x] \end{array}$$

The π-calculus is equipped with a rich type theory [31], from the basic type system for counting the arity of channels [27] to sophisticated linear types in [22]. The notion of type assignment we use here is the one first defined in [8] and differs from systems presented in the past in that types do not contain channel information, and in that it expresses *implication*, *i.e.* has functional types and describes the '*input-output interface*' of a process.

Definition 9 (Context assignment for π [8]). Functional type assignment for the π-calculus is defined by the following sequent system:

$$(0): \frac{}{0 : \Gamma \vdash \Delta} \quad (!): \frac{P : \Gamma \vdash \Delta}{!P : \Gamma \vdash \Delta} \quad (in): \frac{P : \Gamma, x{:}A \vdash x{:}A, \Delta}{a(x).P : \Gamma, a{:}A \vdash \Delta}$$

$$(\nu): \frac{P : \Gamma, a{:}A \vdash a{:}A, \Delta}{(\nu a)\,P : \Gamma \vdash \Delta} \quad (out): \frac{P : \Gamma, b{:}A \vdash b{:}A, \Delta}{\overline{a}\langle b\rangle.P : \Gamma, b{:}A \vdash a{:}A, b{:}A, \Delta}\,(a \neq b)$$

$$(\mid): \frac{P : \Gamma \vdash \Delta \quad Q : \Gamma \vdash \Delta}{P \mid Q : \Gamma \vdash \Delta} \quad (pair\text{-}out): \frac{P : \Gamma, b{:}A \vdash c{:}B, \Delta}{\overline{a}\langle b,c\rangle.P : \Gamma, b{:}A \vdash a{:}A{\rightarrow}B, c{:}B, \Delta}\,(b \notin \Delta; a, c \notin \Gamma)$$

$$(W): \frac{P : \Gamma \vdash \Delta}{P : \Gamma' \vdash \Delta'}\,(\Gamma' \supseteq \Gamma, \Delta' \supseteq \Delta) \quad (let): \frac{P : \Gamma, y{:}B \vdash x{:}A, \Delta}{let\,\langle x,y\rangle = z\ in\ P : \Gamma, z{:}A{\rightarrow}B \vdash \Delta}\,(y, z \notin \Delta; x \notin \Gamma)$$

We adjust the system for the type constant $\perp$ by allowing that only in right-hand contexts. We write $P : \Gamma \vdash_\pi \Delta$ if there exists a derivation using these rules that has this expression in the conclusion.

We should perhaps stress that it is not known if this system has a relation with logic.

The following rule is derivable:

$$(pair\text{-}in): \frac{P : \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}{a(x,y).\,P : \Gamma, a{:}A{\rightarrow}B \vdash_\pi \Delta}\,(y, a \notin \Delta,\ x \notin \Gamma)$$

The soundness result is stated as:

Theorem 10 (Witness reduction [8]). If $P : \Gamma \vdash_\pi \Delta$ and $P \rightarrow_\pi Q$, then $Q : \Gamma \vdash_\pi \Delta$.

4 Context and Background of This Paper

In the past, there have been several investigations of encoding from the λ-calculus [11] into the π-calculus [24, 29]. Research in this direction started by Milner's encoding $[\![\cdot]\!]^{\mathrm{M}}\cdot$ of λ-terms [24]; Milner's encoding is *input* based and the translation of closed λ-terms respects large-step *lazy* reduction $\rightarrow_{\mathrm{L}}$ [3] to normal form up to substitution. Standard operational soundness result hold for this translation, and full abstraction has been shown by in [29] for an (input-based, as Milner's) encoding $\mathcal{H}\lceil\cdot\rfloor\ \langle\cdot\rangle$, of the lazy λ-calculus into the higher-order π-calculus (where in synchronisation not names are sent, but processes).

In [10], we presented a *logical*, *output-based* translation $\lceil\cdot\rfloor^{\mathrm{S}}\cdot$ that interprets abstraction $\lambda x.M$ not using *input*, but via an asynchronous *output* which leaves the translation of the body M free to reduce. That translation is defined as:

$$\begin{array}{rcl}
\lceil x\rfloor^{\mathrm{S}} a & \triangleq & x(w).\overline{a}\langle w\rangle \\
\lceil \lambda x.M\rfloor^{\mathrm{S}} a & \triangleq & (\nu xb)\,(\lceil M\rfloor^{\mathrm{H}} b \mid \overline{a}\langle x,b\rangle) \\
\lceil MN\rfloor^{\mathrm{S}} a & \triangleq & (\nu c)\,(\lceil M\rfloor^{\mathrm{H}} c \mid c(v,d).\,(!\lceil N\rfloor^{\mathrm{H}} v \mid d\rightarrow a)) \\
\lceil M\langle x := N\rangle\rfloor^{\mathrm{S}} a & \triangleq & (\nu x)\,(\lceil M\rfloor^{\mathrm{H}} a \mid !\lceil N\rfloor^{\mathrm{H}} x)
\end{array}$$

For this translation, [10] showed (using $\uparrow$ to denote non-termination)

1. $M\uparrow \Rightarrow \lceil M\rfloor^{\mathrm{S}} a\uparrow$, and $M \rightarrow_{\mathrm{xH}} N \Rightarrow \lceil M\rfloor^{\mathrm{S}} a \rightarrow^{*}_{\pi}\sim_{\mathrm{C}} \lceil N\rfloor^{\mathrm{H}} a$.
2. $\Gamma \vdash M : A \Rightarrow \lceil M\rfloor^{\mathrm{S}} a : \Gamma \vdash_{\pi} a{:}A$.

As argued in [10], to show the above result, which formulates a direct *step-by-step* relation between β-reduction and the synchronisation in the π-calculus, it is necessary to make the substitution explicit. This is a direct result of the fact that, in the π-calculus, λ's implicit substitution gets 'implemented' *one variable at the time*, rather than all in one fell swoop. Since we aim to show a similar result for $\Lambda\mu$, we will therefore define a notion of explicit substitution. Although termination is not studied in that paper, it is easily achieved through restricting the notion of reduction in the π-calculus by not allowing reduction to take place inside processes whose output cannot be received, or by placing a guard on the replication as we do in this paper.

A natural extension of this line of research is to see if the π-calculus can be used to interpret more complex calculi as well, as for example calculi that relate not to intuitionistic logic, but to classical logic, as $\lambda\mu$, $\overline{\lambda}\mu\tilde{\mu}$, or $\mathcal{X}$. There are, to date, a number of papers on this topic. In [22] an interpretation of Call-by-Value $\lambda\mu$ is defined that is based on Milner's. The authors consider *typed processes only*, and use a much more liberal notion of reduction on processes by allowing reduction *under* guards, making the resulting calculus very different from the original π-calculus. Types for processes prescribe usage of names

In [8] an interpretation into π of the sequent calculus $\mathcal{X}$ is defined that enjoys the Curry-Howard isomorphism for Gentzen's LK [18], which is shown to respect reduction. However, this result is only partial, as it is formulated as "*if $P \rightarrow_{\mathcal{X}} Q$, then* $[\![P]\!]\ {}_{\mathrm{C}}\sqsupseteq\ [\![Q]\!]$", allowing $[\![P]\!]$ to have more observable behaviour than $[\![Q]\!]$. Although in [8] it is reasoned that this is natural in the context of non-confluent, symmetric sequent calculi, and is shown that the interpretation preserves types, it is a weaker result than could perhaps be expected.

