Syntactic approaches to negative results in process algebras and modal logics

by

Elli Anastasiadi

Dissertation submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

October 17, 2022

Thesis Committee:

Supervisors:	Luca Aceto, Chair, Professor, Reykjavík University, Iceland Anna Ingólfsdóttir, Professor, Reykjavík University, Iceland
Examining Board:	Antonis Achilleos, Assistant Professor, Reykjavík University, Iceland
External Examiners:	Ilaria Castellani, INRIA Senior Researcher, Sophia Antipolis, France Catuscia Palamidessi, INRIA Saclay. Director of Research & École polytechnique Lix, France

ISBN 978-9935-9694-0-8 Print version
ISBN 978-9935-9694-1-5 Electronic version
Authors ORCID: 0000-0001-7526-9256

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

Date of Signature

Luca Aceto
Professor, Chair, School of Computer Science,
Reykjavik, Iceland

Anna Ingólfsdóttir
Professor, School of Computer Science,
Reykjavik, Iceland

Antonis Achilleos
Assistant Professor, School of Computer Science,
Reykjavik, Iceland

I. Castells

Ilaria Castelani
INDES, INRIA Senior Researcher,
Sophia Antipolis, France

Catuscia Palamidessi,
INRIA Saclay Director of Research &
Lix École polytechnique, France

REYKJAVIK UNIVERSITY

Syntactic approaches to negative results in process algebras and modal logics

Elli Anastasiadi

October 17, 2022

Abstract

Concurrency as a phenomenon is observed in most of the current computer science trends. However the inherent complexity of analyzing the behavior of such a system is incremented due to the many different models of concurrency, the variety of applications and architectures, as well as the wide spectrum of specification languages and demanded correctness criteria. For the scope of this thesis we focus on state based models of concurrent computation, and on modal logics as specification languages. First we study syntactically the process algebras that describe several different concurrent behaviors, by analyzing their equational theories. Here, we use well-established techniques from the equational logic of processes to older and newer setups, and then transition to the use of more general and novel methods for the syntactical analysis of models of concurrent programs and specification languages. Our main contributions are several positive and negative axiomatizability results over various process algebraic languages and equivalences, along with some complexity results over the satisfiability of multi-agent modal logic with recursion, as a specification language.

Setningarfræðilegar aðferðir við neikvæðar niðurstöður í algebrum vinnslu og mótalógík

Elli Anastasiadi

October 17, 2022

Útdráttur

Samhliða sem fyrirbæri sést í flestum núverandi tölvunarfræði stefnur. Hins vegar er eðlislægt flókið að greina hegðun slíks kerfis- tem er aukið vegna margra mismunandi gerða samhliða, fjölbreytileikans af forritum og arkitektúr, svo og breitt svið forskrifta mælikvarða og kröfðust réttmætisviðmiða. Fyrir umfang þessarar ritgerðar leggjum við áherslu á ástandsbundin líkön af samhliða útreikningum og á formlegum rökfræði sem forskrift tungumálum. Fyrst skoðum við setningafræðilega ferlialgebrurnar sem lýsa nokkrum mismunandi samhliða hegðun, með því að greina jöfnukenningar þeirra. Hér notum við rótgróin tækni mynda jöfnunarrökfræði ferla til eldri og nýrri uppsetningar, og síðan umskipti yfir í notkun almennari og nýrra aðferða fyrir setningafræðileg greining á líkönum samhliða forrita og forskriftartungumála. Helstu framlög okkar eru nokkrar jákvæðar og neikvæðar niðurstöður um axiomatizability yfir ýmis ferli algebrumál og jafngildi, ásamt nokkrum sam-Sveigjanleiki leiðir af því að fullnægjanleiki fjölþátta formrökfræði með endurkomu, sem a forskrift tungumál.

Acknowledgments

This thesis and this PhD cover the content of one of the most significant pieces of my life for the last three and a half years. Over those, I came to be thankful to a vast set of people who helped practically, scientifically, and through emotional support. It is impossible to list everyone in this set, but I will try to at least express my gratitude as clearly as I can to as many as I can because they all deserve it.

The first people I would like to mention are my supervisors, PhD committee, and collaborators. First, I am thankful to my primary supervisor and mentor, Luca Aceto. He played a leading role by guiding me, providing feedback, exchanging ideas, and being an impeccable example of scientific expertise and mentoring ability that I hope to achieve myself one day. He was always there to help despite his workload, and he was smart enough to understand what ideas I had (which was probably not easy) and find a way to lead me to make something out of them.

Anna Ingolfsdottir was my second supervisor and significantly affected me and my teaching style and experience. She had completely different guiding methods from what I knew before (both towards me and while teaching together), and through our interaction I expanded my mentality to account for the advantages of her methods. I am happy to continue as an academic after having evolved through our interaction.

Antonis Achilleos is not only a co-worker but also a friend. He has spent many (maybe too many) hours of his life working with me. As researchers or instructors, our collaboration has always been safe, judgment-free (even for not-so-smart ideas), creative, and most of all, friendly and relaxed. When I grow up (more), I would like to be lucky enough to have such relationships with all my colleagues.

My two PhD examiners, Ilaria Castellani and Catuscia Palamidessi, showed attention to my work, and their detailed comments helped me make it better. They were very supportive of my efforts and ideas. It was very inspiring to have met them and have them discuss my work with me.

Finally, through my PhD I met and interacted with several other researchers who had a guiding and inspirational role for me. I want to mention Adrian Francalanza, & Karoliina Lehtinen, who constitute a large part of the researchers I consider close to me. Our interactions have always taught me things about science or life in academia itself. They were always happy to support me.

The people I mentioned above, along with many others I hold dearly in my mind, are the kind of people I hope to have around me for the rest of my academic life, and they have made me look forward to it. I thank all of them warmly.

However, aside from the support I received in my work environment, there are also the people who made this journey happy through our informal relationships.

My parents, Giorgos and Emmanouela, have done their best to be good parents, and even though they do not think so, they are doing it great. They are some of the most selfless people I know, and I am lucky I had their unconditional support and their home to go back to whenever I needed. My siblings, Petros and Eva, have always been people I can talk to without any social boundaries and stress, something quite handy for completing a PhD. They are also great researchers, and I am proud to be their sister.

For my friends, it is tough to mention them all, and the more I write, the more I am sure I forget many. Yiota, Giannis, Vangelis, & Duncan is the absolute shortest list I can think of. To my friends in Crete, my mentors and friends from the Rethymno Philharmonic Orchestra, my friends in Athens, my friends from my time in NTUA, and my friends from Iceland (both permanent and passing through it): you have made me think of new things, see new places, become less weird, explore, and you have given me some of my best memories. Without all of these things, I am sure I would not be confident enough to attempt the few things I am proud of or healthy enough to have lived to tell the tale.

Great thanks also go to prof. Stathis Zachos, prof. Aris Pagourtzis and Dr. Angeliki Chalki, along with all the members of CoReLab, and the Logic Study Group that I had the luck to meet. They were my first introduction to Theoretical Computer Science, they were there while I was exploring it, and their passion and character are what pushed me to make it my life's work.

Finally, I would like to thank Joshua Springer, who has been selflessly surviving this PhD for the last few years with me and knows more than anyone how easy or hard, happy or sad, it has been. He has been supportive, patient, and kind-hearted and has provided both practical and emotional assistance whenever needed. I believe I am a better person due to our coexistence, and I am proud of it being so. Thanks to all of the other people mentioned here, I would have maybe made it somehow to finish the PhD even without his support, but he being there for it is what made me do it while being this happy.

Thank you all. This PhD is dedicated to all of you.

List of Tables

2.1 The inference rules of equational logic, for an n-ary $f \in \mathcal{F}$. 19

2.2 SOS rules for BPA with the termination predicate. 25

2.3 SOS rules for CCS ($\alpha \in \mathit{Act}$, $a \in L$). 26

2.4 Monitors, dynamics, and instrumentation. 31

3.1 Operational semantics of processes in BPA_Θ. 44

3.2 Rules of equational logic over BPA_Θ. 47

3.3 Some axioms of BPA_Θ. 48

3.4 Inference rules for the auxiliary transition relations. The symmetric versions of rules a_6–a_8 have been omitted. 54

3.5 Operational semantics and some axioms of the unless operator. 78

4.1 Operational semantics of processes in Mon_F. 90

4.2 Equational Laws, over Mon_F. 93

4.3 The axioms of $\mathcal{E}_v$ 94

4.4 Our axiom systems. 137

6.1 The SOS rules for BCCSP operators ($a \in \textsc{Act}$). 158

6.2 Finite equational basis for BCCSP modulo bisimilarity. 159

6.3 The SOS rules for CCS_a interleaving parallel composition. 160

6.4 SOS rules for the parallel operator $|_A$, $A \subseteq \textsc{Act}$. 164

6.5 Additional axioms for $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$. 176

6.6 The SOS rules for $\;|\;$ operator ($\alpha \in \textsc{Act}_\tau$, $\mu \in \textsc{Act} \cup \overline{\textsc{Act}}$). 179

6.7 The SOS rules for the $\backslash$ operator ($\alpha \in \textsc{Act}_\tau$, $\mu \in \textsc{Act} \cup \overline{\textsc{Act}}$). 179

6.8 The SOS rules for the $[f]$ operator (with f a symmetric relation in $\textsc{Act} \times \textsc{Act} \cup \{\tau, \tau\}$). 182

7.1 Semantics of modal formulas on a model $\mathsf{M} = (W, R, V)$. We omit M from the notation. 191

7.2 The tableau rules for $\mathbf{L} = \mathbf{L}^{\mu}_{n}$. 208

8.1 Basic axioms for CCS. $\mathsf{E}_0 = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{A3}\}$ and $\mathsf{E}_1 = \mathsf{E}_0 \cup \{\mathrm{P0}, \mathrm{P1}\}$.223

8.2 The expansion law. 224

List of Figures

2.1 Two trace equivalent, but not bisimilar processes. 21
2.2 The framework of runtime verification 28
2.3 The semantics of modal logic 33

4.1 Transitions the monitor $\sigma_{bad}(m)$ can perform 128

7.1 The frame property hierarchy 195
7.2 An example where the use of the alternative axiom for symetricity would wield a non-valid translation. 199
7.3 The new model, based on a world s in $\mathcal{M}$, with two neighbors t_1, t_2. 203
7.4 Tableaux for φ_1 and φ_2. The dots represent that the tableau keeps repeating as from the identical node above. The `x` mark represents a propositionally closed branch. 213

Contents

Abstract . . . iii
Acknowledgments . . . vii
List of Tables . . . x
List of Figures . . . xiii

I On the Shoulders of Giants 1

1 Introduction 3
1.1 Our Results . . . 8
1.2 Negative Results in Formal Methods . . . 10
1.3 Historical Notes . . . 11

2 Preliminaries 17
2.1 Equational Logic . . . 18
2.2 Labeled Transition Systems and Bisimulation . . . 20
2.3 Process Algebra . . . 23
2.4 Runtime Verification . . . 27
2.4.1 Regular Monitors . . . 30
2.5 Modal Logic . . . 31
2.5.1 Syntax . . . 32
2.5.2 Semantics . . . 32
2.5.3 Connections to Process Algebra . . . 34

II Contributions 37

3 Axiomatizability of BPA$_\Theta$ 39
3.1 Introduction . . . 39
3.1.1 On the Axiomatizability of Priority . . . 40
3.1.2 Our Contribution . . . 41
3.1.3 Outline of the Chapter . . . 42
3.1.4 What's New . . . 42
3.2 Background . . . 43
3.2.1 BPA$_\Theta$: Syntax and Semantics . . . 43
3.2.2 Order-Insensitive Bisimulation . . . 46

3.2.3 Equational Logic, in the Context of BPA$_\Theta$ 47
3.3 Towards a Negative Result 49
3.3.1 The Idea 49
3.3.2 The Choice of $\mathbb{P}(n)$ 49
3.3.3 The Choice of n 50
3.3.4 The Family of Equations 51
3.4 Semantics for Open Terms 52
3.4.1 From Open to Closed Transitions... 52
3.4.2 ...and Back Again 58
3.5 Uniform Determinacy 69
3.6 The Property: Uniform (n, Θ)-Dependency 71
3.7 The Negative Result Over BPA$_\Theta$ 73
3.8 On the Use of Auxiliary Operators 78
3.9 A complexity question 80
3.10 Conclusions 83

4 Axiomatizing monitors 85
4.1 Introduction 85
4.2 Preliminaries 88
4.3 Ground-Complete Axiomatizations 93
4.3.1 Axiomatizing ω-Verdict Equivalence 100
4.4 Open Terms 103
4.4.1 Infinite Set of Actions 107
4.4.2 Finite Set of Actions 111
4.5 A Non-Finite-Axiomatizability Result 136
4.6 Conclusions 142

5 CCS Modulo Weak Equivalences 145
5.1 Background 145
5.2 The negative Result in the Weak Setting 147
5.3 Conclusion 151

6 Axiomatizability results via reductions 153
6.1 Introduction 153
6.2 Preliminaries 157
6.3 The Proof Strategy: Reduction Mappings 160
6.4 Results for CSP Parallel Composition 162
6.4.1 The Languages BCCSP$^{\mathrm{p}}_A$, BCCSP$^{\mathrm{p}}_{\textsc{Act}}$, BCCSP$^{\mathrm{p}}$ and BCCSP$^{\mathrm{p}}_\tau$ 163
6.4.2 The Negative Result for BCCSP$^{\mathrm{p}}_A$ 165
6.4.3 The Case of BCCSP$^{\mathrm{p}}$ and the Negative Result for BCCSP$^{\mathrm{p}}_\tau$. 169
6.4.4 The Case of BCCSP$^{\mathrm{p}}_{\textsc{Act}}$ 175
6.5 Axiomatizability Results for CCS Full Merge 177
6.5.1 The Case of Restriction 179
6.5.2 The Case of Relabeling 182
6.6 Concluding Remarks 185

7 Complexity of Modal Logics with Recursion 187
7.1 Introduction . . . 187
7.2 Definitions and Background . . . 189
7.2.1 The Multi-Modal Logics with Recursion . . . 189
7.2.2 Known Results . . . 192
7.3 Complexity Through Translations . . . 194
7.3.1 Translating Towards $\mathbf{K}_k$ . . . 194
7.3.2 Embedding $\mathbf{K}_n^\mu$ . . . 204
7.3.3 Complexity Results . . . 206
7.4 Tableaux for $\mathbf{L}_k^\mu$ . . . 206
7.5 Conclusions . . . 214

III Epilogue 217

8 Closing Remarks 219
8.1 Summary of the Contributions . . . 219
8.2 Future Work . . . 222
8.2.1 Towards a Complete Axiomatization of CCS . . . 222

Bibliography 227

Part I

On the Shoulders of Giants

Chapter 1

Introduction

Developing computing systems that do reliably and efficiently what they were designed to achieve has consistently been one of the key challenges in computer science.

A leading approach for checking whether a computing system offers its intended behavior is formal verification [184, 185, 145], which uses mathematical tools to eradicate human error. Formal verification has been used not only for sequential programs but also substantially to provide guarantees for systems with concurrent components. However, applying formal verification techniques in the concurrent setup is becoming increasingly complex as a result of the variety of systems and implementations that utilize non-linear models of computation. At the same time, system correctness is now more relevant than ever because of the energy cost of large-scale computing systems, their omnipresence, and the extremely high financial and personal stakes depending on them. The above reasons demand that software is of high quality, very trustworthy, correct, and robust. Unfortunately, concurrent computation is inherently more complex than its linear counterpart. Thus, it has been a long-standing tradition to understand the mathematical nature of such computations and verify their behaviors through formalization and rigorous proofs.

Over the years, many varied approaches have been aiming to achieve the aforementioned goals, with formal proofs being a common denominator among all. We will briefly discuss a non-exhaustive list of attributes that characterize these approaches, in order to introduce our fields of research. This characterization will help in understanding our contributions and is also useful when designing new methods, combinations of existing approaches, or identifying unexplored areas of research. We limit ourselves to mention the following factors that affect formal verification procedures:

- Whether one *has access* to sufficient information about the system, e.g., in the form of a faithful model.

  Conversely, one might have to perform their analysis by assuming the system is a black box. This attribute is, for example, what separates the approaches of model checking ([74]) and runtime verification ([47]), as we will see later on.

- Whether one aims to *build* a "correct-by-design" system — something we call the *synthesis* problem ([88, 109]) — versus studying an already existing one.

  In the first case, one starts from a *specification* and aims to construct a *model* for it if one exists. The decidability and complexity of this question will become very relevant later on when we study modal logics ([59]).

- Whether one aims to create new specification languages that capture different aspects of computation (such as execution time, space requirements, and agent knowledge).

  Conversely, much research takes place over a fixed specification language, where the analysis aims to study systems against fixed properties expressed in it.

- Whether one uses the same language to describe the systems *and their specifications* versus using different ones.

  The first option (referred to as the one-language approach) is the field we will study for most of this thesis. The second option is often called the two-language approach.

The above characteristics give rise to scientific disciplines, that not only have many flavors within themselves but also produce a plethora of sub-practices when combined. All the contributions we will describe later will be a combination of some of the aforementioned characteristics. We will give a more detailed and technical overview of all fields relevant to any of our contributions in the Preliminaries (Chapter 2). However, before doing so, we discuss one common prevalent attribute in our work, namely, the *syntactical* style of analysis.

In all works included in this PhD thesis, we focus on using *syntactical* descriptions of the objects we analyze. Therefore, we first describe our objects through some formal syntax and then perform our analysis and processing of those defined objects. Thus, for the reader, when conceptualizing an *object*, it is best to assume a syntactical construct, such as a written piece of code, instead of focusing on its computational meaning. Of course, semantics, that is, the computational meaning assigned to the objects we study, does play some role in our work, for example, when establishing which expressions are "correct" or which transformations are allowed. However, this merely *informs* our procedures, but their nature remains syntactic.

Our first category of contributions in the syntactic analysis of properties of computation is in the field of process algebras ([101]), and specifically, in the equational study of their objects. A key concept underlying this field is that the description of systems, *and their intended behavior* is given in the same formal language. This reduces the question of whether a system "matches" its specification to whether the two can be seen as equivalent under some appropriate notion of equality or approximation relation. This immediately turns formal verification questions into questions about the validity of equations or inequalities. Additionally, it means that by using the mathematical theory underlying the provability of equations, one can potentially identify *all* objects that satisfy a specification by exploring all available valid proofs. Finally, it can allow us to decide whether the above tasks are impossible under certain assumptions.

Chapters 3 and 4 present our results over the equational theories of two distinct process algebras. In Chapter 3, we study a process algebraic language that describes systems whose behavior considers priority-based execution of the available actions. Here we answer questions regarding the provability and decidability of whether two systems behave the same under some suitable notion of equivalence. We also discuss the complexity of answering this question. Chapter 4 applies those techniques in the study of the algebraic theory of monitors, namely over a syntax that describes objects aimed to be used for runtime verification (a purpose that affects the validity of equations between monitors).

However, as one sees from those two contributions, this type of analysis can become lengthy and has the disadvantage of being very case-specific. What we mean by "case-specific" is that by tweaking the syntactic constructs one uses or which equations are valid in each case, the whole argument about the provability of equations must be repeated from the beginning. Moreover, even if one has already heavily invested in solving a problem, they will not necessarily have any insight regarding the answer of the same questions over an almost indistinguishable setup (such as the variations we study in these chapters). One would hope that then it is not necessary to study too many variations, but this, unfortunately, is not the case, as formal methods constantly have to catch up with the variety of emerging applications.

Thus, it becomes apparent that, even though isolated results are helpful, they still require dedication, creativity, and expertise, and we need to develop more general approaches when possible. Examples of this trend can be found in computability theory, where establishing the undecidability of several computational problems on a case-by-case basis was replaced by Rice's theorem ([186]), and the search for fixed-parameter tractability gave rise to Courcelle's theorem ([79, 90]). This necessity, along with the thirst for knowledge itself, leads our research from the case-by-case analysis to a spectrum of results we describe as lifting results. There, we use established results to identify the core similarities and difficulties between applications (variations) to create new results and extend the validity of existing ones. As one can see, even by the decrease in the length of individual proofs in Chapters 5 and 6 of this thesis, these methods are typically more general and succinct, when they can be applied.

Chapter 5 focuses on the differences between two kinds of equivalences: strong and weak. Strong equivalences (mostly strong bisimulation [152] in process algebras) require that processes can "match" each other's behavior, including internal steps, in order to be considered equivalent. Conversely, weak equivalences abstract away from the internal computation of a system and only focus on its interactions with the environment. Here we prove some general properties of equational proofs and use them to generalize a known result from the study of strong behavioral equivalences to that of weak ones.

Having transferred this result into a new setting motivates us further to employ similar techniques elsewhere. Therefore, in Chapter 6, we emphasize even more on lifting results. Here we study several process languages, each with different power to model systems, and focus on finding a common structure between them. This is done by utilizing a known but not very widespread lifting technique [12], which revolves around the notion of a *reduction mapping*. A reduction mapping is meant to be a projection of objects between languages, which, as long as it satisfies specific properties, ensures that results on the equational questions we have been studying are preserved between target and source language. Thus, after constructing such mappings between several theories and a given target one, we manage to extend a specific equational result over all the languages we consider. This technique both provides us with these results and also helps us avoid performing the more traditional analysis demonstrated in Chapters 3 and 4 for each theory.

Finally, we continue our study of syntactical analysis techniques in the setting of modal logics. Modal logic is also closely associated with the formal verification of sequential and concurrent programs. Here, we focus on multi-agent modal logics for two reasons. First, they present the most similarities to concurrent computation, and second, their expressive power captures some of our target research beyond the scope of this thesis. The objects studied here become logical formulas, used as specifications in the two-language approach of formal methods. In this context, checking whether the system satisfies a specification is no longer a matter of equality checking. It is instead tackled through model checking and algorithms for answering provability, satisfiability, and validity of formulas written in some logic.

Chapter 7, focuses on the satisfiability problem, which asks whether a given formula *has any* model. This problem can be associated with the synthesis problem in computer science, as the construction of a model could also be turned into an implementation of a “good” system. Therefore, it is essential to be able to compute a model — and hopefully not too inefficiently. One way to tackle this complexity question could be to demonstrate an algorithm and prove through its semantics that it is correct.

However, there are already existing complexity results about logics similar to the ones we wanted to analyze. Thus, our method here focused on identifying syntactical ways to relate our new logics with these existing ones. We managed to do so by defining and using syntactical manipulations between modal logic formulas that preserved our aimed complexity results. Here, again, we avoid more traditional approaches that would prove those complexity results individually. We instead substitute the sheer volume of those proofs with shorter ones that identify and exploit similarities between setups. Our technique here is original, and even though still a rough version of what we believe it could become, it manages to provide us with a variety of valuable results.

Summarizing, this dissertation offers several original contributions to the study of syntactic approaches in the field of formal verification of systems correctness. We provide answers to questions regarding the equational logic of processes, and the complexity of modal logics with recursion. Almost all of our results are negative in nature and demonstrate the impossibility to perform efficiently, or even to perform at all, several tasks over the mentioned fields. During our efforts we switch from case-by-case analysis to more general approaches that aim to generalize results via identifying structural similarities between setups. This leads us to produce results in a more succinct manner, and eventually to design lifting techniques ourselves.

1.1 Our Results

The contributions we will present in this thesis produced the following results in the fields discussed above:

- Bergstra and Klop's basic process algebra (BPA) enriched with an operator allowing the priority-based execution of available actions does not afford a finite axiomatization modulo order insensitive bisimilarity (Chapter 3).

  The results reported in this chapter describe the work contained in [17, 19].

- Monitors, produced with a specific recursion-free algebraic syntax with no variables over a finite set of actions, afford finite axiomatizations, both modulo verdict equivalence and its asymptotic version. However, the landscape of the axiomatizability results is more varied in the setting of open terms — that is, terms that may include variables. In the presence of more than one, but finitely many, actions, we prove that no finite axiomatization exists, and we provide an infinite one that we prove complete. We then perform an exhaustive study for the remaining cases for terms over one action or an infinite set of actions, modulo both equivalences, and show that over a singleton action set, we can acquire completeness with finitely many axioms. In the case of infinitely many actions, our axiomatic basis for closed terms extended by only one axiom is also complete for open terms (Chapter 4).

  The results reported in this chapter describe the work contained in [28].

- We extend a celebrated negative result by Moller [157] over open terms in CCS modulo bisimilarity to the weak setting modulo rooted weak and rooted branching bisimilarity (Chapter 5).

  This result was presented in the invited paper [23], as an original contribution.

- We further study Moller's negative result and prove it applies to several new settings over languages obtained as extensions of BCCSP, which is a basic process algebra for the description of finite synchronization trees. The technique we employ here is based on creating mappings from different process algebraic languages onto Moller's CCS. Our main contribution is identifying the core similarities between behaviors of the used operators in each variation and, by exploiting those, defining the mappings mentioned above. Our first lifting of Moller's result is BCCSP extended with the merge operator $|_A$ from CSP, where A denotes a subset of the actions available in the system. We consider the cases where the resulting process algebras contain exactly one such operator for a fixed A or when they contain all such operators, one for each A. In the first case, we show that BCCSP extended with those operators does not afford a finite ground-complete axiomatization when $A \subset \textsc{Act}$. When $A = \textsc{Act}$, instead, we provide a finite ground-complete axiomatization for bisimilarity. Moreover, in the second case, we show that without the presence of τ actions, our proof technique through mappings could not be applied to prove the non-existence of a finite ground-complete axiomatization. Finally, we create two new mappings and use them to prove two negative results, one for the restriction- and recursion-free, and one for the relabeling- and recursion-free fragments of CCS (Chapter 6).

Part of the work reported in this chapter describes the contents of [29]. The final result regarding the non-finite axiomatizability of restriction- and recursion-free CCS is an original contribution of this thesis.

- Finally, in Chapter 7, we define the multi-agent modal μ-calculus over frames that satisfy the modal axioms corresponding to reflexive, symmetric, serial, transitive, and euclidean models. We then study the complexity of the corresponding satisfiability problems. We manage to produce tight lower and upper bounds for several sub-logics employing many such axioms and at least some hardness results for other ones. We also provide sound and complete tableaux for these logics and use them to prove the decidability of the satisfiability problem in cases not addressed in the literature. However, perhaps the most significant contribution of this final study was our novel methodology, which advocates for a more uniform tackling of complexity questions over modal logics.

  The work reported in this chapter describes the contents of [27].

Aside from the results described in Chapters 3-7, the work done in the course of this PhD program produced the results described in [26], and [21]. We chose not to include these two works in this thesis, as their content, even though relevant to the field of formal methods, did not directly fall in line with the theme of this dissertation. However, the work reported in them did motivate the author to explore new fields.

1.2 The Role of Negative Results in Formal Methods

As one can notice, most of the results presented in this thesis can be seen as *hardness results*. This means that each one proves, in some sense, that a particular task is impossible, very complicated, very slow, or something in between. It is possible to think that such results do not contribute much to the grand scheme of recent advances in formal verification, where faster, safer, more agile software is constantly produced, and impossibility does not "match" with the general, arguably optimistic, attitude.

However, hardness results contribute a great deal to the foundations of several fields, and surprisingly enough, formal hardness guarantees find themselves practical applications from time to time [179, 106, 102]. When presenting our results, we will discuss their importance on an individual level, while for now, we will limit ourselves to a more general overview of the importance of negative results in the field of formal methods. We begin by setting the stage with a quote from Christos Papadimitriou from [164, page 16]. In his words:

> ..negative results are the only possible self-contained theoretical results.
> . . . A related point is that successful exploratory theoretical research is

> bound to produce predominantly negative results. After all, delimitation (discovering that "that's all there is!") is the ultimate success in exploration. Identifying the limitations of a model is valuable information for further model-building, and for crystallizing the subject.

To elaborate, first of all, negative results are useful because they are *results*. What we mean by this is that in a field like theoretical computer science, where conjectures of solutions without actual proofs are not uncommon (see, for example, exponential time hypothesis — ETH — and strong ETH [131, 69]), a hardness result can be a breakthrough. Before such a conjecture is resolved scientists tend to explore many possible versions of reality, in hopes both for useful results, but also of potential contradictions that would resolve the original claim. For example, this search when trying to resolve the famous P versus NP problem has created a plethora of sub-fields which utilize different assumptions, such as access to random bits, a bounded instance space, fault tolerance and others [102]. Thus, a hardness result can end this natural branching of explorations (see, for example, the large amount of research exploring the potential, but not yet provably fruitful, use of *randomness* when trying to tackle NP-hard problems [132]) and help focus the research efforts on the "correct" version of reality, where potential theorems would find the most practical application.

Secondly, a hardness result implies the existence of a proof. In the field of formal methods, new proofs and proof techniques are essential not because of the result they prove necessarily, but also because they show us *why* something is hard, and how to go around it. For example, in the field of process algebra, non-finite axiomatizability results have justified the introduction of auxiliary operators — like Bergstra and Klop's left and communication merge [55, 159, 11]. Moreover, a proof itself has the possibility to be applied elsewhere. For example, an old proof technique used to produce a (possibly expected) result may not be as big a contribution as a new proof technique developed to answer (possibly negatively) a long-standing question. With such a new proof, which say constructs a counter-example for a given theorem, we could also potentially construct a useful witness solution to some other problem.

Finally, a negative (or hardness) result can have applications itself in other fields. For instance, complexity results have been applied for a long time to provide safety guarantees for cryptography protocols, where it is vital that the task an attacker must perform in order to break an encryption has no easy solution. Even more so, lately, with newer developments in blockchains, specific proven hard tasks are associated with the so-called "proof of work" of an agent, which guarantees fairness in resource distribution. Moreover, through fine-grained complexity, more and more hardness results are finding application to fields such as cryptography and cryptocurrencies (see, for example, [42], for a recent application), as the fine-grained analysis of problem complexity allows for tractable problems also to be used to provide proof-of-work guarantees. Besides the above, arguably well-known, applications, one can also see the large wave of research that was triggered from the result of Daskalakis and Papadimitriou, regarding the hardness of the Nash equilibrium problem [84]. That hardness result led to applications both within computer science, but also in other fields such as economics, and social and behavioral sciences [165].

1.3 Historical Notes

Hoare [128] and his axiomatic view of computer programming changed the way we view computation. With the purpose of verifying a program's behavior, logical formulas of the form $P\{S\}Q$ were introduced, with the meaning that if a precondition P holds before the execution of program S then Q will hold when S terminates. This type of theory supports the pre-existing idea of a program being an input-output mechanism, and also enables its further decomposition into smaller programs (commands), that are executed in series, such that the outcomes of one affect the next ones. Such modeling of a sequential execution is ideal for verification as the mathematical objects involved are clear and well-defined, down to the most basic program commands. Through breaking a program execution into sub-executions we were first able to start viewing a computation as a series of states with transitions between them, rather than merely an input-output mechanism, an approach that was later referred to as *labeled transition system* (LTS) based reasoning. A first attempt to extend this LTS based reasoning to concurrent programs came from Ashcroft [34], who also introduced the concept of invariance throughout the execution of a program. Soon after, a first version of parallel composition came into play in the seminal work by Owicki and Gries [162] towards generalizing Hoare's method to concurrent programs.

The above attempts all were in the realm of describing programs formally, but they lacked in formalizing the ideas of what a "correct" program is. Another significant advance in the world of concurrent program verification came with the adaptation of Prior's Temporal Logic [173, 119, 172] to a specification language by Pnueli [169] in 1977. Temporal logic was not a new concept at the time, and it goes back even to the time of Aristotle, who studied modal logic (the predecessor of temporal logic) in his quest for reasoning about necessary and possible truths. However, it was only with a new wave of research starting with Kripke in the late 1950s, who assigned formal semantics to modal logic statements, that modal logic left the realm of philosophy of mathematics and started being treated as a concrete mathematical tool. Kripke's semantics belonged to a wave of research that began replacing the *total* truth value of sentences with one that was associating them instead to *worlds* [127]. As a result, the semantics described by Kripke for temporal logic corresponds to a state-based program execution model and is therefore ideal for studying properties of program behaviors. Intuitively, a formula associated with a state of the program's execution is an assertion about its future behavior.

Temporal logic gave us some very important capabilities. The first was that we could specify the abstract properties one would require of programs, without having to do so though a specific program formalism. Moreover, temporal logics allow one to specify and reason about properties of non-terminating systems. Most reactive systems, that is, systems that compute by maintaining and ongoing interaction with their environment, are best viewed as having "infinite" lifespan. However, the original version of temporal logic was not expressive enough and therefore was extended with various temporal operators such as *until* and *next* [135]. These operators were satisfactory to express important properties such as “It is always the case that a sent action is followed by a receive action”, and “The program never encounters a deadlock”. [92, 115].

After Pnueli, temporal logic became a natural and flexible choice for writing program specifications. This approach was called axiomatic specifications, as the method was to write down a set of formulas (called axioms) that should hold for an execution. However, one also needed a methodology for proving that a property is valid during a specific execution. The axiomatic approach seemed to lack strength in this sector. An attempt to bridge this gap came with the introduction of operational specifications by Lam and Shankar [144] in the 80s. In this approach, an abstract program is given as a specification of what another, specific, program *should* do. Such specifications were closer to the programs for which they were specified, and stricter in the sense that a program had to behave exactly like the specification and not just satisfy it — something that assisted in the design of verification mechanisms.

However, for both these specification frameworks to truly develop there was need for abstract program description languages, that would be able to fully define programs, and yet with enough abstraction to make them clear also as specifications. In the field of formal verification of concurrent systems objects with this functionality had already been emerging, starting with Hoare’s Communicating Sequential Processes (CSP) [129] in 1978, which was followed by the proposal of the Calculus of Communicating Systems (CCS) [153] by Milner in 1980. These were the first algebraic approaches to event-based description of concurrent systems and their specifications, which tried to replace state-based reasoning with new proof techniques. Most importantly, in the methodology supported by CCS and CSP, systems and specifications were stated in the same language, enabling the viewing of correctness proofs as equivalence or preorder checking. This marked the start of a new era in the world of formal verification as it provided the toolbox of algebraic methodology and equational logic in program verification.

Today CCS and CSP are two leading modeling languages for concurrent systems. In addition, many variations of these have been introduced, the most important ones being BCCSP, ACP, BPA, the Meije Calculus and the π-Calculus, among others [55, 57, 53, 126, 35, 155], with the purpose of capturing behavioral aspects of different systems. These modeling languages are based on a finite set of carefully chosen operators that can be used to create new terms and revolve around the idea of parallel composition, which corresponds to two programs executing “concurrently”. Since all of these languages have an algebraic structure and a formal semantics, they encourage the compositional model-based correctness analysis of concurrent systems, and automated verification tools [146, 76, 114] have been developed based on their theory.

A principal methodology in this research field is the search for equational axiomatizations modulo a notion of equivalence over some process description language. The significance of this method is witnessed by the literature on this subject over the last forty years. (See, for instance, [7, 36, 37, 55, 68, 197, 124, 126, 130, 154, 152] for early references as well as survey and accounts, and the papers [25, 20, 108, 136] for examples of the rich body of recent contributions to this field.) This research avenue has its intellectual roots in the time-honored study of the existence of finite (conditional) equational proof systems for equality of regular expressions, as presented in, for instance [78, 140, 141, 177, 182].

There are manifold reasons for studying equational axiomatizations of equivalences over processes. For example, the existence of a finite, or at least finitely specified, equational axiomatization for some notion of process equivalence is often considered one of the yardsticks to assess its mathematical tractability. Additionally, equational axiomatizations provide a purely syntactic description of the chosen notion of equivalence over processes and characterize the essence of a process semantics by means of a few revealing axioms. These syntactic descriptions can be used to compare a variety of semantics in a model-independent way (as done, for instance, in [197]). Moreover, such axiomatizations pave the way for using theorem-proving techniques to establish that two process descriptions express the same behavior modulo the chosen notion of behavioral equivalence [80, 112, 149] and play an essential role in the partial evaluation of programs [123].

At the same time, computer scientists were also exploring formal verification in the setting where a system and its specification languages need *not* be equivalent, but the system is just one of the possibly inequivalent implementations of the specification. There, the system will contain more implementation details than the specification and therefore it would not be possible to view it as equivalent to the specification directly. In the single-language approach, this more relaxed view of system correctness is supported by the use of preorders as yardsticks for correctness - see for instance the failures, and testing preorders [70, 61, 85] and the taxonomic treatment given by van Glabbeek in [193, 194]

Another refinement technique, which had been introduced in the sequential setup even from the time of Dijkstra [89], revolved around rewriting the abstract program through a series of sound transformations to end with a final concrete implementation. The validity of said transformations would be defined through studying the computational meaning assigned to atoms of abstract syntax. In other cases actual software was created that was able to study the models of actual systems and verify them against a given specification, usually given in some temporal logic logic. These methods, which were pioneered by Clarke, Emerson, Queille, and Sifakis [73, 92, 93, 174], initiated the field of *model checking*, which is now considered one of the most fundamental approaches in formal verification, and, as the name suggests, is heavily dependent in the existence of a *model*.

However, due to the possibility of not having access to such a model of a system (for example the case of proprietary software), computer scientists also had to develop methods for formal verification that did not require this knowledge. A relatively newer method in the formal study of programs is the field of runtime verification (RV) [14, 48, 103]. RV is based on studying an execution (either online as it occurs or offline in the form of logs) rather than analyzing a system model and is also an instance of the "two-language" approach, as the specification is usually given as some temporal property, while the analyzed object in this case is an execution trace.

The name runtime verification was first introduced in a workshop, as a complementary technique to model checking in 2001, but it quickly evolved into a field of its own. RV's actual origins were earlier than that, with work done in the spectrum of space exploration in the NASA labs, where a formal specification was verified over the logs of single executions of Java programs [122]. Since those times, the field has bloomed, and now RV is an established formal verification technique with various extensions and applications. Most recent directions in this field include attempts to handle parallel program executions [188, 31, 64, 97] by utilizing more complex representations of trace events, more complex specification logics, and adapting old techniques from the literature on the formal verification of concurrent systems.

Chapter 2

Preliminaries

This chapter gives a short overview of techniques and background notions necessary to present our results. The subjects on which we focus in this thesis are equational logic, process algebra, runtime verification, and modal logic. In the first three, the models we ultimately study are labeled transition systems, while when discussing modal logic, we transition to the model of Kripke structures which are also used in formal methods, but their theory is less bound to computer science. Moreover, in this overview, we also move from the process to the monitoring verification setup and we use this move to highlight the modifications on proof techniques, semantics and equivalences that are relevant in each case.

We have selected the following organization to deliver the necessary background for the above-mentioned subjects. First, we present equational logic as a general subject without introducing specific constructs for forming terms and, subsequently, equations. We also do not select models for the terms or a semantic interpretation of equality. Then we introduce labeled transition systems — a semantic model that underlies the algebraic languages we present and study later — and bisimilarity as a notion of equivalence. Having established these two, we move to process algebra and explain its basic theory, some core examples, and the mechanism that connects process algebras to LTSs, i.e., structural operational semantics. We then present a baseline framework, methods, and research questions that occur through the interplay of process algebra and equational logic when the terms of equational logic are formed through some algebraic syntax. Before moving on, we also introduce the basic notions of runtime verification and specifically focus on the branch of their theory called *regular* monitors. Regular monitors make up the branch of RV that demonstrates the most similarities with process algebra, something that will enable us, later on, to use the techniques of equational logic in the context of RV. Finally, we present the field of modal logic, which is relatively stand-alone, and its semantics are not operational in nature.

$$(\textsc{Ref})\ t = t \qquad (\textsc{Sym})\ \frac{t = u}{u = t} \qquad (\textsc{Tran})\ \frac{t = u \quad u = v}{t = v}$$

$$(\textsc{Subs})\ \frac{t = u}{\sigma(t) = \sigma(u)} \qquad (\textsc{Cl})\ \frac{t_i = t_i',\ i = 1, 2, \ldots n}{f(t_1, \ldots t_n) = f(t_1', \ldots, t_n')}$$

Table 2.1: The inference rules of equational logic, for an n-ary $f \in \mathcal{F}$.

2.1 Equational Logic

Equations are expressions of the form $t = u$, where t and u are terms built from a collection of operator symbols and variables that range over the domain of interest. Many mathematical structures and especially algebras can be described by sets of equations called axioms that are valid based on the interpretations given to the operator symbols in the theory's models. In order to acquire such axioms one uses the interpretation of algebraic operators in a model, and then identifies equal models, which in turn establish the validity of the quality predicate $(=)$ between terms.

For example, in the theory of regular expressions, the $\cdot$ operator is not commutative, while over the theory of integers it is, which means that the equation $x \cdot y = y \cdot x$ is valid over the integers. This equation, would not necessarily hold over *any* kind of binary operation f over the integers (such as exponentiation, e.g., $f(1, 2) = 1^2 \neq 2^1 = f(2, 1)$). Both the model and the interpretation of the operations are important when determining the validity of equations.

Formally equational logic is a fragment of first order logic which contains variables, constants, and operator symbols, but no connectives and no quantifiers (apart form i1mplicit universal quantification over the values of variables). The only predicate used is that of equality $(=)$. Formulas are built from the equality predicate connecting first-order terms, i.e. well-formed expressions constructed from a set $\mathcal{F}$ of function symbols, called a *signature*, and the set $\mathcal{V}$ of variables. The set of these terms is denoted $T(\mathcal{F}, \mathcal{V})$. Variables occurring in a term may be instantiated by substitutions, defined as mappings σ from variables to terms. Note here that an equation can be created by associating *any* two first order terms via the equality predicate, but as discussed earlier, one usually is concerned with equations that are valid over the models assigned to terms.

Just like in many logics, at this point we can rightfully wonder what is the advantage of formally describing equations between terms in this way, if our only way of determining whether these equations are "correct" is to semantically verify this claim for every equation. Fortunately, this is indeed not the only aspect of this approach.

In the practice of equational logic, one states sets of equations called *axioms*, such as for example, the law of commutativity we described earlier, that describe specific terms in $T(\mathcal{F}, \mathcal{V})$, whose interpretations in some models are considered equivalent. A set of such axioms is called an axiom system, denoted $\mathcal{E}$, and the set of all equations provable from it via the rules of equational logic, seen in Table 2.1, is called its equational theory, denoted $Th(\mathcal{E})$. After having stated such an axiom system $\mathcal{E}$, one would hope that they have captured the essence of what constitutes *any* two models equal. The goal is thus, for a given a model M (that is, a structure over which terms in $T(\mathcal{F}, \mathcal{V})$ can be interpreted), to find a $\mathcal{E}$ that $Th(\mathcal{E})$ is exactly the set of all equations that are valid in M.

That is, these axioms now can be used as premises into any equational proof and create more equations by utilizing the inference rules of equational logic of Table 2.1, where f is any operation from the signature $\mathcal{F}$, of arity n, and σ is any substitution. We write $\mathcal{E} \vdash t = u$ iff the equation $t = u$ is provable from those in $\mathcal{E}$ using the rules of equational logic. For example, if one has established that $1 + 2 = 3$, over the integers, then by applying rule Sym we can acquire $3 = 2 + 1$.

These derived equations are exactly the ones we concern ourselves with, and are the ones we request our models to satisfy, beyond the initial stated axioms in $\mathcal{E}$. Specifically, Birkhoff's 1935 theorem [58], also referred to as completeness for equational logic, relates provability and validity of equations, that is,

$$\mathcal{E} \vdash t = u \text{ iff } (t = u) \text{ is valid in all models of } \mathcal{E} \ .$$

The above allows us to discover equal models, through the use of the inference rules of equational logic, over a few revealing axioms. Moreover, this theorem guarantees that if we manage to truly identify the axioms that exactly are valid in a specific model of a specific $\mathcal{E}$, then, in that model, equivalent terms will always be provably equal (a property particularly alluring for the practice of the one language approach in formal verification as we will see shortly).

2.2 Labeled Transition Systems and Notions of Equivalence

From the previous discussion, it becomes apparent that, in order to define (and prove the validity of) equations, one needs to study their semantic models, and a semantic notion of equivalence. We now fix and present one such semantic model, which remains relevant throughout our work, namely *labeled transition systems (LTSs).*

Here we chose to present the most general version of LTSs, which also allows for predicates being assigned to the states of a model [4].

Definition 2.1 (Labeled Transition System). *A* labeled transition system *is a quadruple* $(\mathcal{P}, \textsc{Act}, \rightarrow, \text{Pred})$*, where:*

- $\mathcal{P}$ *is a set of* states*, ranged over by* s*;*
- $\textsc{Act}$ *is a set of actions or* labels*, ranged over by* a, b*;*
- $\rightarrow \subseteq \mathcal{P} \times \textsc{Act} \times \mathcal{P}$ *is a (*labeled*)* transition relation*. We use the more intuitive notation* $s \xrightarrow{a} s'$*, instead of* $(s, a, s') \in \rightarrow$*, and* $s \not\rightarrow$ *if there are no* a, s' *for which* $s \xrightarrow{a} s'$*;*

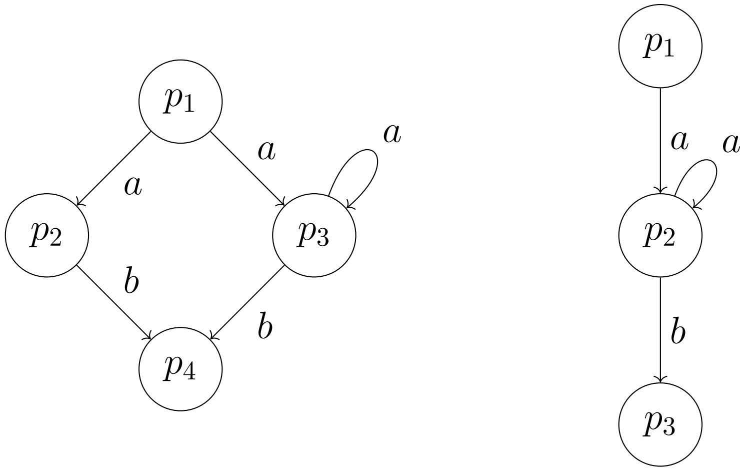

Figure 2.1: Two trace equivalent, but not bisimilar processes.

- $\mathrm{P} \subseteq \mathcal{P}$*, for every* $\mathrm{P} \in \mathrm{Pred}$*. We write* $s\mathrm{P}$ *(and* $s\neg\mathrm{P}$ *respectively) if state* s *satisfies (resp. does not satisfy) predicate* P*.*

The relations $s \xrightarrow{a} s'$ *are called transitions.*

Later on, in our contributions, when a set of predicates is not specified we mean the set of predicates is empty. From now on, in all our work on process algebraic languages, the models are always LTSs. Over these models we will study which states exhibit "equivalent behaviors". Moreover, the LTSs associated with a process algebraic term are also *rooted*, in the sense that there will be a specific root node that is associated with the term, and the remaining nodes are associated with its possible executions.

There are several ways to implement this concept of behavioral equality; for example one could request that equal LTSs must be able to exhibit the same *traces*[1]. The above equality is called *trace equality.* It is also a common request that LTSs that are considered "equal" have the same *depth*[2]. However, as was pointed out by Milner in his early work on CCS, there are many LTSs that demonstrate the exact same traces, but would not behave the same in the way we can observe them.

An example of this phenomenon is given in Figure 2.1, where the two LTS can exhibit the same traces, but only one of them can reach a state after reading an a, where it can no longer observe any more a's. This could mean that when interacting with a user that wants to perform two a actions in sequence, the first LTS could *deadlock* after the first a, while the second one would not. When studying concurrent systems this type of issue is very prevalent as we are mostly interested in the way systems behave when put in a certain environment. Therefore other notions of equivalence were introduced, in order to capture these intuitions [193].

[1] A trace is a series of actions obtained by starting at a root node and following valid state transitions on an LTS.

[2] The depth of a process p, denoted $\mathrm{depth}\,(p)$ is the length of a longest trace its LTS can demonstrate (and is infinite if no longest trace exists).

In the case of process algebra, and in most of the work in this thesis, the most wide-spread notion of equivalence is that of *bisimilarity* [166, 152] which is a formalization of the intuition that two LTSs are equal when they behave the same while interacting with any environment. This notion will allow us later on to study which process terms correspond to equal LTSs and to define the equations between such terms to use as axioms. In this thesis we use the classic definition of bisimilarity [152, Chapter 4, Definition 1], where we also consider predicates assigned to the LTS's states [4].

Definition 2.2 (Strong bisimulation). *A binary relation R over the set of states of an LTS is a bisimulation iff whenever $s_1 R s_2$:*

- *for each action a, and state s_1', if $s_1 \xrightarrow{a} s_1'$, then there is a transition $s_2 \xrightarrow{a} s_2'$ such that $s_1' R s_2'$, and*
- *for each action a, and state s_2', if $s_2 \xrightarrow{a} s_2'$, then there is a transition $s_1 \xrightarrow{a} s_1'$ such that $s_1' R s_2'$, and*
- *$s_1\mathrm{P}$ iff $s_2\mathrm{P}$, for each predicate P.*

Two states s_1 and s_2 in an LTS are bisimilar, notation $s_1 \sim s_2$, if they are related by a bisimulation R.

It is well known that $\sim$ is an equivalence relation over $\mathcal{P}$, and it is the largest bisimulation relation [152, Chapter 4, Proposition 2].

Remark 1. *Bisimilarity preserves the depth of LTSs, i.e., whenever $s_1 \sim s_2$, then* $\mathrm{depth}\,(s_1) = \mathrm{depth}\,(s_2)$*.*

As one can see from the above definition, all actions that a state can perform must be matched by an identical action from any state bisimilar to it, regardless of whether those actions are "observable" to the environment. However, in many applications, some of the actions that systems perform are either not observable externally or are not relevant for establishing some correctness property. Such actions on an LTS are given the name τ. This setting gave rise to the equally important notion of of equivalence, namely *weak bisimilarity.* To define it, we utilize a more "relaxed" version of a transition, namely (we use α to range over $\textsc{Act} \cup \tau$):

Definition 2.3 (Notation). *For each action $\alpha \in \textsc{Act} \cup \{\tau\}$, we write $s \overset{\alpha}{\Rightarrow} s'$, if*

- *$\alpha = \tau$ and $s(\xrightarrow{\tau})^* s'$, or*
- *$\alpha \neq \tau$ and there are processes s_1, s_1', such that $s(\xrightarrow{\tau})^* s_1$, $s_1 \xrightarrow{\alpha} s_1'$, and $s_1'(\xrightarrow{\tau})^* s'$.*

The adaptation of bisimilarity to a weaker version that "ignores" the τ actions is given in the following definition, where we assume no predicates over the states of an LTS (as we will not study any such combination in our contributions):

Definition 2.4 (Weak bisimilarity). *A binary relation R over the set of states of an LTS is a weak bisimulation iff whenever $s_1 R s_2$ and α is an action (including τ):*

- *if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \overset{\alpha}{\Longrightarrow} s_2'$ such that $s_1' R s_2'$, and*
- *if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \overset{\alpha}{\Longrightarrow} s_1'$ such that $s_1' R s_2'$.*

Two states s_1 and s_2 in an LTS are weakly bisimilar, notation $s_1 \approx s_2$, if they are related by a weak bisimulation R.

Similarly to strong bisimilarity, it is well known that $\approx$ is an equivalence relation over $\mathcal{P}$, and it is the largest weak bisimulation relation [152].

This definition concludes our introduction of the necessary notions one needs in order to apply equational reasoning to LTSs. We now proceed to present process algebras, which will be the languages in which we describe terms, and show how those are turned into LTSs and instantiate all of the above notions.

2.3 Process Algebra

The introduction of concurrency into computer science and the necessity of modeling the systems that implement it led to the development of a variety of modeling languages, sharing some important characteristics and generally referred to as process algebras. The main idea behind process algebra is to describe the communications, interactions and synchronizations between independent agents, such as programs or processes, as terms built from a collection of operators that can be used to construct new system descriptions from already built ones. The abstract model of a Turing machine and all of the equivalent models of computation, although ideal for describing sequential computation gave little insight as to how to capture these concepts of parallel systems. (We note, in passing, that there have been some interesting proposals of variants of Turing machines that try to model reactive computation [39, 107]. However the established methods are algebraic in nature).

The most famous process algebras are, arguably, Milner's Calculus of Communicating Systems (CCS) [153], Milner, Parrow, and Walker's π-Calculus [156], Hoare's Communicating Sequential Processes (CSP) [129] and Begstra and Klop's Algebra of Communicating Processes (ACP) [55]. The syntax of each process algebra builds on a set of actions, which represent atomic computational tasks a concurrent process can perform, either locally or as communications with other systems. In most cases one has also access to a set of variables. Then there is a set of operators that define the way that existing terms (denoted by p, q) of the syntax can be composed. The most common operators are:

- $a.p$ of arity 1, where a is one of the atomic actions, called **action prefixing**. Action prefixing corresponds to the idea that the system can perform action a and then behave like the following term p.

$$(r_1)\ \frac{}{a \xrightarrow{a} \surd\!\!\surd} \qquad (r_2)\ \frac{p \xrightarrow{a} \surd\!\!\surd}{p \cdot q \xrightarrow{a} q} \qquad (r_3)\ \frac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q}$$

$$(r_4)\ \frac{p \xrightarrow{a} \surd\!\!\surd}{p + q \xrightarrow{a} \surd\!\!\surd} \qquad (r_5)\ \frac{q \xrightarrow{a} \surd\!\!\surd}{p + q \xrightarrow{a} \surd\!\!\surd} \qquad (r_6)\ \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad (r_7)\ \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'}$$

Table 2.2: SOS rules for BPA with the termination predicate.

- $p + q$ of arity 2, called **alternative composition** or **non-deterministic choice**. This operator corresponds to the idea that the system can behave like either one of p and q during its execution, with the choice happening non-deterministically upon performance of the first computation step of one of its arguments.