An encoding of $\overline{\lambda}\mu\tilde{\mu}$ is studied in [15]; the interpretation defined there strongly depends on recursion, is not compositional, and preserves only outermost reduction; no relation with types is shown.

5 $\Lambda\mu$ with Explicit Substitution

One of the main achievements of [10] is that it establishes a strong link between reduction in the π-calculus and step-by-step *explicit substitution* for the λ-calculus, by formulating a result not only with respect to explicit head reduction and the spine encoding defined there, but also for Milner's encoding with respect to explicit lazy reduction.

In view of this, we decided to study a variant of $\Lambda\mu$ with explicit substitution as well, and present here $\Lambda\mu\mathbf{x}$. Explicit substitution treats substitution as a first-class operator, both for the logical and the structural substitution, and describes all the necessary steps to effectuate both.

Definition 11 (**$\Lambda\mu\mathbf{x}$**). 1. The syntax of the *explicit $\Lambda\mu$ calculus*, $\Lambda\mu\mathbf{x}$, is defined by:

$$M,N ::= x \mid \lambda x.M \mid MN \mid M\langle x := N\rangle \mid \mu\alpha.M \mid [\beta]M \mid M\langle \alpha := N\cdot\gamma\rangle$$

We call a term *pure* if it does not contain explicit substitution.

2. The reduction relation $\rightarrow_{\mathbf{x}}$ on terms in $\Lambda\mu\mathbf{x}$ is defined as the compatible closure of the rules (we only show the important ones):
   (a) Main reduction rules:
   $$\begin{array}{rcll} (\lambda x.M)N & \rightarrow & M\langle x := N\rangle & N\,\textit{pure} \\ (\mu\alpha.M)N & \rightarrow & \mu\gamma.M\langle \alpha := N\cdot\gamma\rangle & N\,\textit{pure} \\ \mu\beta.[\beta]M & \rightarrow & M & \textit{if}\ \beta \notin \textit{fn}(M) \\ \mu\beta.[\delta]\mu\gamma.M & \rightarrow & \mu\beta.M[\delta/\gamma] & \end{array}$$
   (b) Term substitution rules, like
   $$\begin{array}{rcll} x\langle x := N\rangle & \rightarrow & N & \\ M\langle x := N\rangle & \rightarrow & M & x \notin \textit{fv}(M) \end{array}$$
   (c) Structural rules, like
   $$\begin{array}{rcll} ([\alpha]M)\langle \alpha := N\cdot\gamma\rangle & \rightarrow & [\gamma](M\langle \alpha := N\cdot\gamma\rangle)N & \\ ([\beta]M)\langle \alpha := N\cdot\gamma\rangle & \rightarrow & [\beta](M\langle \alpha := N\cdot\gamma\rangle) & \alpha \neq \beta \\ M\langle \alpha := N\cdot\gamma\rangle & \rightarrow & M & \alpha \notin \textit{fn}(M) \end{array}$$
   (d) Contextual rules, like
   $$M \rightarrow N \;\Rightarrow\; \left\{\begin{array}{lcl} ML & \rightarrow & NL \\ LM & \rightarrow & LN \\ M\langle x := L\rangle & \rightarrow & N\langle x := L\rangle \\ L\langle \alpha := M\cdot\gamma\rangle & \rightarrow & L\langle \alpha := N\cdot\gamma\rangle \end{array}\right.$$
3. We define $\rightarrow_{:=}$ as the notion of reduction where the main reduction rules are not used, and $=_{\mathbf{x}}$ as the smallest equivalence relation generated by $\rightarrow_{\mathbf{x}}$.

Notice that this is a system different from that of [4], where a version with explicit substitution is defined for a variant of $\lambda\mu$ that uses de Bruijn indices [13].

Explicit substitution describes explicitly the process of executing a $\beta\mu$-reduction, *i.e.* expresses syntactically the details of the computation as a succession of atomic steps (like in a first-order rewriting system), where the implicit substitution of each $\beta\mu$-reduction step is split up into reduction steps. Thereby the following is straightforward:

Proposition 12 ($\Lambda\mu\mathbf{x}$ implements $\Lambda\mu$-reduction). 1. $M \rightarrow_{\beta\mu} N \Rightarrow M \rightarrow_{\mathbf{x}}^{*} N$.
2. $M \in \Lambda\mu \;\&\; M \rightarrow_{\mathbf{x}} N \Rightarrow \exists L \in \Lambda\mu\, [N \rightarrow_{:=}^{*} L]$.

The notion of type assignment on $\Lambda\mu\mathbf{x}$ is a natural extension of the system for the $\Lambda\mu$-calculus of Def. 2 by adding rules $(T\text{-}cut)$ and $(C\text{-}cut)$.

Definition 13. *Using the notion of type assignment in Def. 2, type assignment for $\Lambda\mu\mathbf{x}$ is defined by adding:*

$$(T\text{-}cut):\ \frac{\Gamma, x{:}A \vdash M : B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash M \langle x := N \rangle : B \mid \Delta}$$

$$(C\text{-}cut):\ \frac{\Gamma \vdash M : C \mid \alpha{:}A{\rightarrow}B, \gamma{:}B, \Delta \quad \Gamma \vdash N : A \mid \gamma{:}B, \Delta}{\Gamma \vdash M \langle \alpha := N{\cdot}\gamma \rangle : C \mid \Delta}$$

We write $\Gamma \vdash_{\mu\mathbf{x}} M : A$ for judgements derivable in this system.

We also consider the notion of head reduction;

Definition 14. *1. We define* head *reduction $\rightarrow_{\mathrm{H}}$ as a restriction of $\rightarrow_{\beta\mu}$ by removing the contextual rule $M \rightarrow N \;\Rightarrow\; LM \rightarrow LN$.*
2. The $\Lambda\mu$ and $\Lambda\mu\mathbf{x}$ head-normal forms are defined through the grammar:

$$\begin{array}{rl} \boldsymbol{H} ::= & xM_1 \cdots M_n \; (n \geq 0) \mid \lambda x.\boldsymbol{H} \mid [\alpha]\boldsymbol{H} \\ \mid & \mu\alpha.\boldsymbol{H} \;\; (\boldsymbol{H} \neq [\alpha]\boldsymbol{H}' \;\&\; \alpha \notin \boldsymbol{H}', \boldsymbol{H} \neq [\beta]\gamma.\boldsymbol{H}') \end{array}$$

3. The head variable *of M, $hv(M)$, and* head name *$hn(M)$ are defined as expected.*

The following is straightforward:

Proposition 15 ($\rightarrow_{\mathrm{H}}$ implements $\Lambda\mu$'s head reduction). If $M \rightarrow_{\beta\mu}^{*} N$ with N in head-normal form, then there exists L in $\rightarrow_{\mathrm{H}}$-normal form such that $M \rightarrow_{\mathrm{H}}^{*} L$, and $L \rightarrow_{\beta\mu}^{*} N$, and none of these last steps are reductions in $\rightarrow_{\mathrm{H}}$.