- $p \cdot q$ of arity 2, called **sequential composition**. Sequential composition is an extension of action prefixing. Intuitively, $p \cdot q$ stands for a process that behaves like p and, if and when p terminates, it continues by behaving like q.

- $p \| q$ of arity 2, called **parallel composition**. Intuitively, $p \| q$ describes a process that can interleave the computational steps of its arguments arbitrarily. Moreover, the processes p and q can interact by suitably synchronizing their individual computational steps when those are, in some sense, "compatible".

By way of example, we present below the syntax and operational semantics of BPA and CCS. Both of these process algebras will be used later on when we present our contributions.

The syntax and semantics of BPA The collection of *process terms* t in BPA [55] is generated by the following grammar:

$$t ::= a \quad | \quad t \cdot t \quad | \quad t + t \quad | \quad x$$

with a ranging over a set of actions $\textsc{Act}$, t ranging over process terms, and x ranging over a set of variables $\mathcal{V}$. We let $\mathbf{P}$ denote the set of BPA processes and let $p, q, \ldots$ range over it. The semantics of BPA is given by a labeled transition system (Definition 2.1, with only available predicate the one of termination $\xrightarrow{a} \surd\!\!\surd$), meaning that processes are interpreted as edge-labeled directed graphs. The SOS rules for BPA (given in Table 2.2) yield transitions of the form $p \xrightarrow{a} p'$ that express that term p can evolve into term p' by the execution of action a and predicates $p \xrightarrow{a} \surd\!\!\surd$ to express that term p can terminate successfully by executing action a.

The syntax and semantics of CCS In Milner's CCS the formal syntax is given over a set of action names $\textsc{Act}$ (also referred to as channels when interpreted as

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \| Q \xrightarrow{\alpha} P' \| Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \| Q \xrightarrow{\alpha} P \| Q'} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'}$$

$$(\alpha, \overline{\alpha} \neq b)\ \frac{P \xrightarrow{\alpha} P'}{P \setminus b \xrightarrow{\alpha} P' \setminus b} \qquad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad (A \overset{\mathrm{def}}{=} P)\ \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}$$

Table 2.3: SOS rules for CCS ($\alpha \in Act$, $a \in L$).

the way the system interacts with the environment). We use the notation $\overline{a}$ for output on a channel a, while the symmetric input action (called a co-action) is assigned the notation a. The union of the sets of actions and co-actions is referred to as L and it is the set of labels that describe the observable actions of the process. The set L extended with the silent action τ, is denoted $\textsc{Act}_\tau$. I.e., in this setup, $\textsc{Act}_\tau = L \cup \tau$, and will be ranged over by α, while a will range over L. We also use a set of process constants that are used as placeholders for processes that have already been defined. We use A to refer to such a process constant. Given a set of action names $\textsc{Act}$, the set of CCS processes is defined by the following BNF grammar:

$$P := \mathbf{0} \ \mid \ A \ \mid \ \alpha.P \ \mid \ P + Q \ \mid \ P \| Q \ \mid \ P[f] \ \mid \ P \setminus a \ .$$

The nil process $\mathbf{0}$ here stands for a process that is completely inactive; it cannot perform any actions or interact with another process. The constant construct A is relative to a definition $A \overset{\mathrm{def}}{=} P$, where we assume P to be a process already defined in CCS syntax. Thus, the operation assigns the identifier A to the process P. Here, we highlight that P can contain other mentions of A which enables recursive definitions.

The terms $\alpha.P_1$, $P_1 + P_2$, $P_1 \| P_2$ have the standard meanings mentioned above. The expression $P_1[b/a]$ stands for the process P_1 with all actions named a renamed as b (allowed for any action except the silent action τ, and with all complement actions $\overline{a}$ renamed to $\overline{b}$) and the expression $P_1 \setminus a$ stands for the process P_1 where it can no longer perform $a, \overline{a}$ actions. As semantic model for the algebraic process description languages that we will study, we consider the classic LTSs, [137], where in this case, we utilize no predicates (as is also the case with most of our contributions later). The rules that define the LTS giving the operational semantics of CCS expressions are listed in Table 2.3.

Based on bisimulation ($\sim$), over CCS, we can state the equation $x \mid y \approx y \mid x$, where one can verify easily that indeed the two sides are bisimilar. Such equations are called valid over CCS *modulo* bisimulation, and are the kind of of equations we will use later on when developing our equational theories.

An important notion to mention here, regarding equivalences between process algebraic terms is that of a congruence. A congruence is an equivalence R, such that process descriptions that are related by R can be used interchangeably as parts of a larger process description without affecting its overall behavior. This is extremely essential in the equational logic of processes as it substantiates the intuition that within equational proofs we can perform equational steps of the form *substitution of equals by equals.*

As process algebras support the single-language approach to formal verification, behavioral equivalences are crucial since they enable the verification to be done through equivalence checking. The amalgamation of the fields of equational logic and process algebras has given rise to the following questions,among others:

- *Is a collection of axioms* complete*?*

  This means that we are interested in verifying whether all the equations that hold modulo the chosen notion of behavioral congruence can be derived from the set of axioms using the rules of equational logic.

- *Does the process algebra modulo the chosen behavioral congruence afford a* finite *equational axiomatization?*

  This means that we are interested in verifying whether there is a finite collection of axioms for the algebra that is sound and complete.

Unfortunately, of the two notions or equivalence we presented earlier, weak bisimulation is not a congruence over the algebras we consider. This means that when carrying out equational proofs one cannot use compositionality (or induction) on equational proofs. For example, it is easy to check that $\tau.a.\mathtt{nil} \approx a.\mathtt{nil}$, On the other hand, $\tau.a.\mathtt{nil} + b \not\approx a.\mathtt{nil} + b$. For this reason, later on when we try to answer these questions modulo weak equivalences we will study a variation of weak bisimilarity, which is indeed a congruence.

2.4 Runtime Verification

Possibly the most crucial difference that separates runtime verification from other methods is the lack of (a model of) the system. Namely, in this approach, the system acts as a black-box, and we can only observe and analyze its *execution*(usually with a computational entity called a *monitor*). Thus what so far has been a set of actions (an alphabet) used to express system behavior, now switches to the role of a set of *observable events* which occur over time, and form a *trace* (as defined earlier).

Figure 2.2[3] demonstrates a well-established view of the field of runtime verification.

Of course, the fact that we do not have access to a model of the system-under-scrutiny will definitely limit the amount of information we can infer about it by

[3] Image taken from [103], with authors permission.

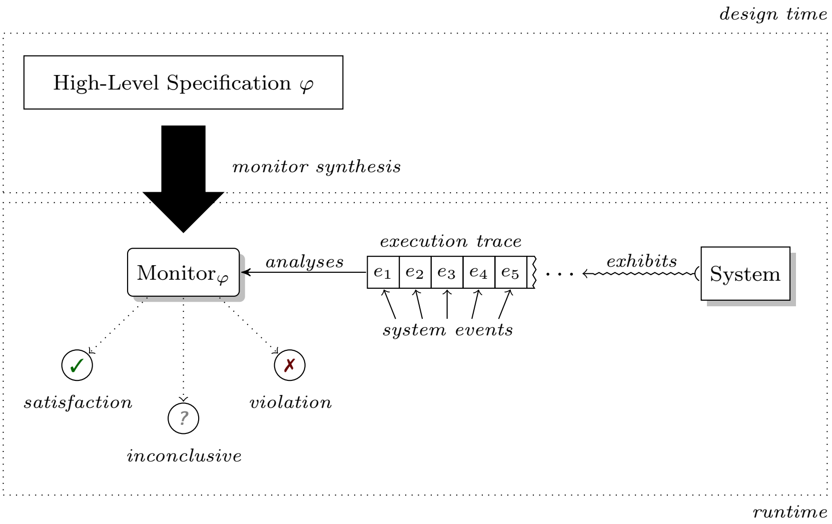

Figure 2.2: The framework of runtime verification

looking only at its traces. Even if we had *all* traces that can be produced by a system, it seems like this information is in a sense "weaker" than having access to a model. What we mean is that given an LTS model of a process algebraic term, one can infer all of its possible executions (traces). However, the converse does not hold. For example looking at the traces that the processes in Figure 2.1 can exhibit does not allow us to distinguish between them. Thus we are in need to tackle the verification of systems, when we only have access to the traces they produce.

Ideally, in order to be able to use all of the already established formal verification methods, we should be able to extract an accurate LTS model for a process though observing and analyzing its traces offline. Such an achievement would be extremely useful, which is why there is substantial research taking place to manage to tackle the task of *model learning* [178, 191]).

However, the bulk of the work in this field does not assume any extraction of such a model, and instead accepts that indeed we will lack the ability to verify all relevant properties over a system. Moreover, there is another particularity in this field, which stems from the specification of properties. Specifically, one has to address whether specifications are aimed for the system or for the trace, i.e., if a given property must hold on *the observed trace* or on *any possible trace* the system can produce. For example, a property such as:

The system never produces an error.

now has to be interpreted over a trace, and not over a system. In this case *if* an error is observed along a trace then indeed we can infer that the system itself violates the property. However if we do not observe an error, we cannot necessarily generalize a successful conclusion over a trace to over a system. Indeed, another execution of the same system could potentially fail. To make things worse, in the case of a possibly infinite execution, we might never reach a point where we can infer such a success for the trace itself. Such details regarding how we interpret systems and their executions play a crucial role when one has to define formally the semantics of properties over traces, and when studying which properties can and cannot be verified at runtime (a study referred to as *monitorability*).

As one can understand from the above discussion, we cannot provide the same correctness guarantees when we have and when we do not have the model of a system. In the second case we have less information. Moreover, since the analysis takes place *during the execution*, we have significantly stricter restrictions on the computational cost of the analysis. This second drawback is not always present in every implementation (for example the analysis of a program log can happen offline), but it is very important in the general case and unfortunately cannot be completely mitigated with studying a system before it is deployed. The reason is that when not in possession of complete knowledge of the system, or some global guarantees, there will always be room for a current execution to behave differently from what has been established about the system so far.

Therefore, if we do possess some kind of mechanism that provides guarantees for traces, it can always be useful to launch it along the current execution of a program. This means that often, systems and their monitors have to share the available computational resources. However, modern software is expected also to be very *efficient*. Thus, since the system and the runtime verification algorithm (monitor) must be executed on the same resources, it is widely requested that monitors be *lightweight*. Lightweight is a term referring to complexity, and corresponds to the cost of execution *per system event*. This is because systems and their monitors cannot be associated to classical complexity notions since they are usually expected to run for an unbounded amount of time. Thus, a monitor is expected to perform a very low amount of processing per system event (ideally constant [175], or even *exactly one* computational step).

For a new monitoring setup to be considered successful, it must not only provide correctness guarantees, but also to impose a very low computational cost on the system it will be launched with. It is left up to the designer to provide guarantees for both these features. Moreover this design must be model-independent, as the monitoring algorithm must be correct no matter the system it will be launched with (since the system is a black-box). This led to the wide popularity of the method for producing monitoring algorithms called *synthesis*. The synthesis is a procedure (or a function) of building a monitoring algorithm based on purely and automatically analyzing the specification. By stating a synthesis algorithm and formally proving it correct one can manage to have correct and efficient monitors, without having to prove those properties independently.

Runtime verification has observed a large growth in research done in many of these topics, and has produced a wide variety of successful monitoring paradigms, specification languages and tools. In our approach we will focus on a specific part of this research that is the most relevant to our contribution, namely *regular* monitors which are algebraic in nature and thus allow us to use equational reasoning techniques to study them. We now give a general overview of the field of regular monitors, for the sake of completeness, while later on, when presenting our

Syntax:

$$\begin{array}{rllllll}
m, n \in \textsc{REMon} ::= & v \in \textsc{Verd} & \mid a.m & \mid m + n & \mid \mathit{rec}\ x.m & \mid x \\
v \in \textsc{Verd} ::= & \mathit{end} & \mid \mathit{yes} & \mid \mathit{no} & &
\end{array}$$

Dynamics:

$$\textsc{mAct}\,\frac{}{a.m \xrightarrow{a} m} \qquad \textsc{mRec}\,\frac{}{\mathit{rec}\ x.m \xrightarrow{\tau} m[^{\mathit{rec}\ x.m}/_{x}]}$$

$$\textsc{mSelL}\,\frac{m \xrightarrow{\alpha} m'}{m + n \xrightarrow{\alpha} m'} \qquad \textsc{mSelR}\,\frac{n \xrightarrow{\alpha} n'}{m + n \xrightarrow{\alpha} n'} \qquad \textsc{mVer}\,\frac{}{v \xrightarrow{\alpha} v}$$

Instrumentation:

$$\textsc{iMon}\,\frac{p \xrightarrow{a} p' \quad m \xrightarrow{a} m'}{m \triangleleft p \xrightarrow{a} m' \triangleleft p'} \qquad \textsc{iTer}\,\frac{p \xrightarrow{a} p' \quad m \not\xrightarrow{a} \quad m \not\xrightarrow{\tau}}{m \triangleleft p \xrightarrow{a} \mathit{end} \triangleleft p'}$$

$$\textsc{iAsyP}\,\frac{p \xrightarrow{\tau} p'}{m \triangleleft p \xrightarrow{\tau} m \triangleleft p'} \qquad \textsc{iAsyM}\,\frac{m \xrightarrow{\tau} m'}{m \triangleleft p \xrightarrow{\tau} m' \triangleleft p}$$

Table 2.4: Monitors, dynamics, and instrumentation.

contributions we focus on a sub-language of the ones presented here.

2.4.1 Regular Monitors

Regular monitors are programs that are defined through a process-algebraic syntax. They were originally introduced by Aceto et al. [104], and in their most expressive form they are stated via syntax in Table 2.4, where $a \in \textsc{Act}$, $\alpha \in \textsc{Act} \cup \{\tau\}$, and $x \in \mathcal{V}$. The terms *end*, *yes* and *no* are called *verdicts* and are what the monitor can produce as a result from analyzing a trace.

Since these monitor programs are described in an algebraic syntax, they can be equipped with structural operational semantics. However, such monitors are meant to be executed *along* a program. Thus, even though they correspond themselves to automata-like entities as can be seen through the dynamics given in Table 2.4, their computational advance is also described when *instrumented* with a process p (an operation denoted as $m \triangleleft p$), and the advancement of this joint computation is given via the instrumentation rules given in Table 2.4.

The verdicts that these monitors can produce for a given execution are irrevocable, and they inform us about whether the given trace is accepted or rejected by a monitor. If monitors are used to check for program properties, ideally those verdicts should tell us whether the observed trace is, or is not, in the semantics of a given formula. Here we will not provide a logic and its semantics over traces (although there is a lot of work done in this direction [201, 103, 51]) because it is not relevant to our work that we will present later.

In our approach we define notions of equivalence between monitors, and study their equational theory. In this setup we focus more on trace based notions of equivalence among monitors, as it seems more relevant whether two monitors “agree” on their analysis of a system execution, rather than whether they behave the same way as "computing systems".

2.5 Modal Logic

Modal logic has been one of the most popular formalisms used for the specification of properties for sequential and concurrent systems. Depending on the interpretation, it can be used to express not only the evolution of a system over time — which is the most relevant case for this work — but also, among others, knowledge, beliefs, and obligations of agents [94, 33].

2.5.1 Syntax

Plain modal logic (M_L) formulas are constructed via the following BNF grammar:

Definition 2.5 (Single-agent modal logic)**.**

$$\varphi, \psi \in M_L ::= p \quad | \ \neg p \quad | \ \top \quad | \ \bot \quad | \quad \varphi \wedge \psi \quad | \ \varphi \vee \psi \quad | \ \Diamond\varphi \quad | \ \Box\varphi \ ,$$

where p is a propositional variable from a set of propositional variables Prop*. Note that the notation p in the past chapters has been used to describe processes, while here it stands for propositional variables. We chose to not adapt this notation to keep it in line with the existing work in both fields.*

From these operations we can also express negation ¬, and logical implication → in the usual way. The monadic connectives ◊ and □ (referred to as diamond and box respectively) are called modalities and are dual, in the sense that $\Diamond\varphi \equiv \neg\Box\neg\varphi$, and they are what really distinguishes modal logic from other formalisms.

When viewed in a multi-agent setup, where we express formulas over a finite set of agents, which we will denote as Act, these modalities become finite sets of modalities, indexed by the name of the relevant agent. The set Act can be either finite or infinite, but so far, in formal methods it is more common to assume a finite Act. The modalities ◊ and □ now become agent specific, and are denoted $\langle a \rangle$ and $[a]$, while the remaining syntax remains the same. The duality property mentioned above now holds only between modalities corresponding to the same agent.

2.5.2 Semantics

For the purposes of this thesis we present here the Kripke semantics of modal logic, which is the most widely used one [60], and the one we will use later on.

Intuitively Kripke models are sets of wordls W, connected by edges through an accessibility relation R, and on each world certain propositional formulas hold. Logical formulas that do not contain modalities are evaluated "locally" in each world, through the propositional variables that are valid on the given world, and their evaluation does not use at all the existence of other worlds, or whether they are accessible or not from the current one. However, when evaluating the semantics of formulas containing modalities, the accessibility relation comes into play. Specifically, the formula $\Box\varphi$ is assigned the meaning that "φ holds in all accessible worlds

$$
\begin{aligned}
&\mathcal{M}, w \not\models \bot\ ,\\
&\mathcal{M}, w \models \top\ ,\\
&\mathcal{M}, w \models p\ \text{, iff } w \in V(p),\\
&\mathcal{M}, w \models \neg p\ \text{, iff } w \notin V(p),\\
&\mathcal{M}, w \models \varphi \wedge \psi\ \text{, iff both } \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi\\
&\mathcal{M}, w \models \varphi \vee \psi\ \text{, iff } \mathcal{M}, w \models \varphi \text{ or } \mathcal{M}, w \models \psi\\
&\mathcal{M}, w \models \Diamond\varphi\ \text{, iff } \mathcal{M}, w' \models \varphi,\ \text{for at least one } w',\ \text{such that } (w, w') \in R\\
&\mathcal{M}, w \models \Box\varphi\ \text{, iff } \mathcal{M}, w' \models \varphi,\ \text{for all } w' \text{ such that } (w, w') \in R
\end{aligned}
$$

Figure 2.3: The semantics of modal logic

” (including the possibility for no accessible worlds), while $\Diamond\varphi$ corresponds to “φ holds in at least one accessible world”. This is how in modal logic one manages to state formulas that make the whole model and its structure relevant for their evaluation, rather than only studying one world.

In formal methods, we associate modal logic formulas to the states of a program model, giving a formal meaning to the general properties we want to verify about the program, or its executions. In this view, boxes and diamonds express that something is *certain* (it will be true in all possible next states of this system), or *possible* (it *can* be true in some next state). By nesting connectives we can then “reach” further into the possible continuations of a programs, and by using recursion operators (something we will only see later on in our contributions), we manage to state even more complex properties ,such as *never, until, eventually, and always*, that are very relevant in formal verification.

Definition 2.6. *A Kripke model $\mathcal{M}$ is a triple (W, R, V), where $R \subseteq W \times W$, and V : $\text{PROP} \rightarrow 2^W$ is a valuation of propositional variables, such that for every p, $V(p) \subseteq W$.*

Formally, the truth ($\models$) of modal logic formulas is evaluated in a world w of a model $\mathcal{M}$, through the rules given in Figure 2.3.

A formula $\varphi \in M_L$ is called *satisfiable*, if there exists a model $\mathcal{M}$ such that $\mathcal{M}, w \models \varphi$ holds in some world w of $\mathcal{M}$. It is called *valid*, if it is satisfied in all worlds of all models (the equivalent of a tautology), and it is called valid for model $\mathcal{M}$, if it is satisfied in all worlds of $\mathcal{M}$.

The notions of satisfiability and validity for formulas of a given logic have had a central role in the field of logic in computer science, where they have been associated to a variety of problems such as the model-checking and the synthesis problems. We will see a variety of such results, both existing and new, later on, in Chapter 7.

2.5.3 Connections to Process Algebra

Modal logic, even though it has long been established as useful to formal verification, is not the logic that is seen as the most proximate to process algebras. That is Hennessy-Milner Logic (HML) [125], introduced in 1980 to express properties of labeled transition systems, which are the core semantic models of process algebras.

The syntax of HML is very similar to that of modal logic, in the sense that it also employs box and diamond modalities to semantically refer to an accessibility relation. The main difference here is that HML is a multi- modal logic, and its modalities are parameterized by some action a (i.e. $\langle a \rangle$), and therefore they directly relate to the labels of the edges connecting the nodes in an LTS model, very similarly to multi-agent modal logic. In the case of multi-agent modal logic the labels are interpreted as agents while in HML the labels correspond to computational actions. This distinction does give different flavors to the different approaches with each one, but the rest of their structure remains essentially the same. This relatively straightforward syntactic and semantic correspondence enables the transfer of results between the two fields.

Possibly the most prominent result from the study of HML is its connection to bisimilarity, which is expressed by the following theorem:

Theorem 2.1 (Hennessy-Milner Theorem, from [125])**.** *Let s, s' be states in an image-finite*[4] *LTS L, with $L = (Proc, \textsc{Act}, \{\xrightarrow{a} \mid a \in Act\})$. Then $s \sim s'$, iff P and Q satisfy the exact same formulas in HML.*

Theorem 2.1 is an example of a *modal characterization theorem*, such as the one for simple modal logic [59, Theorems 2.20, 2.24]. In the case of HML and multi-agent modal logic, such results are expected to hold either for both of for none of the two logics, due to their structural similarities. Specifically, even though we will not present the full syntax and semantics of Hennessy-Milner logic it suffices to mention that its syntax is almost identical to the one of multi-agent modal logic. Thus, results like the above, or the ones we will present later, are carried over between them, by simply translating them in the relevant syntax.

Later on, in Chapter 7 we study the full μ-Calculus, which is an extension of modal logic through the use of recursive operators. Its semantics are given over Kripke structures, just as before, and thus, all results we prove are transferable to an equivalent setting over a relevant extension of HML with recursion operators. However, in this study we focused on certain details of the μ-calculus that are associated to its epistemic interpretation, and have not traditionally been directly associated to the temporal meaning of recursive HML. Lately though combinations of the two interpretations have found application, for example in systems that contain both computational and epistemic aspects [43].

[4] An image-finite LTS L is one for which the set $\{t \mid s \xrightarrow{a} t\}$ is finite, for each action a and each state s.

Part II

Contributions

Chapter 3

Axiomatizability of BPA$_\Theta$

3.1 Introduction

A fundamental feature that has been implemented within the process algebra framework is the possibility to express that some actions have *priority* over others (we refer the interested reader to [77] for an overview of the proposals). This allows for modelling, for example, that an interrupt or shutdown action may be needed when a system deadlocks or starts exhibiting erroneous behavior, and, likewise, that a scheduler needs to assign a different level of urgency to actions based on its scheduling policy. Here we consider the approach taken in [40], where a priority operator Θ is introduced. This operator is based on an irreflexive partial order, called the *priority order*, over the actions that are available to the process, and only allows an action to be performed if no other action with a higher priority is possible at the given moment.

In the literature we can find a variety of results on the equational theory of the priority operator Θ in different settings, as we review below. With this chapter, we give our contribution to these studies by discussing the equational axiomatization for a process algebra having both Θ and the *sequential composition operator* of BPA [55], modulo a notion of bisimulation equivalence, called *order-insensitive bisimilarity* [13], that holds *irrespectively* of the chosen priority order over actions.

3.1.1 On the Axiomatizability of Priority

Earlier studies on the axiomatizability of the priority operator were carried out with respect to a chosen, arbitrary, priority order. In the seminal papers [40, 54] it was shown that, provided that the set of actions is finite, the priority operator admits a *finite*, *ground-complete* equational axiomatization. (A set of axioms is called *ground-complete* if every sound equation between process terms without variables can be derived from those axioms using the rules of equational logic.) For an infinite set of actions, it was proved in [8] that the operator Θ admits no finite equational axiomatization over the process algebra BCCSP$_\Theta$, which consists of basic operators from CCS [152] and CSP [129], enriched with Θ. Furthermore, a specific priority order was exhibited for which no finite equational ground-complete axiomatization exists.

Later, in [13], the first study of an equational axiomatization of an equivalence that is irrespective of the chosen priority order was provided. More precisely, it considers the notion of *order-insensitive bisimilarity*, denoted by $\underline{\leftrightarrow}_*$, over processes in BCCSP$_\Theta$: two processes are $\underline{\leftrightarrow}_*$-equivalent if they are bisimilar under every priority order. Now, one may expect that if we consider order-insensitive bisimilarity then there are no sound equations of interest that involve the priority operator. However, as shown in [13], this is not the case. If the set of actions contains at least two distinct elements, then there is no finite, ground-complete equational axiomatization modulo order-insensitive bisimilarity. To prove their negative result, the authors of [13] showed that no finite set of equations valid modulo $\underline{\leftrightarrow}_*$ can prove all of the equations in the following infinite family

$$a^n.(b+c)+a^n.b+a^n.c \;\approx\; a^n.(b+c)+a^n.b+a^n.c+a^n.\Theta(b+c) \qquad (n \geq 0)\ . \quad \text{(E)}$$

However, they also remarked that if we replace BCCSP's action prefixing with BPA's *sequential composition* operator, then all the equations in (E) could be replaced by the following valid equation

$$x \cdot (b+c) + x \cdot b + x \cdot c \;\approx\; x \cdot (b+c) + x \cdot b + x \cdot c + x \cdot \Theta(b+c)\ .$$

This observation left the following open problem:

> *Is order-insensitive bisimilarity finitely axiomatisable over the process algebra* BPA$_\Theta$, *namely* BPA *enriched with the priority operator?* (P)

In this chapter, we provide a *negative* answer to this question.

3.1.2 Our Contribution

Our main result consists in proving that, provided there are at least two distinct actions, the priority operator admits no finite, ground-complete equational axiomatization modulo order-insensitive bisimilarity over the process algebra BPA$_\Theta$.

The first issue we need to overcome is that, differently from classical bisimulations, order-insensitive bisimilarity is not coinductive: the derivatives of two order-insensitive bisimilar processes cannot be, in general, paired-up in order-insensitive bisimilarity equivalence classes. Hence, we will first of all identify a class of processes on which order-insensitive bisimilarity always behaves coinductively (Proposition 3.3).

Then, to prove our negative result we use proof-theoretic techniques that have their roots in Moller's classic results to the effect that bisimilarity is not finitely based over CCS (see, e.g., [6, 157, 158, 160]). Roughly speaking, we will identify a special property of processes, called the (n, Θ)-*dependency* property, associated with each finite set E of sound axioms and a natural number n. Informally, a process satisfies (n, Θ)-dependency if by performing a trace of length n it reaches a process whose behavior depends on the considered priority order, and is thus determined by the priority operator. Moreover, we require that, at each step, the process has the possibility of terminating. The idea is that, when n is *large enough*, whenever an equation $p \approx q$ is derivable from E, then either both terms p and q satisfy (n, Θ)-dependency, or none of them does. The negative result is then obtained by exhibiting an infinite family of valid equations $\{e_n \mid n \geq 0\}$ in which the (n, Θ)-dependency property is not preserved, that is, for each $n \geq 0$, only one side of e_n satisfies (n, Θ)-dependency. Due to the choice of the special property, this means that the equations in the family cannot all be derived from a finite set of valid axioms and therefore no finite, sound axiom system can be complete (Theorem 3.1). We remark that the requirement on the possibility of termination after each step will ensure that the processes on both sides of the equations e_n cannot be written as a sequential composition, thus preventing the replacement of the infinite family with a finite number of equations that occurred in the case of the equations in (E).

In the axiom system ACP_Θ the axioms for the priority operator made use of an auxiliary operator, called the *unless* operator. It is then natural to wonder whether by adding also the unless operator to the syntax of BPA_Θ it would be possible to obtain a finitely based axiomatization of order-insensitive bisimilarity. We show that also in this case the answer is negative (Theorem 3.4).

Finally we study the complexity of the order-insensitive bisimilarity checking. As two processes are order-insensitive bisimilar if and only if they are bisimilar under all possible priority orders, the simplest algorithm for order-insensitive bisimilarity would consists in checking all of them. Our main contribution to this problem is not in the cost of a bisimilarity check, which can be done in $O(m_t \log m_s)$, where m_t is the number of transitions and m_s the number of states [163], but it consists in showing that we actually need to do the check for all possible priority orders. In fact, we prove that for each priority order there exists at least a pair of processes that are bisimilar with respect to all priority orders with the sole exception of the chosen one (Theorem 3.6). Following [138], there are $2^{k^2/4+3k/4+O(\log k)}$ partial orders over a set of k actions. Hence, we show that the problem of deciding whether two processes are order-insensitive bisimilar is in coNP and can be solved in time $2^{k^2/4+3k/4+O(\log k)} \cdot O(n^2)$, where n is the sum of the sizes of the two processes (Theorem 3.5).

3.1.3 Outline of the Chapter

We start by reviewing background notions in Section 3.2. Section 3.3 gives an informal presentation of our proof strategy, whose technical development is provided in Sections 3.4–3.7. In detail: Section 3.4 comes with technical results necessary to reason on the semantics of open process terms. In Section 3.5 we provide the properties necessary to ensure that order-insensitive bisimilarity behaves coinductively. In Section 3.6 we present the (n, Θ)-dependency property of processes necessary to prove our negative result. Our main result is in Section 3.7 where we prove that the order-insensitive bisimilarity is not finitely based over BPA with the priority operator. In Section 3.8 we briefly argue that the negative result would still hold even if we enrich the syntax of BPA_Θ with the auxiliary operator unless. Then, we devote Section 3.9 to discussing the complexity of order-insensitive bisimilarity checking.

Finally, we draw some conclusions and discuss future work in Section 3.10.

3.1.4 What's New

A preliminary version of this work appeared as [16]. Besides providing the full proofs of our results and new examples, we have enriched our previous contribution as follows:

a. We discuss the general reasoning behind the proof of our main result (Theorem 3.1) and present our proof strategy at an informal level, thus providing a guide for the reader through the technical development of our result (Section 3.3).

b. We discuss the possibility of using auxiliary operators to axiomatize the priority operator Θ and thus regaining a finite ground-complete axiomatization over the enriched language BPA$_\Theta$, modulo bisimilarity. We argue that due to some features of order-insensitive bisimilarity, this is not the case (Section 3.8).

c. We discuss the complexity of order-insensitive bisimilarity check and we show that it is indeed necessary to always check for bisimilarity with respect to all priority orders (Section 3.9).

3.2 Background

In this section we review some preliminary notions on operational semantics and equational logic. Since our work naturally builds on [13, 9] we will use the notation from those papers as much as possible.

3.2.1 BPA$_\Theta$: Syntax and Semantics

The syntax of *process terms* in BPA$_\Theta$, namely BPA [55] enriched with the priority operator [40], is generated by the following grammar

$$t ::= a \mid x \mid t \cdot t \mid t + t \mid \Theta(t) \ ,$$

with a ranging over a set of actions $\textsc{Act}$, x ranging over a countably infinite set of variables $\mathcal{V}$ and t ranging over process terms. We write $\mathrm{var}(t)$ for the set of variables occurring in t. A process term is *closed* if no variable occurs in it. We shall, sometimes, refer to closed process terms simply as *processes*. We let $\mathbf{P}$ denote the set of BPA$_\Theta$ processes and let $p, q, \ldots$ range over it.

We use the *Structural Operational Semantics* (SOS) framework [168] to equip processes with a semantics. A *literal*, or *open transition*, is an expression of the form $t \xrightarrow{a} t'$ for some process terms t, t' and action $a \in \textsc{Act}$. It is *closed* if both t, t' are closed process terms.

The inference rules for *sequential composition* $\cdot$, alternative *nondeterministic choice* $+$ and *priority* Θ are reported in Table 3.1. We remark that the semantics of Θ is based on a strict irreflexive partial order $>$ on $\textsc{Act}$, called the *priority order*,

$$(r_1)\ \frac{}{a \xrightarrow{a}_{>} \checkmark} \qquad (r_2)\ \frac{p \xrightarrow{a}_{>} \checkmark}{p \cdot q \xrightarrow{a}_{>} q} \qquad (r_3)\ \frac{p \xrightarrow{a}_{>} p'}{p \cdot q \xrightarrow{a}_{>} p' \cdot q}$$

$$(r_4)\ \frac{p \xrightarrow{a}_{>} \checkmark}{p + q \xrightarrow{a}_{>} \checkmark} \qquad (r_5)\ \frac{q \xrightarrow{a}_{>} \checkmark}{p + q \xrightarrow{a}_{>} \checkmark} \qquad (r_6)\ \frac{p \xrightarrow{a}_{>} p'}{p + q \xrightarrow{a}_{>} p'} \qquad (r_7)\ \frac{q \xrightarrow{a}_{>} q'}{p + q \xrightarrow{a}_{>} q'}$$

$$(r_8)\ \frac{p \xrightarrow{a}_{>} \checkmark \quad \forall\, b > a \,.\, p \not\xrightarrow{b}_{>}}{\Theta(p) \xrightarrow{a}_{>} \checkmark} \qquad (r_9)\ \frac{p \xrightarrow{a}_{>} p' \quad \forall\, b > a \,.\, p \not\xrightarrow{b}_{>}}{\Theta(p) \xrightarrow{a}_{>} \Theta(p')}$$

Table 3.1: Operational semantics of processes in BPA_{Θ}.

which justifies the parametrization of the derived transition relation with respect to $>$. For simplicity, given $a, b \in \textsc{Act}$, we write $a > b$ for $(a, b) \in >$. To deal with sequential composition in the absence of deadlock and empty process (see, e.g., [202, 55]), we introduce the *termination predicate* $\rightarrow_{>}\checkmark \subseteq \mathbf{P} \times \textsc{Act}$. Intuitively, $t \xrightarrow{a}_{>}\checkmark$ means that t can terminate successfully in one step by performing action a.

A *substitution* σ is a mapping from variables to process terms. It extends to process terms, literals and rules in the usual way and it is *closed* if it maps every variable to a process. We denote by $\sigma[x \mapsto u]$ the substitution that maps each occurrence of the variable x into the process term u and behaves like σ over all other variables.

In [4] it was shown that we can define a stratification [63, 110] on the set of BPA_{Θ} rules by counting the number of occurrences of the priority operator in the left-hand side of a transition. Hence, the inference rules in Table 3.1 induce a unique supported model [196, 4] corresponding to the $\textsc{Act}$-labeled transition system $(\mathbf{P}, \textsc{Act}, \rightarrow_{>}, \rightarrow_{>}\checkmark)$ whose transition relation $\rightarrow_{>}$ (respectively, predicate $\rightarrow_{>}\checkmark$) contains exactly the closed literals (respectively, predicates) that can be derived by structural induction over processes using the rules in Table 3.1.

As usual, we write $p \xrightarrow{a}_{>} p'$ for $(p, a, p') \in \rightarrow_{>}$, $p \rightarrow_{>} p'$ if $p \xrightarrow{a}_{>} p'$ for some $a \in \textsc{Act}$, and $p \not\xrightarrow{a}_{>}$ if there is no p' such that $p \xrightarrow{a}_{>} p'$. For $k \in \mathbb{N}$, we write $p \rightarrow_{>}^{k} p'$ if there are $p_0, \ldots, p_k$ such that $p = p_0 \rightarrow_{>} \cdots \rightarrow_{>} p_k = p'$. Furthermore, for a sequence of actions $s = a_1 \ldots a_n$, we write $p \xrightarrow{s}_{>} p'$ to mean that $p \xrightarrow{a_1}_{>} p_1 \xrightarrow{a_2}_{>} \cdots p_{n-1} \xrightarrow{a_n}_{>} p'$ for some processes $p_1, \ldots, p_{n-1}$.

We associate two classic notions with each process: its *depth* and its *norm*. As usual, they express, respectively, the length of a *longest* and a *shortest* sequence of transitions that are enabled for the process. Since in our setting the length of sequences of enabled transitions depends on the considered priority order, we define the depth and the norm of a process with respect to the empty order. The reason for this choice is twofold. Firstly, we notice that the depth defined with respect to the empty order is an upper bound for the depths defined with respect to any other priority order. Since for our purposes we will need to consider upper bounds for the depth of processes, and not the exact value of their depths, it is reasonable to consider directly the greatest of the depths. Notice that the norm defined with respect to the empty order is, dually, a lower bound for the norms defined with respect to the other priority orders. Secondly, this choice allows us to give alternative formulations of both notions by induction on the structure of processes.

Definition 3.1 (Depth and norm). *The* depth *of a process is defined inductively on its structure by*

- $\text{depth}\,(a) = 1$;
- $\text{depth}\,(p_1 \cdot p_2) = \text{depth}\,(p_1) + \text{depth}\,(p_2)$;
- $\text{depth}\,(p_1 + p_2) = \max\{\text{depth}\,(p_1)\,, \text{depth}\,(p_2)\}$;
- $\text{depth}\,(\Theta(p)) = \text{depth}\,(p)$.

Similarly, the norm *of process is defined inductively on its structure by*

- $\text{norm}\,(a) = 1$;
- $\text{norm}\,(p_1 \cdot p_2) = \text{norm}\,(p_1) + \text{norm}\,(p_2)$;
- $\text{norm}\,(p_1 + p_2) = \min\{\text{norm}\,(p_1)\,, \text{norm}\,(p_2)\}$;
- $\text{norm}\,(\Theta(p)) = \text{norm}\,(p)$.

Both notions can be extended to process terms *by adding, respectively, the value of the depth and norm of a variable which are defined as* $\text{depth}\,(x) = 1$ *and* $\text{norm}\,(x) = 1$.

We remark that although variables cannot perform any transition, as one can easily see from the inference rules in Table 3.1, their depth, and norm, are set to 1, since the minimal closed instance of a variable with respect to these measures is as a constant in Act.

For $p \in \mathbf{P}$, the set of *initial actions* of p with respect to $>$ is defined as

$$\text{init}_>(p) = \{a \mid p \xrightarrow{a}_> p', p' \in \mathbf{P}\} \cup \{a \mid p \xrightarrow{a}_> \surd\!\!\!/\}.$$

We extend this notion to sequences of transitions as $\text{init}^k_>(p) = \bigcup_{p \to^k_> p'} \text{init}_>(p')$ and $\text{init}^\omega_>(p) = \bigcup_{k \in \mathbb{N}} \text{init}^k_>(p)$ be, respectively, the set of actions that are enabled with respect to $>$ at depth k and at some depth. We say that action a is *maximal* with respect to $>$ if there is no $b \in \text{Act}$ such that $b > a$. We can restrict this notion to the set of actions that are enabled for a process. Given a process p, we say that an action $a \in \text{init}^\omega_>(p)$ is *maximal in* p, or *locally maximal*, with respect to $>$ if there is no $b \in \text{init}^\omega_>(p)$ such that $b > a$. If $\text{init}^\omega_>(p) = \{a\}$ then a is locally maximal with respect to $>$.

3.2.2 Order-Insensitive Bisimulation

With the priority operator, the set of transitions that are enabled for each process depends on the considered priority order on $\textsc{Act}$. Therefore, any bisimulation relation over BPA_Θ processes will also depend on the priority order. In [13], along all such bisimulations, the authors introduced the notion of *order-insensitive bisimilarity*, $\underline{\leftrightarrow}_*$, formally defined as the intersection over all priority orders of the related bisimulation relations. Since $\underline{\leftrightarrow}_*$ disregards the particular order that is considered, it can be used to study general properties of processes and thus develop a general equational theory for BPA_Θ.

Definition 3.2 (Order-insensitive bisimulation, [13]). *Let $>$ be any priority order. A binary symmetric relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a* bisimulation with respect to $>$ *if whenever $p \mathcal{R} q$ then*

- *for all $p \xrightarrow{a}_> p'$ there is $q \xrightarrow{a}_> q'$ such that $p' \mathcal{R} q'$, and*
- *for all $p \xrightarrow{a}_> \surd\!\!\!\surd$ also $q \xrightarrow{a}_> \surd\!\!\!\surd$ holds.*

We say that p, q are bisimilar with respect to $>$, *denoted by $p \,\underline{\leftrightarrow}_>\, q$, if $p \mathcal{R} q$ holds for some bisimulation $\mathcal{R}$ with respect to $>$.*

We say that p, q are order-insensitive bisimilar, *denoted by $p \,\underline{\leftrightarrow}_*\, q$, if $p \,\underline{\leftrightarrow}_>\, q$ holds for all priority orders.*

For a given priority order $>$, the bisimulation equivalence $\underline{\leftrightarrow}_>$ behaves like a classic bisimulation and therefore the following lemma, from [13], holds.

Lemma 3.1 ([13, Proposition 9]). *Consider processes p, q, assume $p \underline{\leftrightarrow}_> q$ for some priority order $>$ over* $\textsc{Act}$, *and let $k \in \mathbb{N}$. Then:*

1. *For every process p' such that $p \rightarrow^k_> p'$, there is a process q' such that $q \rightarrow^k_> q'$ and $p' \,\underline{\leftrightarrow}_>\, q'$.*
2. $\text{init}^k_>(p) = \text{init}^k_>(q)$ *so, in particular,* $\text{init}^1_>(p) = \text{init}^1_>(q)$.

It is not hard to prove that, since the inference rules in Table 3.1 respect the GSOS format [62], $\underline{\leftrightarrow}_>$ and $\underline{\leftrightarrow}_*$ are congruences over BPA_Θ processes. However, as discussed in [13], $\underline{\leftrightarrow}_*$ does not inherit the coinductive nature of bisimilarity, as we show in the following example.

***Example* 1.** *Consider the processes $p = a \cdot b + a \cdot c + a \cdot (b + c)$ and $q = p + a \cdot \Theta(b + c)$. Notice that*

- *if $b > c$ then $a \cdot \Theta(b + c) \,\underline{\leftrightarrow}_>\, a \cdot b$,*
- *if $c > b$ then $a \cdot \Theta(b + c) \,\underline{\leftrightarrow}_>\, a \cdot c$, and*
- *if b, c are incomparable with respect to $>$ then $a \cdot \Theta(b + c) \,\underline{\leftrightarrow}_>\, a \cdot (b + c)$.*

Therefore, we have that $p \,\underline{\leftrightarrow}_*\, q$. However, $q \xrightarrow{a}_> \Theta(b + c)$ for each order $>$, but there is no p' such that $p \xrightarrow{a}_> p'$ and $p' \,\underline{\leftrightarrow}_*\, \Theta(b + c)$.

For sake of notation, henceforth, whenever $>$ is the empty order, we simply omit the subscript, i.e., $\rightarrow_\emptyset, \underline{\leftrightarrow}_\emptyset$ and $\text{init}_\emptyset(\cdot)$ become, respectively, $\rightarrow, \underline{\leftrightarrow}$ and $\text{init}(\cdot)$.

$$(e_1)\,\frac{}{t \approx t} \qquad (e_2)\,\frac{t \approx u}{u \approx t} \qquad (e_3)\,\frac{t \approx u \quad u \approx v}{t \approx v} \qquad (e_4)\,\frac{t \approx u}{\sigma(t) \approx \sigma(u)}$$

$$(e_5)\,\frac{t_1 \approx u_1 \quad t_2 \approx u_2}{t_1 \cdot t_2 \approx u_1 \cdot u_2} \qquad (e_6)\,\frac{t_1 \approx u_1 \quad t_2 \approx u_2}{t_1 + t_2 \approx u_1 + u_2} \qquad (e_7)\,\frac{t \approx u}{\Theta(t) \approx \Theta(u)}$$

Table 3.2: Rules of equational logic over BPA$_\Theta$.

3.2.3 Equational Logic, in the Context of BPA$_\Theta$

An *axiom system* E is a collection of *process equations* $t \approx u$ over the language BPA$_\Theta$, such as those presented in Table 3.3. An equation $t \approx u$ is *derivable* from an axiom system E, notation $\mathsf{E} \vdash t \approx u$, if there is an *equational proof* for it from E, namely if it can be inferred from the axioms in E using the *rules* of *equational logic*, which are reflexivity, symmetry, transitivity, substitution and closure under BPA$_\Theta$ contexts, and are reported in Table 3.2.

Let E be a sound set of axioms. Rules (e_1)-(e_4) are common for all process languages and they ensure that E is closed with respect to reflexivity, symmetry, transitivity and substitution, respectively. Rules (e_5)-(e_7) are tailored for BPA$_\Theta$ and they ensure the closure of E under BPA$_\Theta$ contexts. They are therefore referred to as the *congruence rules*. Briefly, rule (e_5) is the rule for sequential composition and it states that whenever $\mathsf{E} \vdash t_1 \approx u_1$ and $\mathsf{E} \vdash t_2 \approx u_2$, then we can infer $\mathsf{E} \vdash t_1 \cdot u_1 \approx t_2 \cdot u_2$. Rule (e_6) deals with the nondeterministic choice operator in a similar way and rule (e_7) ensures that the priority operator preserves the equivalence of terms.

As elsewhere in the literature, we assume, without loss of generality, that for each axiom in E also the symmetric counterpart is in E, so that the symmetry rule is not necessary in the proofs, and that substitution rules are always applied first in equational proofs, which means that the substitution rule $\frac{t \approx u}{\sigma(t) \approx \sigma(u)}$ may only be used for axioms $t \approx u$ in E. If this is the case, then $\sigma(t) \approx \sigma(u)$ is called a *substitution instance* of the axiom.

The process equation $t \approx u$ is said to be *sound* with respect to $\underline{\leftrightarrow}_*$ if $\sigma(t)\underline{\leftrightarrow}_*\sigma(u)$ for all closed substitutions σ. For simplicity, if $t \approx u$ is sound, then we write $t \,\underline{\leftrightarrow}_*\, u$. An axiom system is *sound* modulo $\underline{\leftrightarrow}_*$ if and only if all of its equations are sound modulo $\underline{\leftrightarrow}_*$. Conversely, we say that E is *ground-complete* modulo $\underline{\leftrightarrow}_*$ if $p \,\underline{\leftrightarrow}_*\, q$ implies $\mathsf{E} \vdash p \approx q$ for all processes p, q. We say that $\underline{\leftrightarrow}_*$ is *finitely based*, if there is a *finite* axiom system E such that $\mathsf{E} \vdash t \approx u$ if and only if $t \,\underline{\leftrightarrow}_*\, u$. Finally, notice that the notion of depth can be extended to equations by letting $\mathrm{depth}\,(t \approx u) = \max\{\mathrm{depth}\,(t)\,, \mathrm{depth}\,(u)\}$.

3.3 Towards a Negative Result

As disclosed in the Introduction, our order of business for the remainder of this chapter will be to prove the following theorem:

C1	$x + y \approx y + x$	S1	$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$
C2	$(x + y) + z \approx x + (y + z)$	S2	$(x + y) \cdot z \approx (x \cdot z) + (y \cdot z)$
C3	$x + x \approx x$		
P1	$\Theta(\Theta(x) + y) \approx \Theta(x + y)$		
P2	$\Theta(x) + \Theta(y) \approx \Theta(x) + \Theta(y) + \Theta(x + y)$		
P3	$\Theta(x \cdot y) \approx \Theta(x) \cdot \Theta(y)$		
P4	$\Theta(x \cdot y + x \cdot z + w) \approx \Theta(x \cdot y + w) + \Theta(x \cdot z + w)$		
P5	$\Theta(a) \approx a$		

Table 3.3: Some axioms of BPA_Θ.

Theorem 3.1. *If the set of actions* Act *contains at least two distinct actions, then the language* BPA_Θ *modulo order-insensitive bisimilarity is not finitely based.*

Due to the heavy amount of technical results that are needed to fulfill this purpose, we decided to dedicate this section to an informal description of our proof strategy. Hopefully, this will improve the readability of our chapter and work as a guide for the reader in their journey through the technical development of our results.

3.3.1 The Idea

Our method stems from [157, 158, 160], in which Moller discussed the axiomatiazability of the parallel composition operator and proved that (a fragment of) CCS modulo bisimilarity is not finitely based. The key idea is to identify a *special property* of BPA_Θ terms, say $\mathbb{P}(n)$ for $n \geq 0$, that, when n is *large enough*, is preserved by provability under finite axiom systems. Roughly, this means that if E is a finite set of axioms that are sound modulo order-insensitive bisimilarity, the equation $p \approx q$ is provable from E, and n is greater than the depth of the equations in E, then either both p and q satisfy $\mathbb{P}(n)$, or none of them does. Then we introduce a family of infinitely many equations $\{e_n \mid n \geq 0\}$ that are all sound modulo $\underline{\leftrightarrow}_*$, but are such that only one side of e_n satisfies $\mathbb{P}(n)$, for each $n \geq 0$. This implies that the family of equations cannot be derived from any finite axiom system that is sound modulo $\underline{\leftrightarrow}_*$ and, hence, at least infinitely many of those equations must be included in the axiomatization, which is therefore *not* finitely based.

3.3.2 The Choice of $\mathbb{P}(n)$

The property $\mathbb{P}(n)$ will involve the priority operator. We shall say, in a very informal way, that $\mathbb{P}(n)$ will be satisfied by a process p if it reaches, through a sequence of n *steps*, a process, say p', whose behavior is *determined by* Θ. Intuitively, this means that p' behaves differently under different priority orders. For instance, p' could be of the form $\Theta(\Theta(\Theta(a) + b \cdot p''))$ for some $a \neq b$ and process p''. Then p' affords an a-transition and no b-transition if $a > b$, whereas p' affords a b-transition and no a-transition if $b > a$. It is important that Act contains at least two actions, so that we can have different priority orders (possibly) triggering different behaviors of Θ-terms. Moreover, p' must have (a nesting of) Θ as head operator and a nondeterministic choice between (at least) two processes having distinct sets of initial actions must occur within the scope of such (nesting of) Θ.

Borrowing the terminology from [13], we will call Θ-*dependent* the process terms whose initial behavior depends on the priority order. The choice of involving Θ-dependent terms in $\mathbb{P}(n)$ is strongly related to the fact that we are considering order-insensitive bisimilarity. In fact, as we need to take into account the behavior of processes with respect to *all* priority orders, then no axiom can be used to eliminate the head occurrence of Θ from Θ-dependent terms. These terms and their properties will be presented in Section 3.6.

There is, however, another feature of order-insensitive bisimilarity that we will need to take into account to properly define the property $\mathbb{P}(n)$. As previously outlined, differently from classic notions of bisimulations, $\underline{\leftrightarrow}_*$ does not have, in general, a coinductive construction. Hence, to simplify the reasoning in the proofs, we need to define $\mathbb{P}(n)$ in such a way that only those processes on which $\underline{\leftrightarrow}_*$ can be defined coinductively could satisfy it. To this end we introduce, in Section 3.5, the notion of *uniform determinacy* as a sufficient condition to ensure the coinductive behavior of $\underline{\leftrightarrow}_*$.

The special property $\mathbb{P}(n)$ is then defined, in Section 3.6, as the property of *uniform* (n, Θ)-*dependency* of processes, which combines the ideas of determinacy and Θ-dependency of processes and, in addition, will require that all the processes in the sequence of n steps leading to the Θ-dependent term have norm 1. This is to guarantee that no axiom for sequential composition can be used to rewrite such a sequence.

3.3.3 The Choice of n

The choice of n *large enough* will play a fundamental role in proving that whenever p satisfies $\mathbb{P}(n)$ then so does q, especially in the case in which $p \approx q$ is derived by an application of the substitution rule of equational logic (rule (e_4) in Table 3.2). In this case, we have $p = \sigma(t)$ and $q = \sigma(u)$ for some closed substitution σ and BPA$_\Theta$ terms t, u such that $t \approx u \in \mathsf{E}$. Then, if n is *large enough*, which translates into n being greater than the depth of the equations in E (and thus of the depth of all the terms occurring in such equations), we can prove that the fact that p satisfies $\mathbb{P}(n)$ is due to the behavior of the closed instance of some variable x occurring in t. We can also prove that for $t \approx u$ to be sound modulo $\underline{\leftrightarrow}_*$, whenever a variable x occurs in t then it must also occur in u. Actually, we are going to prove the stronger result that if such an occurrence of x in t is within the scope a priority operator, then so is the occurrence of x in u. Hence, we can infer that $\sigma(x)$ will trigger in $\sigma(u)$ the same behavior that it induced in $\sigma(t)$, and thus that also $q = \sigma(u)$ will satisfy $\mathbb{P}(n)$.

To obtain all the results mentioned in this subsection it will be fundamental to study the decomposition of the behavior of closed instances of terms with respect to the behavior of the closed instances of variables occurring in them. Section 3.4 is devoted to such an analysis.

3.3.4 The Family of Equations

Consider the processes $\{P_n\}_{n\in\mathbb{N}}$, defined as follows

$$\begin{array}{rll} & P_n = A_n(a) + A_n(b) + A_n(a+b) & (n \geq 0) \\ \text{where} & A_0(p) = p & \\ \text{and} & A_{n+1}(p) = a \cdot A_n(p) + a & (n \geq 0)\ . \end{array}$$

Intuitively, the process P_n must at the top level decide whether it will end up in a, b, or $a+b$ after n steps. After making this choice, it can take up to n a-transitions, and at each step it can choose whether to terminate or to continue. The possibility of termination at each step is crucial, since it means that the process cannot be written just with sequential composition modulo bisimilarity.

As we will formally prove in Section 3.7, the following family of infinitely many sound equations shows that order-insensitive bisimilarity is not finitely based over BPA$_\Theta$

$$e_n : \quad P_n + A_n(\Theta(a+b)) \approx P_n \qquad (n \geq 0)\ . \tag{3.1}$$

Informally, each equation e_n is sound, because, according to which priority order is considered, $\Theta(a+b)$ will be bisimilar to a, b or $a+b$, and thus the two sides of e_n are order-insensitive bisimilar. However, process $A_n(\Theta(a+b))$ can be proved to be *uniformly* (n,Θ)*-dependent*, whereas P_n is not. We will argue that this implies that not all the equations in the family $\{e_n\}_{n\in\mathbb{N}}$ can be derived from a finite set of valid axioms, thus proving Theorem 3.1.