Notice that $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \;\rightarrow_{\mathrm{H}}\; \lambda f.f((\lambda x.f(xx))(\lambda x.f(xx)))$ and this last term is in head-normal form, and in $\rightarrow_{\mathrm{H}}$-normal form.

In the context of head reduction, we can economise further on how substitution is executed, and perform only those replacements of variables by terms that are essential for the continuation of reduction. We will therefore limit substitution to allow it to only replace the head variable or name of a term. We will show that this is exactly the kind of reduction that the π-calculus naturally encodes.

Definition 16 (Explicit head reduction cf. [10]). We define *explicit head reduction* $\rightarrow_{\mathbf{x}\mathrm{H}}$ on $\Lambda\mu\mathbf{x}$ as $\rightarrow_{\mathbf{x}}$, but for:

1. To avoid looping unnecessarily, application of all term substitution (resp. structural) rules on $M\langle x := N \rangle$ (resp. $M\langle \alpha := N{\cdot}\gamma \rangle$) is only allowed if $hv(M) = x$ (resp. $hn(M) = \alpha$); the only exception are the garbage collection rules, *i.e.* when $x \notin fv(M)$ $(\alpha \notin fn(M))$.

2. We change two cases:
$$\begin{array}{lll}(PQ)\langle x := N\rangle & \to (P\langle x := N\rangle\, Q)\langle x := N\rangle & (x = hv(P))\\ (PQ)\langle \alpha := N{\cdot}\gamma\rangle & \to (P\langle \alpha := N{\cdot}\gamma\rangle\, Q)\langle \alpha := N{\cdot}\gamma\rangle & (\alpha = hn(P))\end{array}$$
3. We add two substitution rules:
$$\begin{array}{rll}M\langle x := N\rangle\langle y := L\rangle & \to M\langle y := L\rangle\langle x := N\rangle\langle y := L\rangle & (y = hv(M))\\ M\langle \alpha := N{\cdot}\gamma\rangle\langle \beta := L{\cdot}\delta\rangle & \to M\langle \beta := L{\cdot}\delta\rangle\langle \alpha := N{\cdot}\gamma\rangle\langle \beta := L{\cdot}\delta\rangle & (\alpha = hn(P))\end{array}$$
4. We remove the contextual rules:
$$M \to N \Rightarrow \begin{cases} LM & \to LN\\ L\langle x := M\rangle & \to L\langle x := N\rangle\\ L\langle \alpha := M{\cdot}\gamma\rangle & \to L\langle \alpha := N{\cdot}\gamma\rangle\end{cases}$$

Notice that, for example, in case 2, the first of the two clauses postpones the substitution $\langle x := N\rangle$ on Q until such time that an occurrence of the variable x in Q becomes the head-variable. It is straightforward to show that this notion of reduction is confluent; remember that in $M\langle x := N\rangle$ and $M\langle \alpha := N{\cdot}\gamma\rangle$, N is a pure term.

The following proposition states the relation between explicit head reduction, head reduction, and explicit reduction.

Proposition 17. 1. If $M \to_{\text{H}}^{*} N$, then there exists L such that $M \to_{\mathbf{x}\text{H}}^{*} L$ and $N \to_{:=}^{*} L$.
2. If $M \to_{\text{H}}^{*} N$ and N is in $\to_{\text{H}}$-normal form, then there exists L such that $M \to_{\mathbf{x}\text{H}}^{*} L$ and $N \to_{\mathbf{x}}^{*} L$.
3. If $M \to_{\mathbf{x}\text{H}}^{*} N$ with $M \in \Lambda\mu$ and N is in $\to_{\mathbf{x}\text{H}}$-normal form, then there exists $L \in \Lambda\mu$ such that $N \to_{:=} L$, and L is in $\Lambda\mu$ head-normal form.

This result gives that we can show our main results for $\Lambda\mu\mathbf{x}$ for reductions that reduce to head-normal form, that are naturally defined as follows:

Definition 18 (cf. [23]). The normal forms with respect to $\to_{\mathbf{x}\text{H}}$ are defined through:
$$\begin{array}{rcll}\mathbf{N} & ::= & xM_1\cdots M_n\ (n \geq 0) \mid \lambda x.\mathbf{N} \mid [\alpha]\mathbf{N} & \\ & \mid & \mu\alpha.\mathbf{N} & (\mathbf{N} \neq [\alpha]\mathbf{N}'\ \&\ \alpha \notin \mathbf{N}', \mathbf{N} \neq [\beta]\gamma.\mathbf{N}')\\ & \mid & \mathbf{N}\langle x := M\rangle & (hv(\mathbf{N}) \neq x)\\ & \mid & \mathbf{N}\langle \alpha := M{\cdot}\gamma\rangle & (hn(\mathbf{N}) \neq \alpha)\end{array}$$

Notice that, for example, under head reduction, any term of the shape $(\lambda x.P)Q$ in one of the M_i in $xM_1\cdots M_n$ is *not* considered a redex.

6 A Logical Translation of $\Lambda\mu$x to π

We will now define our logical, output-based translation $\ulcorner\!\cdot\!\lrcorner\,\cdot$ of the $\Lambda\mu\mathbf{x}$-calculus into the π-calculus. The main idea behind the translation, as in [10], is to give a name to the anonymous output of terms; it combines this with the inherent naming mechanism of $\Lambda\mu$. In the definition below, for readability, we use the symbol $\bullet$ as a channel name to represent an output that cannot be received from.

Definition 19 (Logical translation of $\Lambda\mu\mathbf{x}$ terms). The translation of $\Lambda\mu\mathbf{x}$ terms into the π-calculus is defined in Fig. 1.

$$\begin{array}{rcl}
\ulcorner\!\! x \rrbracket\, a & \triangleq & x(u).\,!u\text{-}\!\!\!\twoheadrightarrow a \\
\ulcorner\!\! \lambda x.M \rrbracket\, a & \triangleq & (\nu xb)\,(\ulcorner\!\! M \rrbracket\, b \mid \overline{a}\langle x,b\rangle) \\
\ulcorner\!\! MN \rrbracket\, a & \triangleq & (\nu c)\,(\ulcorner\!\! M \rrbracket\, c \mid !c(v,d).\,(\ulcorner\!\! v := N \rrbracket \mid !d\text{-}\!\!\!\twoheadrightarrow a)) \\
\ulcorner\!\! M\,\langle x := N\rangle \rrbracket\, a & \triangleq & (\nu x)\,(\ulcorner\!\! M \rrbracket\, a \mid \ulcorner\!\! x := N \rrbracket\,) \\
\ulcorner\!\! x := N \rrbracket & \triangleq & !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\ulcorner\!\! N \rrbracket\, w) \\
\ulcorner\!\! \mu\gamma.M \rrbracket\, a & \triangleq & (\nu\bullet)\,((\ulcorner\!\! M \rrbracket\,\bullet)[a/\gamma]) \\
\ulcorner\!\! [\beta]M \rrbracket\, a & \triangleq & \ulcorner\!\! M \rrbracket\, \beta \\
\ulcorner\!\! M\,\langle \beta := N\cdot\gamma\rangle \rrbracket\, a & \triangleq & (\nu\beta)\,(\ulcorner\!\! M \rrbracket\, a \mid \ulcorner\!\! \beta := N\cdot\gamma \rrbracket\,) \\
\ulcorner\!\! \alpha := N\cdot\gamma \rrbracket & \triangleq & !\alpha(v,d).\,(\ulcorner\!\! v := N \rrbracket \mid !d\text{-}\!\!\!\twoheadrightarrow \gamma)
\end{array}$$