3.4 Relation Between Open and Closed Operational Behavior

Our purpose in the remainder of this chapter is to verify whether the axiomatization for order-insensitive bisimilarity is finitely based over BPA$_\Theta$. To address this question it is fundamental to establish a correspondence between the behavior of open terms and the semantics of their closed instances, with a special focus on the role of variables. In this section, we provide the notions and theoretical results necessary to establish the desired behavioral correspondence.

3.4.1 From Open to Closed Transitions...

Assume a term t, a closed substitution σ, a process p, an action a and a priority order $>$. We aim at investigating how to derive a transition of the form $\sigma(t) \xrightarrow{a}_{>} p$, as well as a predicate $\sigma(t) \xrightarrow{a}_{>} \surd\!\!\!\surd$, from the behavior of t and of $\sigma(x)$ for each variable x occurring in t. In particular we are interested in relating the *initial* behavior of $\sigma(t)$ with the behavior of closed instances of variables occurring in it.

The simplest case is a direct application of the operational semantics in Table 3.1: if action a is maximal with respect to $>$, then $\sigma(t) \xrightarrow{a}_{>} p$ can be inferred directly from $t \xrightarrow{a}_{>} t'$, for some term t' with $\sigma(t') = p$. In fact, the maximality of a guarantees that the execution of the a-transition cannot be prevented by any occurrence of the priority operator. A similar reasoning holds for transition predicates.

Lemma 3.2. *Let t, t' be process terms, let a be an action with maximal priority with respect to $>$. Then for all substitutions σ it holds that:*

1. *If $t \xrightarrow{a}_{>} \surd\!\!\surd$ then $\sigma(t) \xrightarrow{a}_{>} \surd\!\!\surd$.*

2. *If $t \xrightarrow{a}_{>} t'$ then $\sigma(t) \xrightarrow{a}_{>} \sigma(t')$.*

Next we deal with variables. It may be the case, for instance, that the term t is of the form $t = x \cdot u$ for some term u. Clearly, the behavior of $\sigma(t)$, and thus the derivation of $\sigma(t) \xrightarrow{a}_{>} p$, will depend on the behavior of $\sigma(x)$. However, the set of initial actions of $\sigma(t)$ does not depend, in general, solely on those of $\sigma(x)$, but also on the structure of the process into which x is mapped, and on the occurrence of x in t. For instance, for $t = x \cdot u$ we can distinguish two main situations:

(I) Suppose $\sigma(x) = a$, so that $\sigma(x) \xrightarrow{a}_{>} \surd\!\!\surd$. This would give $\sigma(t) \xrightarrow{a}_{>} p$ for $p = \sigma(u)$, namely p is a closed instance of a subterm of t. Therefore, the transition for $\sigma(t)$ could be expressed in terms of a closed instance of an open transition for t, as $t \rightarrow_{>} u$. However, notice that the action that is performed cannot be obtained from the term t as it depends solely on the substitution applied to x. Hence, we will need a formal way to express that the label of the transition depends on x.

(II) Suppose $\sigma(x) = a \cdot b$, so that $\sigma(x) \xrightarrow{a}_{>} b$. Clearly, $\sigma(t)$ will have to mimic such behavior, and thus $\sigma(t) \xrightarrow{a}_{>} p$ with $p = b \cdot \sigma(u)$. Notice that process p subsumes *what's left* of the behavior of $\sigma(x)$. Then the transition for $\sigma(t)$ cannot be inferred from a closed substitution instance of an open transition of the form $t \xrightarrow{a}_{>} t'$, since the structure of t' cannot be known until the substitution $\sigma(x)$ has occurred. Hence, we will need a formal way to express that to reach a subterm of t we need to follow a sequence of transitions performed by x.

For a formal development of the analysis in the above-mentioned cases, we exploit the method proposed in [9] and provide an auxiliary operational semantics tailored for expressing the behavior of process terms resulting from that of closed substitution instances for their variables.

Firstly we introduce the notion of *configuration* over BPA$_\Theta$ terms, which stems from [9]. Configurations are terms defined over a set of variables $\mathcal{V}_\mathrm{d} = \{x_d \mid x \in \mathcal{V}\}$, disjoint from $\mathcal{V}$, and BPA$_\Theta$ terms. We use the variable x_d to express that the closed instance of x has started its execution, but has not terminated yet.

Definition 3.3 (BPA$_\Theta$ configuration). *The collection of* BPA$_\Theta$ *configurations is given by:*

$$c ::= t \mid x_d \mid c \cdot t \mid \Theta(c),$$

where t is a BPA$_\Theta$ *term and $x_d \in \mathcal{V}_\mathrm{d}$.*

$$(a_1)\ \frac{}{x \xrightarrow{x_s}_{>} x_d} \qquad (a_2)\ \frac{}{x \xrightarrow{x}_{>} \checkmark}$$

$$(a_3)\ \frac{t \xrightarrow{x_s}_{>} c}{t \cdot u \xrightarrow{x_s}_{>} c \cdot u} \qquad (a_4)\ \frac{t \xrightarrow{x}_{>} t'}{t \cdot u \xrightarrow{x}_{>} t' \cdot u} \qquad (a_5)\ \frac{t \xrightarrow{x}_{>} \checkmark}{t \cdot u \xrightarrow{x}_{>} u}$$

$$(a_6)\ \frac{t \xrightarrow{x_s}_{>} c}{t + u \xrightarrow{x_s}_{>} c} \qquad (a_7)\ \frac{t \xrightarrow{x}_{>} t'}{t + u \xrightarrow{x}_{>} t'} \qquad (a_8)\ \frac{t \xrightarrow{x}_{>} \checkmark}{t + u \xrightarrow{x}_{>} \checkmark}$$

$$(a_9)\ \frac{t \xrightarrow{x_s}_{>} c}{\Theta(t) \xrightarrow{x_s}_{>} \Theta(c)} \qquad (a_{10})\ \frac{t \xrightarrow{x}_{>} t'}{\Theta(t) \xrightarrow{x}_{>} \Theta(t')} \qquad (a_{11})\ \frac{t \xrightarrow{x}_{>} \checkmark}{\Theta(t) \xrightarrow{x}_{>} \checkmark}$$

Table 3.4: Inference rules for the auxiliary transition relations. The symmetric versions of rules a_6–a_8 have been omitted.

Notice that the grammar above guarantees that each configuration contains at most one occurrence of a variable in $\mathcal{V}_{\mathrm{d}}$, say x_d, and if such occurrence is in the scope of sequential composition, then x_d must occur as the first symbol in the composition.

Define the set of variable labels $\mathcal{V}_{\mathrm{s}} = \{x_s \mid x \in \mathcal{V}\}$, disjoint from $\mathcal{V}$, and assume any priority order $>$. We then introduce two auxiliary relations $\xrightarrow{x_s}_{>}$, $\xrightarrow{x}_{>}$, and the auxiliary predicate $\xrightarrow{x}_{>}\checkmark$, whose operational semantics is given in Table 3.4. These allow us to express how the initial behavior of a term can be derived from that of the variables occurring in it. Informally, the labels allow us to identify the variable that induces a particular transition. Transitions of the form $t \xrightarrow{x}_{>} t'$ and predicates $t \xrightarrow{x}_{>} \checkmark$ allow us to deal with the case described in item ((I)) above. Conversely, transitions $t \xrightarrow{x_s}_{>} c$ are used for the case in item ((II)). The configuration c stores the *yet-to-terminate* behavior of $\sigma(x)$. As an example, for the terms in item ((II)) we would have $c = x_d \cdot u$, and, since $\sigma(x) \xrightarrow{a}_{>} b$, we would let $\sigma[x_d \mapsto b](c) = b \cdot \sigma(u)$.

The following lemma formalizes the intuitions above. To avoid conflicts with any possible occurrence of the priority operator, we focus only on transitions labeled with actions that are (locally) maximal with respect to the chosen priority operator $>$. This type of transition will be sufficient for our purposes in the rest of the chapter.

Lemma 3.3. *Let t be a process term, x a variable, σ a substitution and $a \in \mathrm{ACT}$ be maximal with respect to $>$. Then:*

1. *If $t \xrightarrow{x}_{>}\checkmark$ and $\sigma(x) \xrightarrow{a}_{>}\checkmark$, then $\sigma(t) \xrightarrow{a}_{>}\checkmark$.*

2. *If $t \xrightarrow{x}_{>} t'$ and $\sigma(x) \xrightarrow{a}_{>}\checkmark$, then $\sigma(t) \xrightarrow{a}_{>} \sigma(t')$.*

3. *If $t \xrightarrow{x_s}_{>} c$ and $\sigma(x) \xrightarrow{a}_{>} p$ for some process p, then $\sigma(t) \xrightarrow{a}_{>} \sigma[x_d \mapsto p](c)$.*

Proof. 1. We proceed by induction over the derivation of the predicate $t \xrightarrow{x}_{>}\checkmark$.

- Base case: $t = x$ and $t \xrightarrow{x}_{>} \surd$ is derived by rule (a_2) in Table 3.4. Hence $\sigma(t) \xrightarrow{a}_{>} \surd$ directly follows by $\sigma(x) \xrightarrow{a}_{>} \surd$.
- Inductive step: $t = t_1 + t_2$ and $t \xrightarrow{x}_{>} \surd$ is derived by either rule (a_8) in Table 3.4, and thus by $t_1 \xrightarrow{x}_{>} \surd$, or its symmetric version on t_2. Assume, without loss of generality, that rule (a_8) in Table 3.4 was applied. Then by induction $t_1 \xrightarrow{x}_{>} \surd$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply $\sigma(t_1) \xrightarrow{a}_{>} \surd$. Hence, the premise of rule (r_4) in Table 3.1 is satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \surd$.
- Inductive step: $t = \Theta(u)$ and $t \xrightarrow{x}_{>} \surd$ is derived by rule (a_{11}) in Table 3.4, and thus we have that $u \xrightarrow{x}_{>} \surd$. By induction $u \xrightarrow{x}_{>} \surd$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply $\sigma(u) \xrightarrow{a}_{>} \surd$. Since, per assumption, action a has maximal priority with respect to $>$, the premises of rule (r_8) in Table 3.1 are satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \surd$.

2. We proceed by induction over the derivation of the auxiliary transition $t \xrightarrow{x}_{>} t'$.

   - Base case: $t = t_1 \cdot t_2$ and $t \xrightarrow{x}_{>} t'$ is derived by rule (a_5) in Table 3.4, namely $t_1 \xrightarrow{x}_{>} \surd$ and $t' = t_2$. By Lemma 3.3.1 we have that $t_1 \xrightarrow{x}_{>} \surd$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply that $\sigma(t_1) \xrightarrow{a}_{>} \surd$. Hence, the premise of rule (r_2) in Table 3.1 is satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \sigma(t_2)$.
   - Inductive step: $t = t_1 \cdot t_2$ and $t \xrightarrow{x}_{>} t'$ is derived by rule (a_4) in Table 3.4, namely $t_1 \xrightarrow{x}_{>} t_1'$ and $t' = t_1' \cdot t_2$. By induction we have that $t_1 \xrightarrow{x}_{>} t_1'$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply that $\sigma(t_1) \xrightarrow{a}_{>} \sigma(t_1')$. Hence, the premise of rule (r_3) in Table 3.1 is satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \sigma(t_1' \cdot t_2)$.
   - Inductive step: $t = t_1 + t_2$ and $t \xrightarrow{x}_{>} t'$ is derived either by rule (a_7) in Table 3.4, namely $t_1 \xrightarrow{x}_{>} t_1'$ and $t' = t_1'$, or by its symmetric version for t_2. Assume, without loss of generality, that rule (a_7) was applied. By induction we have that $t_1 \xrightarrow{x}_{>} t_1'$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply that $\sigma(t_1) \xrightarrow{a}_{>} \sigma(t_1')$. Hence, the premise of rule (r_6) in Table 3.1 is satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \sigma(t_1')$.
   - Inductive step: $t = \Theta(u)$ and $t \xrightarrow{x}_{>} t'$ is derived by rule (a_{10}) in Table 3.4, namely $t_1 \xrightarrow{x}_{>} t_1'$ and $t' = \Theta(t_1')$. By induction we have that $t_1 \xrightarrow{x}_{>} t_1'$ and $\sigma(x) \xrightarrow{a}_{>} \surd$ imply that $\sigma(t_1) \xrightarrow{a}_{>} \sigma(t_1')$. Since by the hypothesis action a has maximal priority with respect to $>$, the premise of rule (r_9) in Table 3.1 is satisfied and we can infer that $\sigma(t) \xrightarrow{a}_{>} \sigma(\Theta(t_1'))$.

3. We proceed by induction over the derivation of the auxiliary transition $t \xrightarrow{x_s}_{>} c$.

   - Base case: $t = x$ and $t \xrightarrow{x_s}_{>} c$ is derived by rule (a_1) in Table 3.4, namely $c = x_d$. Hence the proof follows directly by $\sigma(x) \xrightarrow{a}_{>} p$.

- Inductive step: $t = t_1 \cdot t_2$ and $t \xrightarrow{x_s}_> c$ is derived by rule (a_3) in Table 3.4, namely $t_1 \xrightarrow{x_s}_> c'$ and $c = c' \cdot t_2$. By induction we have that $t_1 \xrightarrow{x_s}_> c'$ and $\sigma(x) \xrightarrow{a}_> p$ imply $\sigma(t_1) \xrightarrow{a}_> p'$ for $p' = \sigma[x_d \mapsto p](c')$. Hence, by rule (r_3) in Table 3.1 we can infer that $\sigma(t) \xrightarrow{a}_> p' \cdot \sigma(t_2)$, with $p' \cdot \sigma(t_2) = \sigma[x_d \mapsto p](c' \cdot t_2)$.
- Inductive step: $t = t_1 + t_2$ and $t \xrightarrow{x_s}_> c$ is derived either by rule (a_6) in Table 3.4, namely $t_1 \xrightarrow{x_s}_> c$, or by its symmetric version for t_2. Assume, without loss of generality, that (a_6) was applied. By induction we have that $t_1 \xrightarrow{x_s}_> c$ and $\sigma(x) \xrightarrow{a}_> p$ imply $\sigma(t_1) \xrightarrow{a}_> \sigma[x_d \mapsto p](c)$. Hence, by rule (r_6) in Table 3.1 we can infer that $\sigma(t) \xrightarrow{a}_> \sigma[x_d \mapsto p](c)$.
- Inductive step: $t = \Theta(u)$ and $t \xrightarrow{x_s}_> \Theta(c)$ is derived by rule (a_9) in Table 3.4, namely $u \xrightarrow{x_s}_> c$. By induction we have that $u \xrightarrow{x_s}_> c$ and $\sigma(x) \xrightarrow{a}_> p$ imply $\sigma(u) \xrightarrow{a}_> \sigma[x_d \mapsto p](c)$. Since by the hypothesis action a has maximal priority with respect to $>$, by rule (r_9) in Table 3.1 we can infer that $\sigma(t) \xrightarrow{a}_> \sigma[x_d \mapsto p](\Theta(c))$. □

We will sometimes need to extend the third case of Lemma 3.3 to sequences of transitions. To this end, we provide first an auxiliary technical lemma, that will simplify our reasoning.

Lemma 3.4. *Let $a \in \textsc{Act}$ be maximal with respect to $>$, and let σ be a closed substitution. Consider a configuration c, and processes p, p' such that $p \xrightarrow{a}_> p'$. If c contains an occurrence of x_d, then $\sigma[x_d \mapsto p](c) \xrightarrow{a}_> \sigma[x_d \mapsto p'](c)$.*

Proof. We proceed by structural induction on c.

- Base case $c = t$: since c does not contain an occurrence of x_d, the lemma is vacuously true.
- Base case $c = x_d$: clearly, $\sigma[x_d \mapsto p](c) = p \xrightarrow{a}_> p' = \sigma[x_d \mapsto p'](c)$.
- Inductive step $c = c' \cdot t$: by induction over c' we obtain $\sigma[x_d \mapsto p](c') \xrightarrow{a}_> \sigma[x_d \mapsto p'](c')$. An application of rule (r_3) in Table 3.1 therefore gives
$$\sigma[x_d \mapsto p](c) = \sigma[x_d \mapsto p](c') \cdot \sigma(t) \xrightarrow{a}_> \sigma[x_d \mapsto p'](c') \cdot \sigma(t) = \sigma[x_d \mapsto p'](c).$$
- Inductive step $c = \Theta(c')$: by induction over c' we have $\sigma[x_d \mapsto p](c') \xrightarrow{a}_> \sigma[x_d \mapsto p'](c')$. Since moreover a is maximal with respect to $>$, by applying rule (r_9) in Table 3.1 we obtain
$$\sigma[x_d \mapsto p](c) = \sigma[x_d \mapsto p](\Theta(c')) \xrightarrow{a}_> \sigma[x_d \mapsto p'](\Theta(c')) = \sigma[x_d \mapsto p'](c). \ \square$$

We can now show that the decomposition of the semantics can be extended to sequences of transitions, and we can thus apply inductive arguments to them.

Lemma 3.5. *Let σ be a closed substitution. If $t \xrightarrow{x_s}_> c$ and $\sigma(x) \to_>^n p$ is such that all actions taken along the transitions from $\sigma(x)$ to p are maximal with respect to $>$, then $\sigma(t) \to_>^n \sigma[x_d \mapsto p](c)$.*

Proof. First of all, we notice that since $t \xrightarrow{x_s}_{>} c$, then c must contain an occurrence of x_d.

We proceed by induction over the derivation of the auxiliary transition $t \xrightarrow{x_s}_{>} c$, and for each case, we prove the statement by proceeding by induction over n. However, in each case, the base case of $n = 1$ is given by Lemma 3.3.3 and it is therefore omitted. Furthermore, we remark that $\sigma(x) \to^n_> p$ can be equivalently rewritten as $\sigma(x) \to^{n-1}_> p' \to_> p$ for some process p'.

- Base case: $t = x$ and $t \xrightarrow{x_s}_{>} c$ is derived by applying rule (a_1) in Table 3.4, so that $c = x_d$. By the induction hypothesis over $n - 1$ we get

  $$\sigma(t) = \sigma(x) \to^{n-1}_> \sigma[x_d \mapsto p'](x_d) = p'.$$

  Since, moreover, $p' \to_> p = \sigma[x_d \mapsto p](c)$ we conclude that $\sigma(t) \to^n_> \sigma[x_d \mapsto p](c)$.

- Inductive step: $t = t_1 \cdot t_2$ and $t \xrightarrow{x_s}_{>} c$ is derived by applying rule (a_3) in Table 3.4, so that $t_1 \xrightarrow{x_s}_{>} c'$, and $c = c' \cdot t_2$. By induction over the derivation of $t_1 \xrightarrow{x_s}_{>} c'$ and $n - 1$, we get $\sigma(t_1) \to^{n-1}_> \sigma[x_d \mapsto p'](c')$, which, by rule (r_3) in Table 3.1, gives

  $$\sigma(t) = \sigma(t_1) \cdot \sigma(t_2) \to^{n-1}_> \sigma[x_d \mapsto p'](c') \cdot \sigma(t_2) = \sigma[x_d \mapsto p'](c).$$

  Since $p' \to_> p$, Lemma 3.4 gives $\sigma[x_d \mapsto p'](c) \to_> \sigma[x_d \mapsto p](c)$. We can therefore conclude that $\sigma(t) \to^n_> \sigma[x_d \mapsto p](c)$.

- Inductive step: $t = t_1 + t_2$ and $t \xrightarrow{x_s}_{>} c$ is derived by applying rule (a_6) in Table 3.4, so that $t_1 \xrightarrow{x_s}_{>} c$. By induction over the derivation of $t_1 \xrightarrow{x_s}_{>} c$ and $n - 1$, we get $\sigma(t_1) \to^{n-1}_> \sigma[x_d \mapsto p'](c)$. Then, by applying rule (r_6) in Table 3.1 and Lemma 3.4 we obtain

  $$\sigma(t) \to^{n-1}_> \sigma[x_d \mapsto p'](c) \to_> \sigma[x_d \mapsto p](c).$$

  A similar argument, using rule (r_7), in place of rule (r_6), allows us to prove the symmetric case of the auxiliary transition triggered by t_2.

- Inductive step: $t = \Theta(t')$ and $t \xrightarrow{x_s}_{>} c$ is derived by applying rule (a_9) in Table 3.4, so that $t' \xrightarrow{x_s}_{>} c'$ and $c = \Theta(c')$. By induction over the derivation of $t' \xrightarrow{x_s}_{>} c'$ and $n - 1$, we infer that $\sigma(t') \to^{n-1}_> \sigma[x_d \mapsto p'](c')$. Hence, by applying rule (r_9) in Table 3.1 and Lemma 3.4, we get

  $$\sigma(t) \to^{n-1}_> \sigma[x_d \mapsto p'](\Theta(c')) = \sigma[x_d \mapsto p'](c) \to_> \sigma[x_d \mapsto p](c). \qquad \square$$

3.4.2 . . . and Back Again

So far we have provided a way to derive the initial behavior of a term from the open transitions available for it, especially when determined by variables. Our aim is now to obtain a converse result: knowing that $\sigma(t) \xrightarrow{a}_{>} p$, we want to characterize its possible sources in the behavior of t and of the closed instances of the variables occurring in t.

Firstly, we remark that in Section 3.4.1 we have considered *open* process terms and thus no occurrence of a priority operator, due to substitutions of variables possibly occurring in them, could have been foreseen. Therefore, to avoid conflicts, we have limited our attention to actions that were (locally) maximal with respect to the considered priority order. However, we now start from the closed process term $\sigma(t)$ and therefore we can properly relate the behavior of the closed instances of variables to their potential occurrence in the scope of a priority operator. To this end, we introduce the relation of *initial enabledness* between a variable x and a term t with respect to a natural number $l \in \mathbb{N}$, notation $x \lhd_l t$. Informally, $x \lhd_l t$ holds if x occurs in the scope of l-nested applications of the priority operator in t and the initial behavior of $\sigma(t)$ is possibly determined by $\sigma(x)$, for all substitutions σ. Initial enabledness extends relation $\lhd_l$ from [13], that was defined on BCCSP_Θ terms, to BPA_Θ terms.

Definition 3.4 (Initial enabledness, $\lhd_l$)**.** *The relations $\lhd_l$, for $l \in \mathbb{N}$, between variables and terms are defined as the least relations satisfying the following constraints:*

1. *$x \lhd_0 x$;*
2. *if $x \lhd_l t$ then $x \lhd_l t + u$ and $x \lhd_l u + t$;*
3. *if $x \lhd_l t$ then $x \lhd_l t \cdot t'$;*
4. *if $x \lhd_l t$ then $x \lhd_{l+1} \Theta(t)$.*

If $x \lhd_l t$, for some $l \in \mathbb{N}$, we say that x is initially enabled *in t. We say that x is* initially disabled *in t, otherwise.*

***Example* 2.** *Consider the terms $t_1 = x \cdot \Theta(u_1)$, for some term u_1 such that $x \notin \mathrm{var}(u_1)$, and $t_2 = \Theta(\Theta(\Theta(t_1 + u_2) \cdot y)) \cdot u_3$, for some variable $y \neq x$ and terms u_2, u_3, such that $x \notin \mathrm{var}(u_2), \mathrm{var}(u_3)$. Then we have that $x \lhd_0 t_1$, $x \lhd_0 t_1 + u_2$ and $x \lhd_3 t_2$, so that x is initially enabled in t_1, $t_1 + u_2$ and t_2.*

Conversely, variable y is initially disabled in t_2 as it occurs as second argument of a sequential composition operator. Notice that this implies that no action performed by any closed substitution instance of y can trigger a transition of the corresponding closed instance of t_2.

As stated by the following lemma, there is a close relation between x being initially enabled in t and the auxiliary transition $t \xrightarrow{x_s}_> c$. We write $t = t_1 \odot t_2$ to mean that either $t = t_1$ or $t = t_1 \cdot t_2$, i.e., t_1 may possibly be sequentially followed by t_2. We extend this notation to nested occurrences of possible sequential compositions $\odot$ by $t \bigodot_{i=1}^{n} t_i = (\dots(t \odot t_1) \odot \dots) \odot t_n$. Then, for a process term t and $l \in \mathbb{N}$ we define the set of terms $\Theta^l_\odot(t)$ inductively as follows:

$$\Theta^0_\odot(t) = \left\{ u \mid u = t \bigodot_{i=1}^{n} t_i \text{ for some } n \in \mathbb{N} \text{ and terms } t_1, \dots, t_n \right\}$$

$$\Theta^{l+1}_\odot(t) = \left\{ u \mid u = \Theta(u' \odot t') \bigodot_{i=1}^{n} t_i \text{ for some } u' \in \Theta^l_\odot(t), n \in \mathbb{N} \text{ and } t', t_1, \dots, t_n \right\}.$$

In what follows, we write $t \rightarrow_> \Theta^l_\odot(t')$ to denote that $t \rightarrow_> u$ for some $u \in \Theta^l_\odot(t')$. Substitutions and transitions are lifted to $\Theta^l_\odot(t)$ in a similar fashion.

Lemma 3.6. *Let x be a variable, t a term and $l \in \mathbb{N}$. Then, $x \lhd_l t$ if and only if $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$.*

Proof. We prove the two implications separately. We recall that $x_d \in \Theta^0_\odot(x_d)$. Moreover, we notice that if $t = a$, then there is no variable x such that either $x \lhd_l t$, or transition $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$, can be inferred for any $l \in \mathbb{N}$.

($\Longrightarrow$) We proceed by structural induction on t in $x \lhd_l t$.

- Base case $t = x$. In this case we have $x \lhd_0 x$ and hence an application of rule (a_1) in Table 3.4 gives $t \xrightarrow{x_s}_> x_d \in \Theta^0_\odot(x_d)$.

- Inductive step $t = t_1 + t_2$. In this case $x \lhd_l t$ may be due either to $x \lhd_l t_1$ or to $x \lhd_l t_2$. If $x \lhd_l t_1$, then by induction over t_1 we get $t_1 \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$, so rule (a_6) in Table 3.4 gives $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$. If $x \lhd_l t_2$, we get the same by result by the symmetric version of rule (a_6).

- Inductive step $t = t_1 \cdot t_2$. Then it must be the case that $x \lhd_l t_1$, so by induction over t_1 we get $t_1 \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$. As $u \cdot t_2 \in \Theta^l_\odot(x_d)$ for all $u \in \Theta^l_\odot(x_d)$, an application of rule (a_3) in Table 3.4 then gives $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$, which is still of the correct form.

- Inductive step $t = \Theta(t')$. In this case $x \lhd_l t$ is due to $x \lhd_{l-1} t'$. By induction over t' we get $t' \xrightarrow{x_s}_> \Theta^{l-1}_\odot(x_d)$. Hence, since $\Theta(u) \in \Theta^l_\odot(x_d)$ for all $u \in \Theta^{l-1}_\odot(x_d)$, by applying rule (a_9) in Table 3.4 we obtain $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$.

($\Longleftarrow$) The proof is by induction on the derivation of the auxiliary transition $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$.