Fig. 1. The logical translation

We would like to stress that. although inspired by logic, our translation does not depend on types *at all*; in fact, we can treat untypeable terms as well, and can show that $\ulcorner\!\! (\lambda x.xx)(\lambda x.xx) \rrbracket\, a$ (perhaps the prototype of a non-typeable term) runs to itself (this already holds for $\ulcorner\!\! \cdot \rrbracket^{\mathrm{H}} \cdot$ of [10]).

Notice that, as is the case for Milner's translation and in contrast to the interpretation of [10], a guard is placed on the replicated terms. This is not only done with an eye on proving preservation of termination, but more importantly, to make sure that $(\nu x)\,(\ulcorner\!\! x := N \rrbracket\,) \not\sim_{\mathrm{C}} \mathit{0}$: since a term can have named sub-terms, the translation will generate output not only for the term itself, but also for those named terms, so $(\nu x)\,(\ulcorner\!\! x := N \rrbracket\,)$ *can have* observable behaviour, in contrast to [10], where this process *is* equivalent to $\mathit{0}$.

We could have avoided the implicit renaming in the case for μ-abstraction and defined $\ulcorner\!\! \mu\gamma.M \rrbracket\, a = (\nu\bullet\gamma)\,(\ulcorner\!\! M \rrbracket\,\bullet \mid !\gamma\text{-}\!\!\!\twoheadrightarrow a)$, which is operationally (contextually) the same as $(\nu\bullet)\,((\ulcorner\!\! M \rrbracket\,\bullet)[a/\gamma])$, but then we could not show that terms in head-normal form are translated to processes in normal form (Lem. 24). There is a strong relation between this encoding and the abstract machine defined in [16], but for the fact that that only represents lazy reduction.

Notice that $\ulcorner\!\! \mu\gamma.[\beta]M \rrbracket\, a \triangleq (\nu\bullet)\,((\ulcorner\!\! M \rrbracket\, \beta)[a/\gamma])$, so had we considered to just encode $\lambda\mu$, we could have defined

$$\ulcorner\!\! \mu\gamma.[\beta]M \rrbracket\, a \;\triangleq\; (\nu\bullet)\,((\ulcorner\!\! M \rrbracket\, \beta)[a/\gamma]) \;=\; \ulcorner\!\! M[a/\gamma] \rrbracket\, \beta$$

so $\lambda\mu$'s binding-and-naming has no representation in π.

Moreover, notice the similarity between

$$\begin{array}{c}
\ulcorner\!\! MN \rrbracket\, a = (\nu c)\,(\ulcorner\!\! M \rrbracket\, c \mid !c(v,d).\,(\ulcorner\!\! v := N \rrbracket \mid !d\text{-}\!\!\!\twoheadrightarrow a)) \\
\ulcorner\!\! M\,\langle \beta := N\cdot\gamma\rangle \rrbracket\, a = (\nu\beta)\,(\ulcorner\!\! M \rrbracket\, a \mid !\beta(v,d).\,(\ulcorner\!\! v := N \rrbracket \mid !d\text{-}\!\!\!\twoheadrightarrow \gamma))
\end{array}$$

The first communicates N via the output channel c of M, whereas the second communicates with all the sub-terms that have β as its output name[2]. This very elegantly expresses exactly what the structural substitution does: it 'connects' arguments with the correct position in a term; it also allows us to write $(\nu c)\,(\ulcorner\!\! M \rrbracket\, c \mid \ulcorner\!\! c := N\cdot a \rrbracket\,)$ for $\ulcorner\!\! MN \rrbracket\, a$. This stresses that the π-calculus constitutes a very powerful abstract machine indeed: although the notion of structural reduction in $\lambda\mu$ is very different from normal

[2] A similar observation can be made for the encoding of $\lambda\mu$ in $\mathcal{X}$; see [9].

β-reduction, no special measures had to be taken in order to be able to express it; the component of our encoding that deals with pure λ-terms is almost exactly that of [10] (ignoring for the moment that substitution is modelled using a guard, which affects also the interpretation of variables), but for the use of replication in the case for application. In fact, the distributive character of structural substitution is dealt with entirely by congruence; see also Ex. 23. As standard in the literature [30], we say that a name a occurs in the *output subject position* of a process P if $P \Downarrow \overline{a}$.

Lemma 20. 1. Assume that a is only used for output R, Q. Then:
$(\nu a)\,(!a(x).P \mid Q \mid R) \;\sim_{\mathrm{C}}\; (\nu a)\,(!a(x).P \mid Q) \mid (\nu b)\,(!b(x).P \mid R[b/a])$
2. Assume that a is only used for input in R, Q. Then:
$(\nu a)\,(!\overline{a}\langle p\rangle.P \mid Q \mid R) \;\sim_{\mathrm{C}}\; (\nu a)\,(!\overline{a}\langle p\rangle.P \mid Q) \mid (\nu b)\,(!\overline{b}\langle p\rangle.P \mid R[b/a])$
3. $(\nu a)\,(\llbracket P\rrbracket\, a \mid !a(p).Q) \;\sim_{\mathrm{C}}\; (\nu a)\,((\nu b)\,(\llbracket P\rrbracket\, b \mid !b(p).Q) \mid !a(p).Q)$

To underline the significance of our results, notice that the translation is not trivial, since $\lambda y.y$ and $\lambda yz.y$ are interpreted by, respectively, the processes $(\nu yb)\,(y(u).!u\to b \mid \overline{a}\langle y,b\rangle)$ and $(\nu yb)\,((\nu zb)\,(y(u).!u\to b \mid \overline{b}\langle z,b\rangle) \mid \overline{a}\langle y,b\rangle)$, that differ under $\sim_{\mathrm{C}}$.

It is straightforward to show that typeability is preserved:

Theorem 21 (Type preservation). If $\Gamma \vdash_{\mu\mathbf{x}} M : A \mid \Delta$, then $\llbracket M\rrbracket\, a : \Gamma \vdash_{\pi} a{:}A, \Delta$.