- Base case: $t = x$ and $t \xrightarrow{x_s}_> x_d \in \Theta^0_\odot(x_d)$ is derived by applying rule (a_1) in Table 3.4. We can immediately infer that $l = 0$ and $x \lhd_0 t$.

- Inductive step: $t = t_1 \cdot t_2$ and $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$ is derived by applying rule (a_3) in Table 3.4, so that $t_1 \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$. By induction over the derivation of the auxiliary transition from t_1, we get $x \lhd_l t_1$, which implies $x \lhd_l t_1 \cdot t_2 = t$.

- Inductive step: $t = t_1 + t_2$ and $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$ is derived by applying rule (a_6) in Table 3.4, so that $t_1 \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$. Induction over the derivation of the auxiliary transition from t_1 then gives $x \lhd_l t_1$, which implies $x \lhd_l t_1 + t_2 = t$. The same argument holds for the symmetric version of rule (a_6).

- Inductive step: $t = \Theta(t')$ and $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$ is derived by applying rule (a_9) in Table 3.4, so that $t' \xrightarrow{x_s}_> \Theta^{l-1}_\odot(x_d)$. By induction over the derivation of the auxiliary transition from t', we get $x \lhd_{l-1} t'$, which implies $x \lhd_l \Theta(t') = t$. □

The notation $\Theta^l_\odot(x_d)$ abstracts away from a tail of nested (possible) sequential compositions. This choice is merely for simplification purposes and does not impact the technical development of our results. In fact, the behavior of the terms in the tail and their closed instances will never play a role in the results, as only the contribution of closed instances of x_d to the behavior of terms in $\Theta^l_{\odot}(x_d)$ will be of interest. We remark also that $\Theta^0_{\odot}(x_d)$ denotes a configuration containing an occurrence of x_d which is not in the scope of a priority operator.

***Example* 3.** *Consider the terms t_1, t_2 in Example 2 and assume a priority order $>$. Since $x \xrightarrow{x_s}_> x_d$, by rule ($a_3$) in Table 3.4 we get $t_1 \xrightarrow{x_s}_> x_d \cdot \Theta(u_1)$ which, by rule (a_6) in Table 3.4, gives $t_1 + u_2 \xrightarrow{x_s}_> x_d \cdot \Theta(u_1)$. Hence, by three applications of rule (a_9) and as many of rule (a_3), we infer that $t_2 \xrightarrow{x_s}_> \Theta(\Theta(\Theta(x_d \cdot \Theta(u_1)) \cdot y)) \cdot u_3$. Notice that the right-hand side of the transition from t_2 is of the form $\Theta^3_{\odot}(x_d)$ and that the trailing $\Theta(u_1), y, u_3$ played no role in the derivation of such a transition.*

We are now ready to derive the behavior of the term t and that of the closed instances of the variables occurring in t, from the transitions enabled for $\sigma(t)$.

Proposition 3.1. *Let t be a process term, σ a closed substitution, a an action and p a process. Then:*

1. *If $\sigma(t) \xrightarrow{a}_> \surd\!\!\!\surd$ then*
   - *(a) either $t \xrightarrow{a}_> \surd\!\!\!\surd$;*
   - *(b) or there is a variable x such that $t \xrightarrow{x}_> \surd\!\!\!\surd$ and $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$.*
2. *If $\sigma(t) \xrightarrow{a}_> p$ then one of the following applies:*
   - *(a) there is a process term t' such that $t \xrightarrow{a}_> t'$ and $\sigma(t') = p$;*
   - *(b) there are a process term t' and a variable x such that $t \xrightarrow{x}_> t'$, $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$ and $\sigma(t') = p$;*
   - *(c) there are a variable x, a natural number $l \in \mathbb{N}$, and a process q such that $t \xrightarrow{x_s}_> \Theta^l_{\odot}(x_d)$, $\sigma(x) \xrightarrow{a}_> q$ and $p \in \Theta^l_{\odot}(q)$.*

Proof. 1. We proceed by induction over the derivation of $\sigma(t) \xrightarrow{a}_> \surd\!\!\!\surd$.

- Base case: the last rule applied in the derivation of $\sigma(t) \xrightarrow{a}_> \surd\!\!\!\surd$ is (r_1) in Table 3.1. This means that either $t = a$, or $t = x$ with $\sigma(x) = a$. In the former case it follows that $t \xrightarrow{a}_> \surd\!\!\!\surd$ by rule (r_1) in Table 3.1 and in the latter it follows that $t \xrightarrow{x}_> \surd\!\!\!\surd$ by rule (a_2) in Table 3.4 and $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$.
- Inductive step $t = t_1 + t_2$ and $\sigma(t) \xrightarrow{a}_> \surd\!\!\!\surd$ is derived either by rule (r_4) in Table 3.1, and thus by $\sigma(t_1) \xrightarrow{a}_> \surd\!\!\!\surd$, or by rule ($r_5$) in Table 3.1, and thus by $\sigma(t_2) \xrightarrow{a}_> \surd\!\!\!\surd$. Assume, without loss of generality, that rule (r_4) was applied. By induction over $\sigma(t_1) \xrightarrow{a}_> \surd\!\!\!\surd$ we can distinguish two cases:
  - $t_1 \xrightarrow{a}_> \surd\!\!\!\surd$. Then by rule ($r_4$) in Table 3.1 we derive that $t \xrightarrow{a}_> \surd\!\!\!\surd$.
  - There is a variable x such that $t_1 \xrightarrow{x}_> \surd\!\!\!\surd$ and $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$. Hence, by applying rule (a_8) in Table 3.4 we derive that, for the same variable x, $t \xrightarrow{x}_> \surd\!\!\!\surd$.

- Inductive step: $t = \Theta(u)$ and $\sigma(t) \xrightarrow{a}_{>} \checkmark$ is derived by rule (r_8) in Table 3.1. This implies that $\sigma(u) \xrightarrow{a}_{>} \checkmark$ and $\sigma(u) \not\xrightarrow{b}_{>}$ for all $b > a$. By induction over $\sigma(u) \xrightarrow{a}_{>} \checkmark$ we can distinguish two cases:
  - $u \xrightarrow{a}_{>} \checkmark$. Since moreover from $\sigma(u) \not\xrightarrow{b}_{>}$ for all $b > a$ we can infer that $u \not\xrightarrow{b}_{>}$ for all such b, the premises of rule (r_8) in Table 3.1 are satisfied and we can derive that $t \xrightarrow{a}_{>} \checkmark$.
  - There is a variable x such that $u \xrightarrow{x}_{>} \checkmark$ and $\sigma(x) \xrightarrow{a}_{>} \checkmark$. By applying rule (a_{11}) in Table 3.4 we derive that, for the same variable, $t \xrightarrow{x}_{>} \checkmark$.

2. We proceed by induction over the derivation of $\sigma(t) \xrightarrow{a}_{>} p$. Hence, we assume that the property in Proposition 3.1.2 has been proven for all proper subderivations of the derivation of $\sigma(t) \xrightarrow{a}_{>} p$. We proceed by a case analysis over the structure of t to prove that the desired property holds for $\sigma(t) \xrightarrow{a}_{>} p$ as well. Notice that the case $t = a$ is vacuous, since there is no closed term p such that $a \xrightarrow{a}_{>} p$.

   - Case: $t = x$. Then case (2c) is satisfied directly by rule (a_1) in Table 3.4.
   - Case: $t = t_1 \cdot t_2$. We can distinguish two cases:
     - $\sigma(t) \xrightarrow{a}_{>} p$ is derived by rule (r_2) in Table 3.1, namely by $\sigma(t_1) \xrightarrow{a}_{>} \checkmark$ and $p = \sigma(t_2)$. From $\sigma(t_1) \xrightarrow{a}_{>} \checkmark$ and Proposition 3.1.1 we get that either $t_1 \xrightarrow{a}_{>} \checkmark$ or there is a variable x such that $t_1 \xrightarrow{x}_{>} \checkmark$ and $\sigma(x) \xrightarrow{a}_{>} \checkmark$. In the former case we can apply rule (r_2) in Table 3.1 and obtain $t \xrightarrow{a}_{>} t_2$ with $\sigma(t_2) = p$, thus case (2a) is satisfied. In the latter case we can apply rule (a_5) in Table 3.4 and obtain $t \xrightarrow{x}_{>} t_2$ which together with $\sigma(t_2) = p$ and $\sigma(x) \xrightarrow{a}_{>} \checkmark$ satisfies case (2b).
     - $\sigma(t) \xrightarrow{a}_{>} p$ is derived by rule (r_3) in Table 3.1, namely by $\sigma(t_1) \xrightarrow{a}_{>} p_1$ with $p_1 = q \cdot \sigma(t_2)$. By induction over $\sigma(t_1) \xrightarrow{a}_{>} p_1$ we can distinguish three cases:
       - ∗ Case (2a) applies so that there is a process term t_1' such that $t_1 \xrightarrow{a}_{>} t_1'$ and $\sigma(t_1') = p_1$. Then, by rule (r_3) in Table 3.1 we infer that $t \xrightarrow{a}_{>} t_1' \cdot t_2$ with $\sigma(t_1') \cdot \sigma(t_2) = p$, and thus case (2a) is also satisfied by t.
       - ∗ Case (2b) applies so that there are a process term t_1' and a variable x such that $t_1 \xrightarrow{x}_{>} t_1'$, $\sigma(x) \xrightarrow{a}_{>} \checkmark$ and $\sigma(t_1') = p_1$. Then, by rule (a_4) in Table 3.4 we infer that $t \xrightarrow{x}_{>} t_1' \cdot t_2$ with $\sigma(x) \xrightarrow{a}_{>} \checkmark$ and $\sigma(t_1') \cdot \sigma(t_2) = p$, and thus case (2b) is also satisfied by t.
       - ∗ Case (2c) applies so that there are a variable x, a natural $l \in \mathbb{N}$ and a process s such that $t_1 \xrightarrow{x_s}_{>} \Theta^l_{\odot}(x_d)$, $\sigma(x) \xrightarrow{a}_{>} q$ and $p_1 \in \Theta^l_{\odot}(q)$. Notice that, since in the construction of $\Theta^l_{\odot}(x_d)$ we allow the nesting of trailing sequential components to be of arbitrary depth, we can infer that for all $u \in \Theta^l_{\odot}(x_d)$ the term

$u \cdot t_2$ is also in $\Theta^l_{\odot}(x_d)$. Then, by rule (a_3) in Table 3.4 we infer that $t \xrightarrow{x_s}_{>} \Theta^l_{\odot}(x_d)$. Hence case (2c) is also satisfied by t with respect to $\Theta^l_{\odot}(x_d)$, the variable x, the natural $l \in \mathbb{N}$ and the process q for which $p \in \Theta^l_{\odot}(q)$.

- Case: $t = t_1 + t_2$ and $\sigma(t) \xrightarrow{a}_{>} p$ is derived either from $\sigma(t_1) \xrightarrow{a}_{>} p$ or$\sigma(t_2) \xrightarrow{a}_{>} p$, namely by applying either rule (r_6) or rule (r_7) in Table 3.1. Since induction applies to such a move taken by $\sigma(t_i)$ and in all the rules for nondeterministic choice in Tables 3.1 and 3.4 the moves of t_i are mimicked exactly by t, we can infer that each of the three cases of Proposition 3.1.2 holds for t whenever it holds for t_i.
- Case: $t = \Theta(u)$ and $\sigma(t) \xrightarrow{a}_{>} p$ is derived by applying rule (r_9) in Table 3.1. This implies that $\sigma(u) \xrightarrow{a}_{>} p_1$, with $\Theta(p_1) = p$, and $\sigma(u) \not\xrightarrow{b}_{>}$ for all $b > a$. By induction over $\sigma(u) \xrightarrow{a}_{>} p_1$ we can distinguish three cases:
  - Case (2a) applies so that there is a process term u' such that $u \xrightarrow{a}_{>} u'$ and $\sigma(u') = p_1$. Moreover, we remark that from $\sigma(u) \not\xrightarrow{b}_{>}$ for all $b > a$, it follows that $u \not\xrightarrow{b}_{>}$ for all $b > a$. Then, by rule (r_9) in Table 3.1 we infer that $t \xrightarrow{a}_{>} \Theta(u')$ with $\sigma(\Theta(u')) = p$, and thus case (2a) is also satisfied by t.
  - Case (2b) applies so that there are a process term u' and a variable x such that $u \xrightarrow{x}_{>} u'$, $\sigma(x) \xrightarrow{a}_{>} \surd\!\!\!\surd$ and $\sigma(u') = p_1$. Then, by rule (a_{10}) in Table 3.4 we infer that $t \xrightarrow{x}_{>} \Theta(u')$ with $\sigma(x) \xrightarrow{a}_{>} \surd\!\!\!\surd$ and $\sigma(\Theta(u')) = p$, and thus case (2b) is also satisfied by t.
  - Case (2c) applies so that there are a variable x, a natural $l \in \mathbb{N}$ and a process q such that $u \xrightarrow{x_s}_{>} \Theta^l_{\odot}(x_d)$, $\sigma(x) \xrightarrow{a}_{>} q$ and $p_1 \in \Theta^l_{\odot}(q)$. Now we notice that for each $u \in \Theta^l_{\odot}(x_d)$ it holds that $\Theta(u) \in \Theta^{l+1}_{\odot}(x_d)$. Then, by rule (a_9) in Table 3.4 we infer that $t \xrightarrow{x_s}_{>} \Theta^{l+1}_{\odot}(x_d)$. Hence case (2c) is also satisfied by t with respect to the variable x, the natural $l+1$ and the process q for which $p \in \Theta^{l+1}_{\odot}(q)$. □

Assume a process term t and suppose that depth$(t) = k$ for some $k \in \mathbb{N}$. We recall that the notion of depth as we have defined it in Definition 3.1 is with respect to the empty priority order. Clearly, given any closed substitution σ we will have that depth$(\sigma(t)) = n$ for some $n \geq k$. In particular, whenever n is *strictly* greater than k we can infer that at least one variable occurring in t has been mapped into a process defined using the sequential composition operator. Hence, we need to extend Proposition 3.1 to sequences of transitions of arbitrary length.

To this end, we introduce the following notation: let $w \in (\textsc{Act} \cup \mathcal{V})^*$ be a string $w = \alpha_1 \dots \alpha_h$ in which each α_i can be either an action or a variable. Then, given a substitution σ, we write $t \xrightarrow{s_1 \dots s_h}_{>,w} t'$ if there are process terms $t_0, \dots, t_h$ such that $t = t_0$, $t' = t_h$, and, for all $i \in \{1, \dots, h\}$,

- $s_i \in \textsc{Act}^*$;

- if $\alpha_i \in \mathcal{V}$, then $\sigma(\alpha_i) \xrightarrow{s_i}_{>} \surd\!\!\surd$ and $t_{i-1} \xrightarrow{s_i}_{>} t_i$;
- if $\alpha_i \in \textsc{Act}$, then $s_i = \alpha_i$ and $t_{i-1} \xrightarrow{\alpha_i}_{>} t_i$.

Finally, we write $|s_1 \dots s_h|$ for the *length* of $s_1 \dots s_h$.

***Example* 4.** *Consider the term $t = a \cdot b \cdot x \cdot u$, for some term u, and the strings $w_1 = ab$ and $w_2 = abx$. Clearly, as string w_1 only considers the execution of a particular sequence of actions, we can write $t \xrightarrow{ab}_{>,w_1} x \cdot u$ since $t \xrightarrow{a}_{>} b \cdot x \cdot u \xrightarrow{b}_{>} x \cdot u$. Conversely, string w_2 requires* concatenating *the first two steps of t with the behavior of the variable x. Assume, for instance, a closed substitution σ with $\sigma(x) = a \cdot a \cdot b$, namely $\sigma(x) \xrightarrow{aab}_{>} \surd\!\!\surd$. Then, for the chosen substitution, we can* unfold *the behavior of x in that of t, and write $t \xrightarrow{abaab}_{>,w_2} u$.*

We also notice that by Lemma 3.4, if $p \xrightarrow{a}_{>} p'$ for some action a having (locally) maximal priority with respect to $>$, then $\sigma[x_d \mapsto p](\Theta^l_{\odot}(x_d)) \xrightarrow{a}_{>} \sigma[x_d \mapsto p'](\Theta^l_{\odot}(x_d))$. In this case, we abuse notation slightly and write directly $\Theta^l_{\odot}(p) \xrightarrow{a}_{>} \Theta^l_{\odot}(p')$.

Proposition 3.2. *Let t be a process term, σ a closed substitution, $n \in \mathbb{N}$ and p a process. If $\sigma(t) \to^n_{>} p$ then:*

1. *there exist a process term t', a string $w \in (\textsc{Act} \cup \mathcal{V})^*$ and $s_1 \dots s_h \in \textsc{Act}^*$ such that $t \xrightarrow{s_1 \dots s_h}_{>,w} t'$, $\sigma(t') = p$, and $|s_1 \dots s_h| = n$;*
2. *or $t \xrightarrow{s_1 \dots s_h}_{>,w} t'$ for some $w \in (\textsc{Act} \cup \mathcal{V})^*$ and $s_1 \dots s_h$ such that $|s_1 \dots s_h| = k < n$, and there are a variable x, a natural number $l \in \mathbb{N}$ and a process q, such that $t' \xrightarrow{x_s}_{>} \Theta^l_{\odot}(x_d)$, $\sigma(x) \to^{n-k}_{>} q$ and $p \in \Theta^l_{\odot}(q)$.*

Proof. We proceed by induction over n.

- Base case $n = 1$. This directly follows by Proposition 3.1.2.
- Inductive step $n > 1$. $\sigma(t) \to^n_{>} p$ is equivalent to writing $\sigma(t) \to_{>} p_1 \to^{n-1}_{>} p$, for some process p_1. We can assume without loss of generality that $\sigma(t) \xrightarrow{a}_{>} p_1$. According to Proposition 3.1.2, from $\sigma(t) \xrightarrow{a}_{>} p_1$ we can distinguish three cases:
  1. there is a process term t_1 such that $t \xrightarrow{a}_{>} t_1$ and $\sigma(t_1) = p_1$. Then by induction over $p_1 \to^{n-1}_{>} p$ we can distinguish two subcases:
     - there is $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$ with $t_1 \xrightarrow{s_1 \dots s_h}_{>,w_1} t'$ such that $|s_1 \dots s_h| = n - 1$ and $\sigma(t') = p$. Then, the proof can be concluded by noticing that for the sequence $w = aw_1$ we get $t \xrightarrow{as_1 \dots s_h}_{>,w} t'$ with $|as_1 \dots s_h| = n$ and $\sigma(t') = p$.
     - there are $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$, a variable y, a natural $l \in \mathbb{N}$ and a process q, such that $t_1 \xrightarrow{s_1 \dots s_h}_{>,w_1} t'$ with $|s_1 \dots s_h| = k < n - 1$, $t' \xrightarrow{y_s}_{>} \Theta^l_{\odot}(y_d)$, $\sigma(y) \to^{n-1-k}_{>} q$ and $p \in \Theta^l_{\odot}(q)$. Then, the proof can be concluded by noticing that for the sequence $w = aw_1$ we get $t \xrightarrow{as_1 \dots s_h}_{>,w} t'$ with $|as_1 \dots s_h| = k + 1 < n$ and y, l, q behave as before.

2. there are a process term t_1 and a variable x such that $t \xrightarrow{x}_> t_1$, $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$ and $\sigma(t_1) = p_1$. Then by induction over $p_1 \to_>^{n-1} p$ we can distinguish two subcases:
   - there is $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$ with $t_1 \xrightarrow{s_1 \ldots s_h}_> t'$ such that $|s_1 \ldots s_h| = n - 1$ and $\sigma(t') = p$. Then, the proof can be concluded by noticing that for the sequence $w = xw_1$ we get $t \xrightarrow{as_1 \ldots s_h}_{>,w} t'$ with $|as_1 \ldots s_h| = n$, as $|a| = 1$, and $\sigma(t') = p$.
   - there are $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$, a variable y, a natural $l \in \mathbb{N}$ and a process q, such that $t_1 \xrightarrow{s_1 \ldots s_h}_{>,w_1} t'$ with $|s_1 \ldots s_h| = k < n - 1$, $t' \xrightarrow{y_s}_> \Theta^l_\odot(y_d)$, $\sigma(y) \to_>^{n-1-k} q$ and $p \in \Theta^l_\odot(q)$. Then, the proof can be concluded by noticing that, since $\sigma(x) \xrightarrow{a}_> \surd\!\!\!\surd$ gives $|a| = 1$, for the sequence $w = xw_1$ we get $t \xrightarrow{as_1 \ldots s_h}_{>,w} t'$ with $|as_1 \ldots s_h| = k + 1 < n$ and c, x, q behave as before.
3. there are a variable x, a natural $l \in \mathbb{N}$ and a process p' such that $t \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$, $\sigma(x) \xrightarrow{a}_> p'$ and $p_1 \in \Theta^l_\odot(p')$. Recall that, per assumption, $p_1 \to_>^{n-1} p$. Since, $p_1 \in \Theta^l_\odot(p')$, we have that either all, or part of, the transitions in the sequence $p_1 \to_>^{n-1} p$ are executed within the scope of a priority operator (unless $l = 0$, but then this case would be an instance of Proposition 3.2.1). Therefore, we are guaranteed that the actions labelling the transitions that are performed in the scope of Θ are all locally maximal with respect to $>$. Therefore, Lemma 3.5 allows us to distinguish two cases:
   - $\sigma(x) \to_>^h q$ for some $h \geq n$. In this case the proposition follows by taking the empty string for w and the process q' such that $\sigma(x) \to_>^n q'$ and $p \in \Theta^l_\odot(q')$.
   - $\sigma(x) \to_>^k q \to_> \surd\!\!\!\surd$ for some $k < n$. Notice that this implies that there is some string s_x with $|s_x| = k$ of actions that have been performed by $\sigma(x)$. Due to the structure of $\Theta^l_\odot(x_d)$ we can infer that there are a natural $m \in \mathbb{N}$ and a process term

     $$t_1 = \underbrace{\Theta(\cdots\Theta}_{m \text{ times}}(t'' \odot u_{m+1}) \odot u_m) \ldots) \odot u_1$$

     such that $\sigma(t) \to_>^k \sigma(t_1) = p_1$. Since then $p_1 \to_>^{n-k} p$, by induction we can distinguish two subcases:
     * there is some $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$ with $t_1 \xrightarrow{s_1 \ldots s_h}_{>,w_1} t'$ such that $|s_1 \ldots s_h| = n - k$ and $\sigma(t') = p$. Then, the proof can be concluded by noticing that for the sequence $w = xw_1$ we get $t \xrightarrow{s_x s_1 \ldots s_h}_{>,w} t'$ with $|s_x s_1 \ldots s_h| = n$, as $|s_x| = k$, and $\sigma(t') = p$.
     * there are $w_1 \in (\textsc{Act} \cup \mathcal{V})^*$, a variable y, a process q', and $m' \in \mathbb{N}$, such that $t_1 \xrightarrow{s_1 \ldots s_h}_{>,w_1} t'$ with $|s_1 \ldots s_h| = j < n - k$, $t' \xrightarrow{y_s}_> \Theta^{m'}_\odot(y_d)$, $\sigma(y) \to_>^{n-k-j} q'$ and $p \in \Theta^{m'}_\odot(q')$. Then, the proof can be concluded by noticing that, as $|s_x| = k$, for the sequence

$w = xw_1$ we get $t \xrightarrow{s_x s_1 \ldots s_h}_{>,w} t'$ with $|s_x s_1 \ldots s_h| = k + j < n$ and y, m', q' as above. □

The following result allows us to establish whether the behavior of two bisimilar process terms is determined by the same variable. Moreover, it guarantees that such a variable is initially enabled in one term if and only if it is initially enabled in the other one.

Theorem 3.2. *Assume that* Act *contains at least two actions, a and b. Let x be a variable. Consider two process terms t and u such that* $\mathrm{init}^\omega(t) \subseteq \{a\}$ *and* $t \mathrel{\underline{\leftrightarrow}}_* u$. *Whenever there is t' such that $t \to^k t'$, for some $k \in \mathbb{N}$, and $x \lhd_l t'$, for some $l \in \mathbb{N}$, then there is u' such that $u \to^k u'$ and $x \lhd_m u'$ for some $m \in \mathbb{N}$. Moreover, $l = 0$ if and only if $m = 0$.*

Proof. Let $n \in \mathbb{N}$ be larger than the depths of t and u, and assume the priority order $> = \{(b, a)\}$ over Act. We define the family of closed substitutions $\{\sigma_i\}_{i \in \mathbb{N}}$ inductively as follows:

$$\sigma_0(y) = \begin{cases} a + b & \text{if } y = x \\ a & \text{otherwise.} \end{cases}$$

$$\sigma_i(y) = \begin{cases} a \cdot (\sigma_{i-1}(y) + a) & \text{if } y = x \\ a & \text{otherwise.} \end{cases}$$

Let $\sigma = \sigma_n$. Suppose that $t \to^k t'$, for some $k \in \mathbb{N}$. As $\mathrm{init}^\omega(t) \subseteq \{a\}$ we can infer that there are process terms $t_0, \ldots, t_k$ such that $t = t_0 \xrightarrow{a} \ldots \xrightarrow{a} t_k = t'$ (if $\mathrm{init}(t) = \emptyset$ then $k = 0$ and $t = t'$). Moreover, as in all such terms t_i there is no occurrence of b, a is maximal with respect to $>$ on them, and thus by Lemma 3.2 and an easy induction over k, we obtain that $\sigma(t_0) \xrightarrow{a}{}^{k} \sigma(t_k) = \sigma(t')$ ($\sigma(t) = \sigma(t')$ if $\mathrm{init}(t) = \emptyset$). Suppose now that $x \lhd_l t'$, for some $l \in \mathbb{N}$. By Lemma 3.6, $x \lhd_l t'$ implies that $t' \xrightarrow{x_s}_> \Theta^l_\odot(x_d)$. By the choice of σ we have that $\sigma(x) \xrightarrow{a}{}^{n}_{>} a + b$. Therefore, by Lemma 3.5 we obtain that $\sigma(t') \xrightarrow{a}{}^{n}_{>} p$ for some $p \in \Theta^l_\odot(a + b)$. By combining the two sequences of transitions, we get $\sigma(t) \xrightarrow{a}{}^{k+n}_{>} p$. By the hypothesis we have $t \mathrel{\underline{\leftrightarrow}}_* u$, which in particular implies $t \mathrel{\underline{\leftrightarrow}}_> u$ and thus $\sigma(t) \mathrel{\underline{\leftrightarrow}}_> \sigma(u)$. As $\underline{\leftrightarrow}_>$ is a bisimulation, we can infer that $\sigma(u) \xrightarrow{a}{}^{k+n}_{>} p'$ for some process p' with $p \mathrel{\underline{\leftrightarrow}}_> p'$. As n is larger than the depth of u, by Proposition 3.2 there exists a process term u', a string w with strings $s_1, \ldots, s_h \in \{a\}^*$, a variable y, a natural number m and a process q such that $u \xrightarrow{s_1 \ldots s_h}_{>,w} u'$, $|s_1 \ldots s_h| = j < n$, $u' \xrightarrow{x_s}_> \Theta^m_\odot(y_d)$, $\sigma(y) \to^{k+n-j}_> q$ and $p' \in \Theta^m_\odot(q)$. Therefore:

1. by $k + n - j > 0$;

2. by the choice of $>$ (which gives that the only possible transition enabled for $\Theta^l_\odot(a + b)$ is a b-labeled move);

3. by the choice of σ;

4. by $p \mathrel{\underline{\leftrightarrow}}_> p'$ with $p \in \Theta^l_\odot(a + b), p' \in \Theta^m_\odot(q)$;

we can conclude that $y = x$, $j = k$ and $q = a + b$. Moreover, from item (4) and the choice of $>$, we obtain that $l = 0$ iff $m = 0$. □

3.5 Making Order-Insensitive Bisimilarity Coinductive: Uniform Determinacy

As outlined in Section 3.2, $\underline{\leftrightarrow}_*$ cannot be defined coinductively, contrary to other bisimulation relations. However, in this section we identify a class of processes for which the coinductive reasoning on $\underline{\leftrightarrow}_*$ can be at least partially recovered, and which will be useful later on.