PROOF. By induction on the structure of derivations in $\vdash_{\mu\mathbf{x}}$; we only show one case:

(C-*cut*) Then $M = P\langle\alpha := Q\cdot\gamma\rangle$ and we have both $\Gamma \vdash_{\mu\mathbf{x}} P : C \mid \alpha{:}A\to B, \Delta$ and $\Gamma \vdash_{\mu\mathbf{x}} Q : A \mid \gamma{:}B, \Delta$ for some B. By induction, there exist $\mathcal{D}_1 :: \llbracket P\rrbracket\, a : \Gamma \vdash_{\pi} a{:}C, \alpha{:}A\to B, \Delta$ and, since a is fresh, $\mathcal{D}_2 :: \llbracket Q\rrbracket\, w : \Gamma \vdash_{\pi} w{:}B, \Delta$, and we can construct

$$
\begin{prooftree}
\AxiomC{$\mathcal{D}_1$}
\noLine
\UnaryInfC{$\llbracket P\rrbracket\, a : \Gamma \vdash a{:}B\to A, \Delta$}
\AxiomC{$\mathcal{D}_2$}
\noLine
\UnaryInfC{$\llbracket Q\rrbracket\, w : \Gamma \vdash w{:}B, \Delta$}
\RightLabel{(out)}
\UnaryInfC{$\overline{b}\langle w\rangle.\llbracket Q\rrbracket\, w : \Gamma \vdash b{:}B, w{:}B, \Delta$}
\RightLabel{(ν)}
\UnaryInfC{$(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q\rrbracket\, w) : \Gamma \vdash b{:}B, \Delta$}
\RightLabel{$(!)$}
\UnaryInfC{$!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q\rrbracket\, w) : \Gamma \vdash b{:}B, \Delta$}
\AxiomC{}
\RightLabel{(0)}
\UnaryInfC{$0 : w{:}A \vdash w{:}A$}
\RightLabel{(out)}
\UnaryInfC{$\overline{\gamma}\langle w\rangle : w{:}A \vdash \gamma{:}A, w{:}A$}
\RightLabel{(in)}
\UnaryInfC{$d\to\gamma : d{:}A \vdash \gamma{:}A$}
\RightLabel{$(!)$}
\UnaryInfC{$!d\to\gamma : d{:}A \vdash \gamma{:}A$}
\RightLabel{$(\mid)$}
\BinaryInfC{$\llbracket b := Q\rrbracket \mid !d\to\gamma : \Gamma, d{:}A \vdash \gamma{:}A, b{:}B, \Delta$}
\RightLabel{$(pair\text{-}in)$}
\UnaryInfC{$\alpha(b,d).\,(\llbracket b := Q\rrbracket \mid !d\to\gamma) : \Gamma, \alpha{:}B\to A \vdash \gamma{:}A, \Delta$}
\RightLabel{$(!)$}
\UnaryInfC{$!\alpha(b,d).\,(\llbracket b := Q\rrbracket \mid !d\to\gamma) : \Gamma, \alpha{:}B\to A \vdash \gamma{:}A, \Delta$}
\RightLabel{$(\mid)$}
\BinaryInfC{$\llbracket P\rrbracket\, a \mid !\alpha(b,d).\,(\llbracket b := Q\rrbracket \mid !d\to\gamma) : \Gamma, \alpha{:}B\to A \vdash \gamma{:}A, \Delta$}
\RightLabel{(ν)}
\UnaryInfC{$(\nu\alpha)\,(\llbracket P\rrbracket\, a \mid \llbracket \alpha := Q\cdot\gamma\rrbracket) : \Gamma \vdash \gamma{:}A, \Delta$}
\end{prooftree}
$$

and $(\nu\alpha)\,(\llbracket P\rrbracket\, a \mid \llbracket\alpha := Q\cdot\gamma\rrbracket) = \llbracket P\langle\alpha := Q\cdot\gamma\rangle\rrbracket\, a$. □

We will now show that our translation fully respects the explicit reduction $\to_{\mathbf{x}}$, modulo contextual equivalence, using renaming of output and garbage collection. Renaming is defined and justified via the following lemma.

Lemma 22 (Renaming lemma). 1. $(\nu a)\,(!a\to e \mid \llbracket M\rrbracket\, a) \;\sim_{\mathrm{C}}\; \llbracket M[e/a]\rrbracket\, e.$
2. $(\nu a)\,(!a\to e \mid \llbracket M\rrbracket\, b) \;\sim_{\mathrm{C}}\; \llbracket M[e/a]\rrbracket\, b.$

We will use $\sim_{\mathrm{R}}$ if we want to emphasise that two processes are equivalent just using renaming and write $\rightsquigarrow_{\pi}^{*}$ for the relation $\to_{\pi}^{*} \cup \sim_{\mathrm{G}} \cup \sim_{\mathrm{R}}$.

$$\begin{array}{ll}
\llbracket (\lambda x.x)(\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)) \rrbracket\, a & \triangleq \\
(\nu c)\,((\nu xb)\,(\llbracket x \rrbracket\, b \mid \overline{c}\langle x,b\rangle) \mid !c(v,d).\,(\llbracket v := \mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y) \rrbracket \mid !d\rightarrow a)) & \rightarrow (c) \\
(\nu xb)\,(\llbracket x \rrbracket\, b \mid \llbracket x := \mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y) \rrbracket \mid !b\rightarrow a) \mid (\nu c)\,(!c(v,d).\cdots) & \equiv, \triangleq, \sim_{\mathrm{G}} \\
(\nu xb)\,(x(u).\,!u\rightarrow b \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket \mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y) \rrbracket\, w) \mid !b\rightarrow a) & \rightarrow (x) \\
(\nu wb)\,(!w\rightarrow b \mid \llbracket \mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y) \rrbracket\, w \mid !b\rightarrow a) \mid (\nu x)\,(!\,(\nu w)\,(\overline{x}\langle w\rangle.\cdots)) & \triangleq, \sim_{\mathrm{G}}, =_{\alpha} \\
(\nu\alpha b)\,(!\alpha\rightarrow b \mid (\nu\bullet)\,(\llbracket [\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y) \rrbracket\, \bullet) \mid !b\rightarrow a) & \triangleq, \equiv \\
(\nu\alpha b)\,(!\alpha\rightarrow b \mid (\nu c)\,((\nu q b_1)\,(\llbracket q \rrbracket\, b_1 \mid \overline{c}\langle q,b_1\rangle) \mid & \\
\qquad\qquad !c(v,d).\,(\llbracket v := \mu\beta.[\alpha]\lambda y.y \rrbracket \mid !d\rightarrow \alpha)) \mid !b\rightarrow a) & \rightarrow (c), \sim_{\mathrm{G}}, \triangleq \\
(\nu\alpha b)\,(!\alpha\rightarrow b \mid (\nu q b_1)\,(q(u).\,!u\rightarrow b_1 \mid & \\
\qquad\qquad !\,(\nu w)\,(\overline{q}\langle w\rangle.\,\llbracket \mu\beta.[\alpha]\lambda y.y \rrbracket\, w) \mid !b_1\rightarrow \alpha) \mid !b\rightarrow a) & \rightarrow (q), \sim_{\mathrm{G}}, \triangleq, \equiv \\
(\nu\alpha b)\,(!\alpha\rightarrow b \mid \llbracket \lambda y.y \rrbracket\, \alpha \mid !b\rightarrow a) \qquad \triangleq, \sim_{\mathrm{R}}, \sim_{\mathrm{G}} & (\nu yb)\,(\llbracket y \rrbracket\, b \mid \overline{a}\langle y,b\rangle)
\end{array}$$

Fig. 2. The translation of a term with double output

Using this lemma, we can show the following:

Example 23. The translation of a β-redex reduces as:

$$\begin{array}{ll}
\llbracket (\lambda x.P)Q \rrbracket\, a & \triangleq \\
(\nu c)\,((\nu xb)\,(\llbracket P \rrbracket\, b \mid \overline{c}\langle x,b\rangle) \mid !c(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) & \rightarrow_{\pi} (c) \\
(\nu bx)\,(\llbracket P \rrbracket\, b \mid !b\rightarrow a \mid \llbracket x := Q \rrbracket) \mid (\nu c)\,(!c(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) & \sim_{\mathrm{G}} \\
(\nu bx)\,(\llbracket P \rrbracket\, b \mid !b\rightarrow a \mid \llbracket x := Q \rrbracket) & \sim_{\mathrm{R}} (22) \\
(\nu x)\,(\llbracket P \rrbracket\, a \mid \llbracket x := Q \rrbracket) \qquad\qquad \triangleq & \llbracket P\langle x := Q\rangle \rrbracket\, a
\end{array}$$

This implies that β-reduction is implemented in π by at least one π-reduction.

On the other hand, μ-reduction consists of a reorganisation of the structure of a term by changing its applicative structure. Since application is essentially modelled through parallel composition, this implies that the translation of a μ-redex is essentially dealt with by congruence and renaming. For example,

$$\begin{array}{ll}
\llbracket (\mu\beta.[\beta]P)Q \rrbracket\, a & \triangleq \\
(\nu c)\,((\nu\bullet)\,((\llbracket P \rrbracket\, \beta)[c/\beta]) \mid !c(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) & \sim_{\mathrm{C}} \ (=_{\alpha}) \\
(\nu\beta)\,(\llbracket P \rrbracket\, \beta \mid !\beta(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) &
\end{array}$$

We can show, using Lem. 20, this last process is contextually equivalent to

$$\begin{array}{l}
(\nu\gamma)\,((\nu\beta)\,(\llbracket P \rrbracket\, \gamma \mid !\beta(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) \mid !\gamma(v,d).\,(\llbracket v := Q \rrbracket \mid !d\rightarrow a)) \\
\quad \triangleq \ \llbracket P\langle \beta := Q{\cdot}a\rangle Q \rrbracket\, a
\end{array}$$

(notice that we have separated out the outside name of the term P, being β, which we renamed to γ; this leaves two context substitutions, one dealing with the occurrences of β inside P, and one with γ[3]).

Translations of terms in $\rightarrow_{\mathrm{xH}}$-normal form are in normal form as well.

Lemma 24. $\boldsymbol{N}$ is a $\rightarrow_{\mathrm{xH}}$-nf implies $\llbracket \boldsymbol{N} \rrbracket\, a$ is irreducible.

To illustrate the expressiveness of our translation, we give some examples:

[3] This corresponds to the behaviour of rule $(\backslash\!\!\!\nwarrow imp\text{-}outs)$ in $\mathcal{X}$.

Example 25. 1. In Fig. 2 we run $\ulcorner(\lambda x.x)(\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y))\lrcorner\, a$, as an example of a term that generates two outputs over α, and highlights the need for the repeated use of replication.

2. $$\begin{array}{lll}\ulcorner PQR\lrcorner\, a & \triangleq,\equiv & (\nu cc')\,(\ulcorner P\lrcorner\, c' \mid {!c'(v,d)}.\,(\ulcorner v := Q\lrcorner \mid {!d{\to}c}) \mid \\ & & {!c(v,d)}.\,(\ulcorner v := R\lrcorner \mid {!d{\to}a}))\end{array}$$

so components of applications are placed in parallel under the translation. Similarly,

$$\ulcorner M\langle\alpha := N{\cdot}\beta\rangle\langle\gamma := L{\cdot}\delta\rangle\lrcorner\, a \;=\; (\nu\gamma\alpha)\,(\ulcorner M\lrcorner\, a \mid \ulcorner\alpha := N{\cdot}\beta\lrcorner \mid \ulcorner\gamma := L{\cdot}\delta\lrcorner)$$

so repeated structural substitutions are also placed in parallel under the translation and can be applied independently.

7 Soundness, Completeness, and Termination

As in [24,31], we can now show a reduction preservation result for explicit head reduction for $\Lambda\mu\mathbf{x}$, by showing that $\ulcorner\cdot\lrcorner\,\cdot$ preserves $\rightarrow_{\mathrm{xH}}$ up to $\rightsquigarrow^*_\pi$. Since reduction in interpreted terms takes place over hidden channels exclusively, by Lem. 8, $\rightsquigarrow^*_\pi \subseteq \sim_{\mathrm{C}}$, so we could have shown the following result using $\sim_{\mathrm{C}}$ as well, but the current formulation is more expressive; notice that we do not require the terms to be closed.

Theorem 26 (Soundness). *$M \rightarrow_{\mathrm{xH}} N \Rightarrow \ulcorner M\lrcorner\, a \rightsquigarrow^*_\pi \ulcorner N\lrcorner\, a$.*

PROOF. We show only the interesting cases.

$$\begin{array}{lll}
x\langle x := N\rangle \rightarrow N : \quad \ulcorner x\langle x := N\rangle\lrcorner\, a & \triangleq & \\
\quad (\nu x)\,(\ulcorner x\lrcorner\, a \mid \ulcorner x := N\lrcorner) & \equiv & \\
\quad (\nu x)\,(x(u).{!u{\to}a} \mid (\nu w)\,(\overline{x}\langle w\rangle.\ulcorner N\lrcorner\, w) \mid \ulcorner x := N\lrcorner) & \rightarrow_\pi & (x) \\
\quad (\nu w)\,({!w{\to}a} \mid \ulcorner N\lrcorner\, w) \mid (\nu x)\,(\ulcorner x := N\lrcorner) & \sim_{\mathrm{R}},\sim_{\mathrm{G}} & \ulcorner N\lrcorner\, a
\end{array}$$

$$\begin{array}{ll}
(PQ)\langle x := N\rangle \rightarrow (P\langle x := N\rangle Q)\langle x := N\rangle,\ x = hv(P) : \ \ulcorner (PQ)\langle x := N\rangle\lrcorner\, a & \triangleq \\
\quad (\nu x)\,((\nu c)\,(\ulcorner P\lrcorner\, c \mid {!\ulcorner c := Q{\cdot}a\lrcorner}) \mid \ulcorner x := N\lrcorner) & \sim_{\mathrm{C}} (20) \\
\quad (\nu x)\,((\nu c)\,((\nu x)\,(\ulcorner P\lrcorner\, c \mid \ulcorner x := N\lrcorner) \mid {!\ulcorner c := Q{\cdot}a\lrcorner}) \mid \ulcorner x := N\lrcorner) & \triangleq \\
\quad (\nu x)\,((\nu c)\,(\ulcorner P\langle x := N\rangle\lrcorner\, c \mid {!\ulcorner c := Q{\cdot}a\lrcorner}) \mid \ulcorner x := N\lrcorner) & \triangleq,\equiv \\
\quad (\nu x)\,(\ulcorner P\langle x := N\rangle Q\lrcorner\, a \mid \ulcorner x := N\lrcorner) & \triangleq \\
\quad \ulcorner (P\langle x := N\rangle Q)\langle x := N\rangle\lrcorner\, a &
\end{array}$$