Definition 3.5 (Uniform determinacy). *Let p be a process. We say that p is* uniformly determinate *if $|\mathrm{init}(p)| = 1$, and for all processes p_1 and p_2 such that $p \to p_1$ and $p \to p_2$, we have $\mathrm{norm}\,(p_1) = \mathrm{norm}\,(p_2) = 1$ and $p_1 \underline{\leftrightarrow}_* p_2$. Then, for each $k \in \mathbb{N}$, we say that p is* uniformly k-determinate *if*

- $|\mathrm{init}(p)| = 1$,
- *whenever $p \to^h q$ for some $h \leq k$ then $|\mathrm{init}(q)| = 1$, and*
- *whenever $p \to^k p'$ then p' is uniformly determinate.*

We remark that uniform determinacy and uniform k-determinacy are defined in terms of the empty priority order.

Summarizing, a process is uniformly k-determinate if whenever it takes k steps, it ends up in a process that only has one available action, and in which all immediate successors have norm 1 and are order-insensitive bisimilar.

***Example* 5.** *Consider processes*

$$p_1 = a \cdot b + a \qquad p_2 = a \cdot p_1 + a \qquad p = a \cdot p_2$$
$$q = a \cdot b + a \cdot a \ .$$

First of all, we notice that both p_1 and p_2 have norm 1, due to the branches with the action constant a. Moreover, they are both uniformly determinate. In fact, p_1 can perform only one transition to process b, which has norm 1 and it is clearly order-insensitive bisimilar to itself. Similarly, the only available transition for p_2 is $p_2 \xrightarrow{a} p_1$, which, as previously noticed, has norm 1. We remark that the a action constants in p_1 and p_2 do not trigger any transition for the two processes, but they cause the predicates $p_1 \xrightarrow{a}\surd$ and $p_2 \xrightarrow{a}\surd$ to hold.

As process p can perform only one a-move to p_2, we can directly infer that p is uniformly determinate. Notice that process p does not have norm 1, but such a constraint has to be satisfied only by its derivatives. Moreover, from our observations on p_1 and p_2, we obtain that p is also uniformly 1-determinate and uniformly 2-determinate.

Consider now process q. We have that q is not uniformly determinate since $q \xrightarrow{a} b$ and $q \xrightarrow{a} a$ are both derivable and, clearly, $b \not\underline{\leftrightarrow}_* a$. However, q is uniformly 1-determinate, since both b and a are trivially uniformly determinate.

The notion of uniform k-determinacy is preserved by order-insensitive bisimilarity.

Lemma 3.7. *If* $p \mathrel{\underline{\leftrightarrow}}_* q$ *and* p *is uniformly* k*-determinate for all* $1 \leq k < \text{depth}(p)$*, then so is* q*.*

Proof. The proof proceeds by induction on k. Notice that $p \mathrel{\underline{\leftrightarrow}}_* q$ implies $p \mathrel{\underline{\leftrightarrow}} q$.

- Base case: $k = 1$. Assume, towards a contradiction, that q is not uniformly 1-determinate. This means that either $|\text{init}(q)| > 1$ or there exist q_1 and q_2 such that $q \rightarrow q_1$ and $q \rightarrow q_2$ but $q_1 \not\mathrel{\underline{\leftrightarrow}}_\star q_2$, or $\text{norm}(q_1) \neq 1$, or $\text{norm}(q_2) \neq 1$.

  If $|\text{init}(q)| > 1$, then there are $a, b \in \textsc{Act}$ with $a \neq b$ such that $q \xrightarrow{a} q_a$ and $q \xrightarrow{b} q_b$ for some processes q_a and q_b. Since $p \mathrel{\underline{\leftrightarrow}} q$, there must exist p_a and p_b such that $p \xrightarrow{a} p_a$ and $p \xrightarrow{b} p_b$, but this contradicts $|\text{init}(p)| = 1$.

  If $q_1 \not\mathrel{\underline{\leftrightarrow}}_* q_2$, then $q_1 \not\mathrel{\underline{\leftrightarrow}}_> q_2$ for some priority order $>$. Since we already know that $|\text{init}(q)| = 1$, $q \rightarrow q_1$ and $q \rightarrow q_2$ implies $q \rightarrow_> q_1$ and $q \rightarrow_> q_2$. Hence there exist processes p_1 and p_2 such that $p \rightarrow_> p_1$ and $p \rightarrow_> p_2$ with $p_1 \mathrel{\underline{\leftrightarrow}}_> q_1$ and $p_2 \mathrel{\underline{\leftrightarrow}}_> q_2$. However, since p is uniformly 1-determinate, we know that $p_1 \mathrel{\underline{\leftrightarrow}}_> p_2$, so $q_1 \mathrel{\underline{\leftrightarrow}}_> q_2$, which is a contradiction.

  If $\text{norm}(q_1) \neq 1$, then we know from $p \mathrel{\underline{\leftrightarrow}} q$ and $q \rightarrow q_1$ that $p \rightarrow p_1$ for some process p_1 with $p_1 \mathrel{\underline{\leftrightarrow}} q_1$. But this implies $\text{norm}(q_1) = \text{norm}(p_1) = 1$, which is a contradiction. The argument for $\text{norm}(q_2) \neq 1$ is similar.

- Inductive step: $k > 1$. Assume that q is uniformly k'-determinate for all $k' < k$. We now prove that q is also uniformly k-determinate. Assume towards a contradiction that q is not k-determinate. Then there must exist some q' such that $q \rightarrow^k q'$ and either $|\text{init}(q')| > 1$ or there are q_1 and q_2 such that $q' \rightarrow q_1$ and $q' \rightarrow q_2$, but either $q_1 \not\mathrel{\underline{\leftrightarrow}}_* q_2$, $\text{norm}(q_1) \neq 1$, or $\text{norm}(q_2) \neq 1$.

  The cases of $|\text{init}(q')| > 1$, $\text{norm}(q_1) \neq 1$, and $\text{norm}(q_2) \neq 1$ are essentially the same as for the base case, except that one first gets a process p' such that $p \rightarrow^k p'$, and then reasons as before on p'.

  We now consider the case of $q_1 \not\mathrel{\underline{\leftrightarrow}}_* q_2$. This implies that $q_1 \not\mathrel{\underline{\leftrightarrow}}_> q_2$ for some priority order $>$. Since $p \mathrel{\underline{\leftrightarrow}}_* q$, we also get $p \mathrel{\underline{\leftrightarrow}}_> q$, and since q is uniformly k'-determinate for every $k' < k$, $q \rightarrow^k q'$ implies $q \rightarrow^k_> q'$. (Recall that all the processes reached in the sequence of k'-steps can perform only transitions with the same label). Therefore there exists a process p' such that $p \rightarrow^k_> p'$ and $p' \mathrel{\underline{\leftrightarrow}}_> q'$. Since we already know that $|\text{init}(q')| = 1$, $q' \rightarrow q_1$ and $q' \rightarrow q_2$ implies $q' \rightarrow_> q_1$ and $q' \rightarrow_> q_2$. Hence there exist p_1 and p_2 such that $p' \rightarrow_> p_1$ and $p' \rightarrow_> p_2$ as well as $p_1 \mathrel{\underline{\leftrightarrow}}_> q_1$ and $p_2 \mathrel{\underline{\leftrightarrow}}_> q_2$. However, since p is uniformly k-determinate, we know that $p_1 \mathrel{\underline{\leftrightarrow}}_> p_2$, so we get $q_1 \mathrel{\underline{\leftrightarrow}}_> q_2$, which contradicts our assumption. □

The next proposition shows that if p and q are order-insensitive bisimilar as well as uniformly k-determinate for all k less than some n, then every sequence of n transitions that p can do can be matched by q such that p and q end up in processes that are again order-insensitive bisimilar.

Proposition 3.3. *Let p and q be two processes such that $p \underline{\leftrightarrow}_* q$ and there is an $n \in \mathbb{N}$ such that p and q are uniformly k-determinate for all $k < n$. Suppose that $p \rightarrow^n p'$ for some p'. Then there is a process q' such that $q \rightarrow^n q'$ and $p' \underline{\leftrightarrow}_* q'$.*

Proof. We recall that in [13] a process p is said to be determinate if $|\text{init}(p)| \leq 1$ ([13] considers the language BCCSP which includes the idle process that cannot perform any action), and for all processes p_1, p_2 such that $p \rightarrow p_1$ and $p \rightarrow p_2$ it holds that $p_1 \underline{\leftrightarrow}_* p_2$. Then p is said to be determinate at depth k if all processes p' such that $p \rightarrow^k p'$ are determinate. Since our notion of uniformly k-determinacy implies that of determinacy at depth k in [13], the proof of this proposition directly follows from Lemma 18 of [13]. □

3.6 The Property: Uniform (n, Θ)-Dependency

In this section we formalize the uniform (n, Θ)-dependency property, on which our negative result is built. As previously outlined, this is based on the notion of Θ-*dependent* process from [13].

Definition 3.6 (Θ-dependent process, [13])**.** *A process p is* Θ-dependent *if there exist priority orders $>_1$ and $>_2$ such that* $\text{init}_{>_1}(p) \neq \text{init}_{>_2}(p)$.

Intuitively, a process is Θ-dependent if its possible behavior depends on the choice of priority order. For example, $\Theta(a + b)$ is Θ-dependent, since we can find a priority order that only allows it to make an a-transition, and another priority order that only allows it to make a b-transition. On the other hand, $\Theta(a)$ is not Θ-dependent, since no matter what priority order we choose, it can only do a a-transition.

Moreover, we will make use of the following technical result from [13].

Lemma 3.8 ([13, Lemma 14])**.** *If $p \underline{\leftrightarrow}_* q$ and p is Θ-dependent, then so is q.*

Uniform (n, Θ)-dependency is an extension of Θ-dependency from [13], in that it requires first that it is possible to take a sequence of n transitions and end up in a process that is Θ-dependent, and furthermore it requires that at each step along the way, the process has a norm of 1.

Definition 3.7. *A process p is* uniformly (n, Θ)-dependent *if there are processes $p_1, \ldots, p_n$ such that $p = p_0 \rightarrow p_1 \rightarrow \cdots \rightarrow p_n$, the process p_n is Θ-dependent, and for all $0 \leq k < n$ we have* $\text{norm}(p_k) = 1$.

The following proposition tells us that (n, Θ)-dependency is preserved by closed instantiations of sound equations whose depth is smaller than n and that satisfy some determinacy constraints.

Proposition 3.4. *Let σ be a closed substitution and let t and u be process terms such that $t \underline{\leftrightarrow}_* u$ and $\text{init}^{\omega}(t) = \{a\}$. Assume a natural number $n \in \mathbb{N}$ such that $n > \max\{\text{depth}(t), \text{depth}(u)\}$ and $\sigma(t)$ is uniformly k-determinate for all $1 \leq k \leq n - 1$. If $\sigma(t)$ is uniformly (n, Θ)-dependent, then so is $\sigma(u)$.*

Proof. We start by noticing that $t \underline{\leftrightarrow}_* u$ implies $\sigma(t) \underline{\leftrightarrow}_* \sigma(u)$ and thus, by Lemma3.7, we infer that $\sigma(u)$ is uniformly k-determinate for all $1 \leq k \leq n-1$. Next, since $\sigma(t)$ is uniformly (n, Θ)-dependent, by Definition 3.5 there are processes $p_0, \ldots, p_n$ such that $\sigma(t) = p_0 \to \ldots \to p_n$, $\operatorname{norm}(p_i) = 1$ for all $i = 0, \ldots, n-1$, and p_n is Θ-dependent. Since, moreover, we have $\operatorname{depth}(t) < n$, by Proposition 3.2 there are a process term t' and a string w such that $t \xrightarrow{s_1 \ldots s_h}_w t'$ with $|s_1 \ldots s_h| = j$ and there are a variable x, an $l \in \mathbb{N}$ and a process q such that $t' \xrightarrow{x_s} \Theta^l_{\odot}(x_d)$, $\sigma(x) \to^{n-j} q$, and $p_n \in \Theta^l_{\odot}(q)$. In particular, notice that by the uniform k-determinacy of $\sigma(t)$, for all $k = 1, \ldots, n-1$, we obtain that $|\operatorname{init}(\sigma(t'))| = 1$. As this set of initials is constructed with respect to the empty priority order we can also infer the following:

- $|\operatorname{init}(\sigma(x))| = 1$,
- $|\operatorname{init}(q_i)| = 1$ for all q_i, with $i = 1, \ldots, n-j-1$, such that $\sigma(x) \to q_1 \to \ldots \to q_{n-j-1} \to q$, and
- any action performed in the sequence $\sigma(x) \to {}^{n-j} q$ is locally maximal with respect to the empty priority order.

Notice that, by Lemma 3.6, $t' \xrightarrow{x_s} \Theta^l_{\odot}(x_d)$ is the same as $x \lhd_l t'$. Since, moreover, p_n is Θ-dependent, it must be the case that $|\textsc{Act}| > 1$. We can then apply Theorem 3.2, thus obtaining that there are a process term u' and an $m \in \mathbb{N}$ such that $u \to^j u'$ and $x \lhd_m u'$. Using again Lemma 3.6, $x \lhd_m u'$ is the same as $u' \xrightarrow{x_s} \Theta^m_{\odot}(x_d)$. As above, the uniform k-determinacy of $\sigma(u)$, for all $k = 1, \ldots, n-1$, guarantees that $|\operatorname{init}(\sigma(u'))| = 1$ and thus that $\sigma(x)$ can perform its (locally maximal) action. Thus, from $\sigma(x) \xrightarrow{a}{}^{n-j} q$ and $u' \xrightarrow{x_s} \Theta^m_{\odot}(x_d)$, Lemma 3.5 implies $\sigma(u') \xrightarrow{a}{}^{n-j} \Theta^m_{\odot}(q)$. Hence we can infer that there are processes $q_0, \ldots, q_n$ such that $\sigma(u) = q_0 \to \ldots \to q_n$ with $q_n \in \Theta^m_{\odot}(q)$. According to Theorem 3.2, we can distinguish two cases:

- Case $l > 0$. Then we can infer that $m > 0$, and thus q_n is clearly Θ-dependent.
- Case $l = 0$. Then we have that $p_n = q$ and from p_n being Θ-dependent we can infer that q is Θ-dependent. As $l = 0$ implies $m = 0$, we get that $q_n = q$ and thus q_n is Θ-dependent because q is.

To conclude, we need to show that $\operatorname{norm}(q_i) = 1$ for each $i = 0, \ldots, n-1$. First of all we notice that, since $\sigma(t) \underline{\leftrightarrow}_* \sigma(u)$ and $\operatorname{norm}(\sigma(t)) = 1$, then $\operatorname{norm}(\sigma(u)) = \operatorname{norm}(q_0) = 1$. Moreover, since $\sigma(u)$ is uniformly k-determinate for all $1 \leq k < n$, we get that $\operatorname{norm}(q_i) = 1$ for all $i = 1, \ldots, n-1$ is guaranteed by Definition 3.5. We can therefore conclude that $\sigma(u)$ is uniformly (n, Θ)-dependent. □

3.7 The Negative Result Over BPA$_\Theta$

This section is devoted to our main result, namely that order-insensitive bisimilarity has no finite ground-complete axiomatization in the setting of BPA$_\Theta$.

In Equation (3.1) in Section 3.3, we presented a family of infinitely many sound equations which cannot be derived from any finite axiom system which is sound modulo order-insensitive bisimilarity, which we now proceed to recall. We make use of the following processes, which are defined for each $n \in \mathbb{N}$ as

$$P_n = A_n(a) + A_n(b) + A_n(a+b),$$

where $A_0(p) = p$ and $A_n(p) = a \cdot A_{n-1}(p) + a$. Process P_n must decide at the top level whether after n steps it will end up in a, b, or $a+b$. After this choice, it can take up to n a-transitions, and at each step it can choose whether to terminate or to continue. We stress that the possibility of termination at each step is crucial, as it implies that $A_n(\cdot)$ cannot be written just with sequential composition modulo bisimilarity.

The family of equations that we consider is then

$$e_n : \quad P_n + A_n(\Theta(a+b)) \approx P_n \qquad (n \geq 0) \ .$$

We remark that the processes on the left-hand side of each equation e_n are uniformly (n, Θ)-dependent, whereas those on the right-hand side do not enjoy this property. In detail, for all $n \in \mathbb{N}$, by construction there is no occurrence of Θ in P_n nor in its derivatives, so that P_n cannot have any Θ-dependent successor. On the other hand, we have $P_n + A_n(\Theta(a+b)) \xrightarrow{a} A_{n-1}(\Theta(a+b)) \xrightarrow{a} \ldots \xrightarrow{a} A_0(\Theta(a+b)) = \Theta(a+b)$ with $\Theta(a+b)$ a Θ-dependent process and, by construction, for each $i = 1, \ldots, n$ the process $A_i(\Theta(a+b))$ has norm 1.

To proceed to the proof of Theorem 3.1 we need to show, in the first place, that all the equations in the family $\{e_n\}_{n\in\mathbb{N}}$ are sound. To this end we introduce the final ingredient that we need for our main result, namely the notion of *summand* of a process.

Definition 3.8 (Summand, [13])**.** *We say that* p is a summand of q*, denoted by* $p \sqsubseteq_* q$*, if there exists a process* r *such that* $p + r \underline{\leftrightarrow}_* q$*.*

Proposition 3.5. *For every* $n \in \mathbb{N}$*, the equation* $P_n + A_n(\Theta(a+b)) \approx P_n$ *is sound.*

Proof. It is enough to prove that $A_n(\Theta(a+b)) \sqsubseteq_* P_n$ for all $n \in \mathbb{N}$. So, let $n \in \mathbb{N}$ and $>$ be an arbitrary priority order. Then:

- If $a > b$, then $A_n(\Theta(a+b)) \underline{\leftrightarrow}_> A_n(a)$.
- If $b > a$, then $A_n(\Theta(a+b)) \underline{\leftrightarrow}_> A_n(b)$.
- If a and b are unordered in $>$, then $A_n(\Theta(a+b)) \underline{\leftrightarrow}_> A_n(a+b)$.

Hence, we can conclude that $A_n(\Theta(a+b)) + P_n \underline{\leftrightarrow}_> P_n$ for all priority orders $>$ and naturals $n \in \mathbb{N}$, which implies $A_n(\Theta(a+b)) \sqsubseteq_* P_n$ for all $n \in \mathbb{N}$ □

Interestingly, any process p such that $p \sqsubseteq_* P_n$ must be of a specific form that inherits many of the features of P_n. In particular, such a process must be k-determinate for all k less than n.

Lemma 3.9. *Let* p *be a process and assume* $p \sqsubseteq_* P_n$ *for some* $n \in \mathbb{N}$*. Then* p *is uniformly* k*-determinate for all* $1 \leq k < n$*.*

Proof. We first prove that $\mathrm{init}^k(p) = \{a\}$ for $0 \leq k < n$. We recall that since we are considering BPA$_\Theta$ with constants, and without the empty process and deadlock, for all closed process terms p it holds that $\mathrm{init}_>(p) \neq \emptyset$ for all priority orders $>$. As $p \sqsubseteq_* P_n$, which means that $p + r \mathrel{\underline{\leftrightarrow}}_* P_n$ for some r, we have that $p + r \mathrel{\underline{\leftrightarrow}} P_n$. By Lemma 3.1, we infer $\mathrm{init}^k(p + r) = \mathrm{init}^k(P_n) = \{a\}$. Since, moreover, $\mathrm{init}^k(p) \subseteq \mathrm{init}^k(p + r)$, we get $\mathrm{init}^k(p) = \{a\}$.

We now proceed by contradiction. Let $1 \leq k < n$ be the least number such that p is not uniformly k-determinate. Then there exist processes p', p_1, and p_2 such that $p \to^k p'$, $p' \to p_1$, and $p' \to p_2$, and $p_1 \not\mathrel{\underline{\leftrightarrow}}_* p_2$, or $\mathrm{norm}\,(p_1) \neq 1$, or $\mathrm{norm}\,(p_2) \neq 1$.

If $\mathrm{norm}\,(p_1) \neq 1$, then $p \to^k p'$ and $p' \to p_1$, so there exists P'_n and P''_n such that $P_n \to^k P'_n$ and $P'_n \to P''_n$ with $p_1 \mathrel{\underline{\leftrightarrow}} P''_n$. But then $\mathrm{norm}\,(p_1) = \mathrm{norm}\,(P''_n) = 1$, which is a contradiction. A similar argument holds when $\mathrm{norm}\,(p_2) \neq 1$.

If $p_1 \not\mathrel{\underline{\leftrightarrow}}_* p_2$, then $p_1 \not\mathrel{\underline{\leftrightarrow}}_> p_2$ for some specific priority order $>$. Notice that since $|\mathrm{init}^i(p)\,| = \{a\}$ for all $0 \leq i < n$, we get that $p \to^k p'$, $p' \to p_1$, and $p' \to p_2$ implies $p \to^k_> p'$, $p' \to_> p_1$, and $p' \to_> p_2$. Since $p + r \mathrel{\underline{\leftrightarrow}}_> P_n$ for some r, there exist P'_n, P''_n, and P'''_n such that $P_n \to^k_> P'_n$, $P'_n \to_> P''_n$, and $P'_n \to_> P'''_n$ with $p_1 \mathrel{\underline{\leftrightarrow}}_> P''_n$ and $p_2 \mathrel{\underline{\leftrightarrow}}_> P''_n$. Since $\mathrm{norm}\,(p_1) = 1 = \mathrm{norm}\,(p_2)$, we also get $\mathrm{norm}\,(P''_n) = 1 = \mathrm{norm}\,(P'''_n)$. However, we see from the definition of P_n that P'_n has a unique successor with norm 1. Hence it follows that $P''_n = P'''_n$, so $p_1 \mathrel{\underline{\leftrightarrow}}_> P''_n = P'''_n \mathrel{\underline{\leftrightarrow}}_> p_2$, which contradicts $p_1 \not\mathrel{\underline{\leftrightarrow}}_> p_2$. □

We are now ready to present our main theorem, which states that for n large enough, if p and q are summands of P_n that can be proved equivalent from a finite set of sound equations, and p is uniformly (n, Θ)-dependent, then q must also be uniformly (n, Θ)-dependent.

Theorem 3.3. *Assume that* Act *contains at least two distinct actions. Let* E *be a set of sound process equations of depth less than n, and let p and q be closed processes such that* $p, q \sqsubseteq_* P_n$ *and* $\mathsf{E} \vdash p \approx q$. *If p is uniformly* (n, Θ)*-dependent, then q is also uniformly* (n, Θ)*-dependent.*

Proof. As briefly discussed in Section 3.2, without loss of generality, we can disregard the symmetry rule in our inductive proof below by assuming that $u \approx t \in \mathsf{E}$ whenever $t \approx u \in \mathsf{E}$. Furthermore, we can assume that all applications of the substitution rule in derivations have a process equation from E as premise. This means that we only need to consider a new rule stating that all substitution instances of process equations in E are derivable, rather than considering the axiom rule — which states that all process equations in E are derivable —, and the substitution rule — which states that if a process equation is derivable, then so are all its substitution instances — separately.

We will now present the inductive argument over the number of steps in a proof of an equation $p \approx q$ from E. We proceed by a case analysis on the last rule applied to obtain $\mathsf{E} \vdash p \approx q$.

Case 1: reflexivity and transitivity. In these cases, the proof follows immediately or by the induction hypothesis in a straightforward manner.

<u>CASE 2: VARIABLE SUBSTITUTION.</u> Assume that $\mathsf{E} \vdash p \approx q$ is the result of a closed substitution instance of a process equation $t \approx u \in \mathsf{E}$, namely there exists a substitution σ such that $\sigma(t) = p$ and $\sigma(u) = q$. Since $t \approx u \in \mathsf{E}$, we have that $\mathrm{depth}\,(t), \mathrm{depth}\,(u) < n$. Moreover, from $p, q \sqsubseteq_* P_n$ it follows that $\mathrm{init}^\omega(p) = \mathrm{init}^\omega(q) = \{a\}$ and that, by Lemma 3.9, p and q are uniformly k-determinate for all $k \in \{1, \ldots n-1\}$. Hence by Proposition 3.4, we can conclude that if p is uniformly (n, Θ)-dependent, then so is q.

<u>CASE 3: CONGRUENCE RULE.</u> We can distinguish three cases:

- The last rule applied in $\mathsf{E} \vdash p \approx q$ is the congruence rule for the nondeterministic choice $+$. Then there exist closed process terms p_1, p_2, q_1 and q_2 such that $p = p_1 + p_2$, $q = q_1 + q_2$, $\mathsf{E} \vdash p_1 \approx q_1$ and $\mathsf{E} \vdash p_2 \approx q_2$ by shorter proofs. Since p is uniformly (n, Θ)-dependent, there must exist a process p' such that $p \rightarrow^n p'$, where p' is Θ-dependent and every process along the transitions from p to p' has norm 1.

  We can distinguish four possible subcases, regarding how this property is derived:

  1. p_1 is uniformly (n, Θ)-dependent.
  2. p_2 is uniformly (n, Θ)-dependent.
  3. $\mathrm{norm}\,(p_2) = 1$, $\mathrm{norm}\,(p_1) \neq 1$, and there are processes $p_1^1, \ldots, p_1^n$ such that $p_1 \rightarrow p_1^1 \rightarrow \ldots \rightarrow p_1^n = p'$ and p_1^n is Θ-dependent.
  4. $\mathrm{norm}\,(p_1) = 1$, $\mathrm{norm}\,(p_2) \neq 1$, and there are processes $p_2^1, \ldots, p_2^n$ such that $p_2 \rightarrow p_2^1 \rightarrow \ldots \rightarrow p_2^n = p'$ and p_2^n is Θ-dependent.

  In cases (1) and (2) we can immediately apply the induction hypothesis obtaining, respectively, that either q_1 or q_2 is uniformly (n, Θ)-dependent, and thus that q is uniformly (n, Θ)-dependent as well.