$$\begin{array}{ll}
(\mu\beta.M)N \rightarrow \mu\gamma.M\langle\beta := N{\cdot}\gamma\rangle,\ \gamma\ \mathit{fresh} : \quad \ulcorner(\mu\beta.M)N\lrcorner\, a & \triangleq \\
\quad (\nu c)\,((\nu\bullet)\,((\ulcorner M\lrcorner\,\bullet)[c/\beta]) \mid {!c(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}a})) & =_\alpha \\
\quad (\nu\beta)\,((\nu\bullet)\,(\ulcorner M\lrcorner\,\bullet) \mid {!\beta(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}a})) & \equiv,= \\
\quad (\nu\bullet)\,(((\nu\beta)\,(\ulcorner M\lrcorner\,\bullet \mid {!\beta(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}\gamma})))[a/\gamma]) & \triangleq \\
\quad \ulcorner\mu\gamma.M\langle\beta := N{\cdot}\gamma\rangle\lrcorner\, a &
\end{array}$$

$$\begin{array}{ll}
([\alpha]M)\langle\alpha := N{\cdot}\gamma\rangle \rightarrow [\gamma](M\langle\alpha := N{\cdot}\gamma\rangle)N : \quad \ulcorner([\alpha]M)\langle\alpha := N{\cdot}\gamma\rangle\lrcorner & \triangleq \\
\quad (\nu\alpha)\,(\ulcorner M\lrcorner\,\alpha \mid {!\alpha(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}\gamma})) & \sim_{\mathrm{C}}\ (20) \\
\quad (\nu c)\,((\nu\alpha)\,(\ulcorner M\lrcorner\, c \mid {!\alpha(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}\gamma})) \mid & \\
\qquad\qquad {!c(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}\gamma})) & \triangleq \\
\quad (\nu c)\,(\ulcorner M\langle\alpha := N{\cdot}\gamma\rangle\lrcorner\, c \mid {!c(v,d)}.\,(\ulcorner v := N\lrcorner \mid {!d{\to}\gamma})) & \triangleq \\
\quad \ulcorner[\gamma]M\langle\alpha := N{\cdot}\gamma\rangle N\lrcorner &
\end{array}$$

$\square$

The main soundness result is formulated as:

Theorem 27 (Operational Soundness for $\rightarrow_{\mathrm{xH}}$). 1$M \rightarrow^*_{\mathrm{xH}} N \Rightarrow \llbracket M \rrbracket\, a \rightsquigarrow^*_{\pi} \llbracket N \rrbracket\, a$.
2. $M \uparrow_{\mathrm{xH}} \Rightarrow \llbracket M \rrbracket\, a \uparrow \pi$.

Since $\rightsquigarrow^*_{\pi} \subseteq \sim_{\mathrm{C}}$, which is symmetric, Thm. 27 gives that $\llbracket \cdot \rrbracket \cdot$ preserves $=_{\mathrm{xH}}$ up to $\sim_{\mathrm{C}}$.

Corollary 28 (Adequacy). $M =_{\mathrm{xH}} N \Rightarrow \llbracket M \rrbracket\, a \sim_{\mathrm{C}} \llbracket N \rrbracket\, a$.

This result states that our encoding gives, in fact, a semantics for the explicit head reduction for $\Lambda\mu$. As for a full abstraction result, note that we cannot show the reverse of Cor. 28, since different unsolvable terms like $(\lambda x.xx)(\lambda x.xx)$ and $(\lambda w.www)(\lambda w.www)$ are not equivalent under $=_{\mathrm{xH}}$, but are *contextually* equivalent under $\llbracket \cdot \rrbracket \cdot$, *i.e.* have the same observable behaviour, as is illustrated by the fact that their translations never exhibit an output.

We can also show operational completeness for $\rightarrow_{\mathrm{xH}}$.

Theorem 29 (Operational completeness for $\rightarrow_{\mathrm{xH}}$). If $\llbracket M \rrbracket\, a \rightarrow_{\pi} P$ then there exists N such that $P \rightsquigarrow^*_{\pi} \llbracket N \rrbracket\, a$, and $M \rightarrow_{\mathrm{xH}} N$.

This in turn can be used to show:

Lemma 30. 1. Let M be a term in $\Lambda\mu\mathbf{x}$. If $\llbracket M \rrbracket\, a \rightarrow^*_{\pi} \llbracket N \rrbracket\, a$ then $M \rightarrow^*_{\mathrm{xH}} N$.
2. Let $M \in \Lambda\mu$, *i.e.* a (pure) $\Lambda\mu$-term. If $\llbracket M \rrbracket\, a \rightarrow_{\pi} P$ then there exists $N \in \Lambda\mu\mathbf{x}$ and $L \in \Lambda\mu$ such that $P \sim_{\mathrm{C}} \llbracket N \rrbracket\, a$, and $M \rightarrow^*_{\mathrm{xH}} N$ and $N \rightarrow^*_{:=} L$.

We can show the following termination results:

Theorem 31 (Termination). 1. If $M \rightarrow^*_{\mathrm{xH}} N$, with N in explicit head-normal from, then $\llbracket M \rrbracket\, a \downarrow \pi$.
2. If $M \rightarrow^*_{\beta\mu} N$, with N in head-normal from, then $\llbracket M \rrbracket\, a \downarrow \pi$.
3. Let $M \in \Lambda\mu$. If $\llbracket M \rrbracket\, a \downarrow \pi$ then there exists $N \in \Lambda\mu\mathbf{x}$ and L in $\rightarrow_{\Lambda\mu}$-head normal form such that $\llbracket M \rrbracket\, a \sim_{\mathrm{C}} \llbracket N \rrbracket\, a$, and $M \rightarrow^*_{\mathrm{xH}} N$ and $N \rightarrow^*_{:=} L$.

Notice that, in the first case, the normal form of $\llbracket M \rrbracket\, a$ need not be $\llbracket N \rrbracket\, a$; a similar observation can be made with respect to Milner's encoding. Notice also that this result is stronger than the formulation of the termination result for Milner's encoding in [31], since it models reduction to head-normal form, not just normal form. However, since terms that have a normal form have a head-normal form as well, Thm. 31 immediately leads to:

Corollary 32. If $M \downarrow_{\beta\mu}$, then $\llbracket M \rrbracket\, a \downarrow \pi$.

8 Conclusions

We have defined an output based, logic inspired translation of untyped $\Lambda\mu$ with explicit substitution into the π-calculus and shown that it respects step-by-step head-reduction, assignable types, head-conversion, and termination. We conjecture that we can show the results shown above also for head reduction with *implicit* substitution; for this we would need to show that, if $M \rightarrow_{:=}^{*} N$, then $\ulcorner M \lrcorner\, a \sim_{\mathrm{C}} \ulcorner N \lrcorner\, a$. It seems that the approach via Levy-Longo trees is more suitable for that.

There are many alternatives to the approach we have chosen to follow here; especially our choice for contextual equivalence (inspired by λ-calculus semantics) could be replaced by branching semantics, or a bisimulation-like equivalence. The natural question is then, which of our properties would be affected? Would branching and non-branching equivalences to coincide, maybe by exploiting some confluence properties?

We leave these issues for future work.

References

1. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.-J.: Explicit Substitutions. JFP 1(4), 375–416 (1991)
2. Abadi, M., Gordon, A.: A Calculus for Cryptographic Protocols: The Spi Calculus. In: CCS 1997, pp. 36–47 (1997)
3. Abramsky, S.: The lazy lambda calculus. Research topics in functional programming, pp. 65–116. Addison-Wesley (1990)
4. Audebaud, P.: Explicit Substitutions for the $\Lambda\mu$ Calculus. RR 94-26, ÉNS de Lyon (1994)
5. van Bakel, S.: Completeness and Partial Soundness Results for Intersection & Union Typing for $\bar{\lambda}\mu\tilde{\mu}$. Annals of Pure and Applied Logic 161, 1400–1430 (2010)
6. van Bakel, S.: Completeness and Soundness results for $\mathcal{X}$ with Intersection and Union Types. Fundamenta Informaticae (to appear, 2012)
7. van Bakel, S., Barbanera, F., de'Liguoro, U.: A Filter Model for the $\lambda\mu$-Calculus. In: Ong, L. (ed.) TLCA 2011. LNCS, vol. 6690, pp. 213–228. Springer, Heidelberg (2011)
8. van Bakel, S., Cardelli, L., Vigliotti, M.G.: From $\mathcal{X}$ to π; Representing the Classical Sequent Calculus in the π-calculus. In: CL&C 2008 (2008)
9. van Bakel, S., Lescanne, P.: Computation with Classical Sequents. MSCS 18, 555–609 (2008)
10. van Bakel, S., Vigliotti, M.G.: A Logical Interpretation of the λ-Calculus into the π-Calculus, Preserving Spine Reduction and Types. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 84–98. Springer, Heidelberg (2009)
11. Barendregt, H.: The Lambda Calculus: its Syntax and Semantics. North-Holland (1984)
12. Bloo, R., Rose, K.H.: Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection. In: CSN 1995, pp. 62–72 (1995)
13. de Bruijn, N.G.: Lambda Calculus Notation with Nameless Dummies: A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. Ind. Math. 34, 381–392 (1972)
14. Church, A.: A Note on the Entscheidungsproblem. JSL 1(1), 40–41 (1936)
15. Cimini, M., Sacerdoti Coen, C., Sangiorgi, D.: $\bar{\lambda}\mu\tilde{\mu}$ calculus, π-calculus, and abstract machines. In: EXPRESS 2009 (2009)
16. Crolard, T.: A confluent lambda-calculus with a catch/throw mechanism. JFP 9(6), 625–647 (1999)
17. Curien, P.-L., Herbelin, H.: The Duality of Computation. In: ICFP 2000, pp. 233–243. ACM (2000)
18. Gentzen, G.: Investigations into logical deduction. In: Szabo, M.E. (ed.) The Collected Papers of Gerhard Gentzen, p. 68. North Holland (1935, 1969)

19. de Groote, P.: On the Relation Between the $\lambda\mu$-Calculus and the Syntactic Theory of Sequential Control. In: Pfenning, F. (ed.) LPAR 1994. LNCS, vol. 822, pp. 31–43. Springer, Heidelberg (1994)
20. Herbelin, H.: C'est maintenant qu'on calcule: au cœur de la dualité. Mémoire d'habilitation, Université Paris 11 (2005)
21. Honda, K., Tokoro, M.: An Object Calculus for Asynchronous Communication. In: America, P. (ed.) ECOOP 1991. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)
22. Honda, K., Yoshida, N., Berger, M.: Control in the π-Calculus. In: CW 2004 (2004)
23. Lassen, S.B.: Head Normal Form Bisimulation for Pairs and the $\lambda\mu$-Calculus. In: LICS 2006, pp. 297–306 (2006)
24. Milner, R.: Functions as processes. MSCS 2(2), 269–310 (1992)
25. Parigot, M.: An Algorithmic Interpretation of Classical Natural Deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 190–201. Springer, Heidelberg (1992)
26. Parigot, M.: Strong Normalization for Second Order Classical Natural Deduction. In: LICS 1993, pp. 39–46 (1993)
27. Pierce, B.C., Sangiorgi, D.: Typing and Subtyping for Mobile Processes. MSCS 6(5), 409–453 (1996)
28. Py, W.: Confluence en $\lambda\mu$-calcul. Phd thesis, Univ. Savoie (1998)
29. Sangiorgi, D.: Expressing Mobility in Process Algebra: First Order and Higher Order Paradigms. PhD thesis, Univ. Edinburgh (1992)
30. Sangiorgi, D.: Lazy functions and mobile processes. RR 2515, INRIA, Sophia-Antipolis, France (1995)
31. Sangiorgi, D., Walker, D.: The Pi-Calculus. Cambridge University Press (2001)
32. Sestoft, P.: Demonstrating Lambda Calculus Reduction. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 420–435. Springer, Heidelberg (2002)
33. Wadsworth, C.P.: The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. SIAM JoC 5, 488–521 (1976)

Erratum: Probabilistic Inference and Monadic Second Order Logic

Marijke Hans L. Bodlaender

P.O. Box 80.089, Department of Computing Sciences, Utrecht University, The Netherlands
h.l.bodlaender@uu.nl

J.C.M. Baeten, T. Ball, and F.S. de Boer (Eds.): TCS 2012, LNCS 7604, pp. 43–56, 2012.

DOI 10.1007/978-3-642-33475-7_27

The name of the author of the paper starting on page 43 of this volume has been printed incorrectly. It should read: Hans L. Bodlaender

The original online version for this chapter can be found at
http://dx.doi.org/10.1007/978-3-642-33475-7_4

Erratum: Cinderella versus the Wicked Stepmother

Marijke Hans L. Bodlaender[1], Cor A.J. Hurkens[2], Vincent J.J. Kusters[2], Frank Staals[2], Gerhard J. Woeginger[2], and Hans Zantema[2]

[1] Dept. of Information and Computing Sciences, Universiteit Utrecht, Netherlands
[2] Dept. of Mathematics and Computer Science, TU Eindhoven, Netherlands

J.C.M. Baeten, T. Ball, and F.S. de Boer (Eds.): TCS 2012, LNCS 7604, pp. 57–71, 2012.

DOI 10.1007/978-3-642-33475-7_28

The name of the first author of the paper starting on page 57 of this volume has been printed incorrectly. It should read: Marijke H.L. Bodlaender

The original online version for this chapter can be found at
http://dx.doi.org/10.1007/978-3-642-33475-7_5

Author Index