  The cases (3) and (4) require more attention. We detail only the proof for case (3), since the one for case (4) is symmetric. Firstly, we notice that since $p, q \sqsubseteq_* P_n$ then by Lemma 3.9 both p and q are uniformly k-determinate for all $k \in \{1, \ldots, n-1\}$. This implies that p_1 is uniformly k-determinate for the same values of k. Moreover, as $\mathsf{E} \vdash p_1 \approx q_1$ gives $p_1 \underline{\leftrightarrow}_* q_1$ and $\mathrm{depth}\,(p_1) = n$, by Lemma 3.7 we obtain that also q_1 is uniformly k-determinate for $k \in \{1, \ldots, n-1\}$. Then, by Proposition 3.3 we can infer that there is a process q_1^n such that $q_1 \rightarrow^n q_1^n$ and $q_1^n \underline{\leftrightarrow}_* p_1^n$, which, by Lemma 3.8, implies that q_1^n is Θ-dependent. Furthermore, uniform k-determinacy ensures that all the processes $q_1^1, \ldots, q_1^{n-1}$ in the sequence $q_1 \rightarrow q_1^1 \rightarrow \ldots \rightarrow q_1^{n-1} \rightarrow q_1^n$ have norm 1. Finally, we notice that since $\mathrm{norm}\,(p_2) = 1$ and $\mathsf{E} \vdash p_2 \approx q_2$ implies $p_2 \underline{\leftrightarrow}_* q_2$, we can infer that $\mathrm{norm}\,(q_2) = 1$. By combining the properties of q_1 and q_2, we can conclude that $q = q_1 + q_2$ is uniformly (n, Θ)-dependent.

- The last rule applied in $\mathsf{E} \vdash p \approx q$ is the congruence rule for the sequential composition. This means that $p = p_1 \cdot p_2$, $q = q_1 \cdot q_2$, $\mathsf{E} \vdash p_1 \approx q_1$ and $\mathsf{E} \vdash p_2 \approx q_2$ by shorter proofs. This case is vacuous, as $\mathrm{norm}\,(p) \geq 2$ and therefore p cannot be uniformly (n, Θ)-dependent.

$$\frac{x \xrightarrow{a}_{>} \surd\!\!\surd \quad \forall b > a \quad y \stackrel{b}{\not\rightarrow}_{>}}{x \lhd y \xrightarrow{a}_{>} \surd\!\!\surd} \qquad\qquad \frac{x \xrightarrow{a}_{>} x' \quad \forall b > a \quad y \stackrel{b}{\not\rightarrow}_{>}}{x \lhd y \xrightarrow{a}_{>} x'}$$

U1	$a \lhd b \approx a$ if not $b > a$	U4	$(x \cdot y) \lhd z \approx (x \lhd z) \cdot y$
U2	$x \lhd (y \cdot z) \approx x \lhd y$	U5	$(x + y) \lhd z \approx x \lhd z + y \lhd z$
U3	$x \lhd (y + z) \approx (x \lhd y) \lhd z$	PU	$\Theta(x + y) \approx \Theta(x) \lhd y + \Theta(y) \lhd x$

Table 3.5: Operational semantics and some axioms of the unless operator.

- The last rule applied in $\mathsf{E} \vdash p \approx q$ is the congruence rule for the priority operator Θ. Then there exist p' and q' such that $p = \Theta(p')$, $q = \Theta(q')$, and $\mathsf{E} \vdash p' \approx q'$ by a shorter proof. Since p is uniformly (n, Θ)-dependent, there exists a sequence of processes $p = \Theta(p') \rightarrow \Theta(p_1) \rightarrow \cdots \rightarrow \Theta(p_{n-1}) \rightarrow \Theta(p_n)$ such that $\text{norm}\,(\Theta(p_1)) = \ldots \text{norm}\,(\Theta(p_{n-1})) = 1$ and $\Theta(p_n)$ is Θ-dependent. Note that, since $\Theta(p_n)$ is Θ-dependent, $|\text{init}(p_n)| \geq 2$. Moreover, from the operational rules for Θ, $p' \rightarrow p_1 \rightarrow \cdots \rightarrow p_{n-1} \rightarrow p_n$ and from the definition of norm, $\text{norm}\,(p_1) = \cdots = \text{norm}\,(p_n) = 1$. From $\mathsf{E} \vdash p' \approx q'$, we derive that $p' \underline{\leftrightarrow}_* q'$. Hence, $p' \underline{\leftrightarrow} q'$ holds and therefore we get a sequence $q' \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $p_n \underline{\leftrightarrow} q_n$, which implies that $|\text{init}(q_n)| \geq 2$. Thus, we infer $q = \Theta(q') \rightarrow \Theta(q_1) \rightarrow \cdots \rightarrow \Theta(q_n)$ and, since $|\text{init}(q_n)| \geq 2$, $\Theta(q_n)$ is Θ-dependent. It remains to show that $\text{norm}\,(\Theta(q')) = \text{norm}\,(\Theta(q_i)) = 1$ for each $i \in \{1, \ldots, n-1\}$. As $q \sqsubseteq_* P_n$, by Lemma 3.9 we gather that q is uniformly k-determinate for all $1 \leq k < n$, from which it follows that $\text{norm}\,(\Theta(q_i)) = 1$ for all $i \in \{1, \ldots, n-1\}$. Since, moreover, $p \underline{\leftrightarrow}_* q$ and $\text{norm}\,(p) = 1$, we get $\text{norm}\,(q) = 1$ and we conclude that q is (n, Θ)-dependent. □

As the left-hand side of the equations in (3.1) is uniformly (n, Θ)-dependent while the right-hand side is not, from Theorem 3.3 we can directly infer that for each n, the nth instance of the family of equations in (3.1) cannot be proved using the finite collection of all sound equations whose depth is smaller than n.

We can therefore conclude that Theorem 3.1 (presented in Section 3.3) holds.

3.8 On the Use of Auxiliary Operators

In its first appearance, in [40], the priority operator was defined in terms of the simpler binary operator *unless*, denoted by $\lhd$. Informally, $\lhd$ allows us to capture the priority order among actions, as $a \lhd b$ behaves like a unless b has higher priority than a. In Table 3.5 we report the SOS rules defining the behavior of the unless operator, together with some valid axioms for it. In particular, axiom (PU) allows us to rewrite the behavior of the priority operator in terms of that of unless.

***Example* 6.** *Consider process $p = a \cdot (b \lhd c + c \lhd b)$. If $b > c$, then only the summand $b \lhd c$ of $(b \lhd c + c \lhd b)$ can make a transition, thus giving $p \underline{\leftrightarrow}_> a \cdot b$. Similarly, if $c > b$ then $p \underline{\leftrightarrow}_> a \cdot c$. In case b and c are incomparable with respect to $>$, then $b \lhd c + c \lhd b \underline{\leftrightarrow}_> b + c$, so that $p \underline{\leftrightarrow}_> a \cdot (b + c)$.*

Consider now processes $q_1 = a \lhd (b \cdot c)$ and $q_2 = (a \cdot b) \lhd c$. The unless operator compares only the initial actions of its arguments (cf. axioms (U2) and (U4) in Table 3.5). Hence in q_1 the priority order between a and c plays no role in determining whether q_1 will perform the a-move or not. At the same time, if c has higher priority than a, in q_2 also the execution of b is prevented disregarding the ordering of b and c.

One can prove, in a similar fashion to [40], that, provided the set of actions is finite, for a chosen priority order $>$, the bisimulation equivalence $\underline{\leftrightarrow}_>$ affords a finite axiomatization over $\mathrm{BPA}_{\Theta,\lhd}$, namely BPA enriched with both Θ and $\lhd$. Hence a natural question that arises is whether we can regain a finite axiomatization over $\mathrm{BPA}_{\Theta,\lhd}$ also for order-insensitive bisimilarity. We devote this section to proving that a negative answer applies and thus that the following theorem holds:

Theorem 3.4. *If the set of actions* Act *contains at least two distinct actions, then the language* $\mathrm{BPA}_{\Theta,\lhd}$ *modulo order-insensitive bisimilarity is not finitely based.*

Since the technical development of the negative result for BPA_{Θ} (Theorem 3.1) would apply in major part unchanged in the proof of Theorem 3.4, we actually present only an informal discussion of this result.

Consider the family of equations in (3.1), that we used to prove the negative result for the priority operator. One can prove, by using axioms (PU) in Table 3.5 and (P5) in Table 3.3 together with congruence closure, that

$$A_n(\Theta(a + b)) \approx A_n(a \lhd b + b \lhd a)$$

and thus that the family of equations

$$e'_n : \quad P_n + A_n(a \lhd b + b \lhd a) \approx P_n \qquad (n \geq 0) \tag{3.2}$$

is sound modulo order-insensitive bisimilarity. However, precisely because we are considering the order-insensitive relation, one can notice that it is not possible to eliminate the occurrences of the unless operator from the left-hand side of the equations in (3.2). In fact, as no priority order over actions has been chosen, it is not possible to establish the relation between actions a and b (that we recall are assumed to be distinct) and thus whether $\lhd$ will allow for their execution or not. More formally, we notice that the axiom (U1) in Table 3.5 is not sound modulo order-insensitive bisimilarity (with the only exception of the trivial case in which the actions in the two sides of $\lhd$ coincide). Therefore, the same reasoning applied to prove Theorem 3.3, and thus Theorem 3.1, can be adapted in a straightforward manner to obtain a proof for Theorem 3.4. Intuitively, we simply need to substitute the notions of Θ-dependency and uniform (n, Θ)-dependency with the corresponding notions for the unless operator.

3.9 Complexity of Order-Insensitive Bisimilarity Checking

In this section we investigate some algorithms, and their complexity, for checking order-insensitive bisimilarity of (loop-free) finite labeled transition systems. It is known that bisimilarity over such systems is poly-complete [41], and, moreover, using the Paige-Tarjan algorithm [163] each $\underline{\leftrightarrow}_>$ can be checked in $O(m_t \log m_s)$, where m_t is the number of transitions, and m_s is the number of states. A naive algorithm for $\underline{\leftrightarrow}_*$ would then check $\underline{\leftrightarrow}_>$ for all the possible partial orders $>$ over $\textsc{Act}$. Assuming that $|\textsc{Act}| = k > 0$, there are $2^{k^2/4+3k/4+O(\log k)}$ possible partial orders (see [138] for the result on the number of posets over sets with k elements). Clearly, from these results we can obtain an upper bound on the complexity of $\underline{\leftrightarrow}_*$.

Theorem 3.5. *The problem of deciding whether two processes are order-insensitive bisimilar is in* coNP *and can be solved in time* $2^{k^2/4+3k/4+O(\log k)} \cdot O(n^2)$ *where k is the number of actions and n is the sum of the sizes of the two processes.*

Proof. Let $|p|$ denote the size of process p. We first argue that the complexity of the naive algorithm for checking whether two closed BPA$_\Theta$ terms p and q are related by order-insensitive bisimilarity is

$$2^{k^2/4+3k/4+O(\log k)} \cdot O(n^2) \ ,$$

where $n = |p| + |q|$ is the sum of the sizes of the two processes. To this end, observe that, for each irreflexive partial order $>$ over $\textsc{Act}$, the algorithm checks whether $p \underline{\leftrightarrow}_> q$ holds, which can be done by verifying that the loop-free LTSs with transition relation $\rightarrow_>$ associated with p and q are bisimilar. The latter check can be done in $O(m_t \log m_s)$ using the Paige-Tarjan algorithm. It is not hard to verify that the number m_s of states and the number m_t of transitions in the LTS associated with a closed BPA$_\Theta$ term are linear in the size of the term. Moreover such an LTS can be constructed in time $O(|p|^2)$ from a term p and a priority order $>$. So checking whether p and q are related by $\underline{\leftrightarrow}_>$ can be done in time $O(n^2 + n \log n) = O(n^2)$, where n is the sum of the sizes of p and q. It follows that the naive algorithm has complexity $2^{k^2/4+3k/4+O(\log k)} \cdot O(n^2)$.

We now argue that order-insensitive bisimilarity checking is in coNP. Given two terms p and q that are not order-insensitive bisimilar, one can nondeterministically guess an irreflexive partial order $>$ that separates them, generate the loop-free LTSs with transition relation $\rightarrow_>$ associated with p and q (which can be done in quadratic time), and then verify the correctness of this guess with the Paige-Tarjan algorithm that checks for bisimilarity of the LTSs. Guessing an irreflexive partial over k elements can be done by:

- Guessing an irreflexive relation in time $O(k^2)$;
- Computing its transitive closure in cubic time;
- Checking whether the resulting relation is acyclic in time that is linear in the size of the resulting directed graph.

The coNP bound follows from the above mentioned observations. □

Remark 2. *If* Act *is a singleton, the complexity bounds in Theorem 3.5 can be sharpened. Indeed, in that case,* $\underline{\leftrightarrow}_*$ *coincides with bisimilarity and checking whether two loop-free LTSs over a singleton action set are bisimilar is* poly*-complete [41].*

The main contributor to the complexity of the above-mentioned naive algorithm however is the number of bisimilarity checks that has to be performed. Indeed, when verifying the order-insensitive bisimilarity of two BPA_Θ terms, the only upper bound we can impose on the number of actions appearing in the terms is linear in the size of the terms in the worst case. Therefore the number of possible partial orders that have to be considered is exponential in size of the input terms. It might be possible to improve on the number of the partial orders to consider if we could exclude a priori the checking of some significant number of partial orders. For instance, one could hope that $p \underline{\leftrightarrow}_{>_0} q$ does not need to be checked if $p \underline{\leftrightarrow}_{>_1} q$ for some $>_1$ that extends $>_0$. We dedicate the remainder of this section to showing that this is impossible in general.

Assume that Act is finite and $|\textsc{Act}| = k > 0$. Let $>_0$ be an irreflexive partial order over Act. Our goal is to construct two BPA_Θ terms p and q with the following properties:

(a) $p \not\underline{\leftrightarrow}_{>_0} q$, and

(b) $p \underline{\leftrightarrow}_{>} q$ for each irreflexive partial order $> \neq >_0$.

We introduce next some constructions and notation that will be useful in what follows.

First of all, for each non-empty $S \subseteq \textsc{Act}$, we define the term $v(S)$ thus:

$$v(S) = \begin{cases} a & \text{if } S = \{a\} \text{ for some } a \in \textsc{Act} \\ \sum_{a \in S} a.v(S \setminus \{a\}) & \text{otherwise.} \end{cases}$$

Intuitively, $v(S)$ describes a nondeterministic process that can perform all permutations of the actions in S.

Given an irreflexive partial order $>$ over Act, we let $p_>$ denote a closed BPA_Θ term such that $p_>$ contains no occurrences of Θ and

$$p_> \underline{\leftrightarrow}_> \Theta(v(\textsc{Act})). \tag{3.3}$$

***Example* 7.** *Assume that* $\textsc{Act} = \{a, b\}$ *and let* $>_0 = \emptyset$. *There are only two other irreflexive partial orders over* $\{a, b\}$, *namely* $>_1 = \{(a, b)\}$ *and* $>_2 = \{(b, a)\}$. *Now consider the term*

$$v = v(\{a, b\}) = ab + ba \ .$$

It is easy to see that

- $\Theta(v) \underline{\leftrightarrow}_{>_0} v$,
- $\Theta(v) \underline{\leftrightarrow}_{>_1} ab = p_{>_1}$, *and*

- $\Theta(v) \underline{\leftrightarrow}_{>_2} ba = p_{>_2}$.

Consider now processes $p = a.p_{>_1} + a.p_{>_2}$ and $q = p + a.\Theta(a.b + b.a)$. From the above, it follows immediately that $p \underline{\leftrightarrow}_{>_1} q$ and $p \underline{\leftrightarrow}_{>_2} q$. However, we have that $p \not\underline{\leftrightarrow}_{>_0} q$. Indeed, $\Theta(v) \underline{\leftrightarrow}_{>_0} v$ and thus q can do an a action and become $a.b + b.a$ while p cannot match that transition.

As highlighted by the above example, the traces of the term $\Theta(v(\textsc{Act}))$ with respect to $\rightarrow_>$ are all the *linearizations* of the partial order $>$. A classic result in order theory states that a partial order is uniquely determined by its linear extension [189]. This is the key to the following lemma.

Lemma 3.10. *Two closed process terms $p_{>_1}$ and $p_{>_2}$ defined as in Equation* (3.3) *above have the same traces if and only if $>_1=>_2$.*

Using the above lemma, we can now prove that:

Theorem 3.6. *Assume that* Act *is finite and contains at least two distinct actions. Let $>_0$ be an irreflexive partial order over* Act. *Then there exist closed* BPA$_\Theta$ *terms p and q such that, for each irreflexive partial order $>$ over* Act, *$p \underline{\leftrightarrow}_> q$ if and only if $> \neq >_0$.*

Proof. We need to exhibit two closed BPA$_\Theta$ terms satisfying the above-mentioned properties ((a)) and ((b)) with respect to the chosen partial order $>_0$. To this end, we choose an action $a \in \textsc{Act}$ and define:

$$p = \sum_{>\in PO(\textsc{Act}),\ >\neq>_0} a.p_> \quad \text{and} \quad q = p + a\Theta(v(\textsc{Act})) \qquad (3.4)$$

where $PO(\textsc{Act})$ denotes the set of all irreflexive partial orders on Act.

- p and q satisfy property ((b)).

  We need to show that $p \underline{\leftrightarrow}_> q$ for each $>\in PO(\textsc{Act})$ such that $>\neq>_0$. This follows by construction. In fact, for each $>\neq>_0$, both processes contain a summand bisimilar to the closed term $a.p_>$ and moreover $a.\Theta(v(\textsc{Act})) \underline{\leftrightarrow}_> a.p_>$.

- p and q satisfy property ((a)).

  We need to show that $p \not\underline{\leftrightarrow}_{>_0} q$. To see this, observe that $q \xrightarrow{a}_{>_0} \Theta(v(\textsc{Act})) \underline{\leftrightarrow}_{>_0} p_{>_0}$. On the other hand, if $p \xrightarrow{a}_{>_0} p'$ then $p' = p_> \underline{\leftrightarrow}_> \Theta(v(\textsc{Act}))$ for some partial order $>\neq>_0$. By Lemma 3.10, $p_>$ does not have the same traces as $p_{>_0}$, and thus $p_{>_0} \not\underline{\leftrightarrow}_{>_0} p_>$. This means that p cannot match the transition $q \xrightarrow{a}_{>_0} \Theta(v(\textsc{Act}))$ up to $\underline{\leftrightarrow}_{>_0}$ and thus $p \not\underline{\leftrightarrow}_{>_0} q$. □

3.10 Conclusions

In this chapter we have studied the finite axiomatizability of the equational theory of order-insensitive bisimilarity over the language BPA enriched with the priority operator Θ. As previous similar work suggested, also in this setting, the collection of sound, closed equations is not finitely based in the presence of at least two actions, despite the fact that the sequential composition operator allows one to write more complex axioms than action prefixing. We proved this negative result using an infinite family of closed equations suggested in [13] and showing that no set of sound equations of bounded depth can derive them all.

Finding an infinite (ground-)complete axiomatization of order-insensitive bisimilarity is a natural avenue for future research. It would also be interesting to see whether we can obtain a lower bound on the complexity of order-insensitive bisimilarity checking. Above we discussed various upper bounds for its complexity that all suggest some type of computational hardness and since we have that the problem is in coNP it would be a natural follow-up to prove coNP-hardness. At the time of writing, this hardness result is not obvious to us.

Chapter 4

Axiomatizing recursion free, regular monitors

4.1 Introduction

In this chapter, we study the equational theory of the *monitors* studied by Aceto et al. in, for instance, [14, 15, 104]. Monitors are a key tool in the field of runtime verification (see [47, 96, 120, 121, 147, 170, 187, 190] and the references therein for an overview of this active research area), where they are used to check for system properties by analyzing execution traces generated by processes and are often expressed using some automata-based formalism. The notion of monitorable property has been defined in a seminal paper by Pnueli and Zaks [170]. Intuitively, a property of finite and infinite system executions is s-monitorable, for some finite trace of observable events s, if there is an extension of s after which a monitor will be able to determine conclusively whether the observed system execution satisfies or violates the property. This means that verdicts issued by monitors are *irrevocable.* In that work by Pnueli and Zaks, a property is described by the set of finite and infinite executions that satisfy it. However, in the theory and practice of runtime verification, one often specifies properties finitely using formalisms such as automata or (variations on) temporal logics and studies what specifications in the chosen formalism are 'monitorable' and with what correctness guarantees—see, for instance, [45, 50, 176]. Since monitors are part of the trusted computing base, the automated, correct-by-design *monitor synthesis* from the formal specification of properties has been thoroughly studied in the literature and is often accompanied by the experimental evaluation of the overhead induced by monitoring—see, for example, the study of various approaches to the automated monitor synthesis for systemC specifications given in [190] and the framework for benchmarking of runtime verification tools presented in [24].

In [14, 15, 104], Aceto et al. specified monitors using a variation on the regular fragment of Milner's CCS [153] and studied two trace-based notions of equivalence over monitors, namely verdict and ω-verdict equivalence. Intuitively, two monitor descriptions are verdict equivalent when they accept and reject the same finite execution traces of the systems they observe. The notion of ω-verdict equivalence is the 'asymptotic version' of verdict equivalence, in that it is solely concerned with the infinite traces that are accepted and rejected by monitors. In their work, Aceto et. al. focus on determining the 'monitorable' fragment of Hennessy-Milner Logic with recursion [14, 104] and provide monitor-synthesis algorithms for properties that can be expressed in that fragment. The key (and non-negotiable) property that the monitor synthesized from a formula φ in the monitorable fragment of that logic should satisfy is *soundness*, which means that a verdict issued by the monitor as it examines a system execution determines whether that execution satisfies φ or not correctly. Naturally, sound monitors cannot produce contradictory verdicts for a given trace.

Our contribution When monitors are described by expressions in some monitor-specification language, such as the one employed by Aceto et al. in *op. cit.*, it is natural to ask oneself whether one can (finitely) axiomatize notions of monitor equivalence over (fragments of) that language. This study is devoted to addressing that question in the simplest non-trivial setting. In particular, in order to stay within the realm of classic equational logic over total algebras, we consider a language that allows one to specify unsound monitors. However, all the results we present in the chapter specialize to sub-languages consisting of (sound) monitors that can only issue either positive or negative verdicts.

The main results we present in this chapter are complete equational characterizations of verdict equivalence over both closed (that is, variable-free) and open, recursion-free regular monitors. More specifically, we first provide an equational axiomatization of verdict equivalence over closed terms from the language of monitors we study that is finite if so is the set of actions monitors can observe (Theorem 4.2). The landscape of axiomatizability results for verdict equivalence over open terms turns out to be more varied. This variety is witnessed by the fact that there are three different axiomatizations, depending on whether the set of actions is infinite (Theorem 4.4), finite and containing at least two actions (Theorem 4.5) or a singleton (Theorem 4.6). Only the axiomatization given in Theorem 4.6 is finite and we show that this is unavoidable. Indeed, verdict equivalence has no finite equational basis when the set of actions is finite and of cardinality at least two (Theorem 4.10).

It turns out that the above-mentioned axiomatizations are also complete for ω-verdict equivalence if the set of actions that monitors may observe is infinite, as in that case the two notions of equivalence coincide. On the other hand, if the set of actions is finite, ω-verdict equivalence is strictly coarser than verdict equivalence. We also provide a finite, complete axiomatization of ω-verdict equivalence for closed monitors in the setting of a finite set of actions (Theorem 4.3). Our Theorem 4.8 gives a complete axiomatization of ω-verdict equivalence over open monitors when the set of actions contains at least two actions. If the set of actions is a singleton, ω-verdict equivalence has a finite equational basis (Theorem 4.7).

The equational axiomatizations we present in this chapter capture the 'laws of monitor programming' [130] for an admittedly rather inexpressive language. Indeed, recursion-free regular monitors describe essentially tree-like finite-state automata with distinguished accept and reject states at their 'leaves' with self-loops labeled by every action. (See the operational semantics of monitors in Table 1. Note, however, that those automata may have infinitely many transitions, if the set of actions monitors can observe is infinite. As shown already by Milner in his classic books on CCS [153, 152], this feature is useful when modeling system events that carry data values. See, for instance, the paper [44] for one of the earliest attempts to incorporate data into runtime verification.) However, as witnessed by our results and their proofs, the study of the equational theory of monitors modulo the notions of equivalence we consider is non-trivial even for the minimal language studied in this chapter. In our, admittedly biased, opinion, it is therefore worthwhile to map the territory of axiomatizability results for recursion-free regular monitors, since results for more expressive languages will have to build upon those we obtain in this chapter. We remark, in passing, that the non-finite axiomatizability result in Theorem 4.10 is obtained over a substantially more restrictive syntax than classic negative results for the algebra of regular expressions, which rely on the hardness of expressing the interplay between Kleene star and concatenation equationally [3, 78, 177].

The contribution of this chapter is entirely theoretical and we make no claims pertaining to the applicability of our current results in the practice of runtime verification. However, apart from their intrinsic theoretical interest, (extensions of) the equational axiomatizations we present might be used in the automatic, syntax-driven synthesis of monitors from specifications of 'monitorable properties', as presented in [14, 15, 103], to rewrite monitor expressions in an 'equivalent, but simpler' syntactic form, for instance by eliminating 'redundant' sub-expressions. As witnessed by the study of optimized temporal monitors for SystemC presented in [190], the investigation of monitor optimizations based on equational rewriting or other techniques requires a substantial experimental research effort and is outside the scope of this project. We discuss other avenues for future research in Section 4.6.

4.2 Preliminaries

We begin by introducing recursion-free regular monitors (or simply monitors in this study) and the two notions of verdict equivalence that we study in this chapter. We refer the interested reader to [14, 104] for background motivation and more information.

Syntax of monitors Let $\textsc{Act}$ be a set of visible actions, ranged over by a, b. Following Milner [152], we use $\tau \notin \textsc{Act}$ to denote an unobservable action. The symbol α ranges over $\textsc{Act} \cup \{\tau\}$. Let $\mathcal{V}$ be a countably infinite set of variables, ranged over by x, y, z. We assume that $\textsc{Act} \cup \{\tau\}$ and $\mathcal{V}$ are disjoint.

We write $\textsc{Act}^\omega$ for the set of infinite sequences over $\textsc{Act}$. As usual, $\textsc{Act}^*$ stands for the set of finite sequences over $\textsc{Act}$. Let A be a set of finite sequences and B be a set of sequences. We write $A \cdot B$ for the concatenation of A and B.

The collection Mon_F of (regular, recursion-free) monitors is the set of terms generated by the following grammar:

$$
\begin{aligned}
m, n &::= \ v \ \mid \ a.m \ \mid \ m + n \ \mid \ x \\
v &::= \ \mathit{end} \ \mid \ \mathit{yes} \ \mid \ \mathit{no}
\end{aligned}
$$

where $a \in \textsc{Act}$ and $x \in \mathcal{V}$. The terms *end*, *yes* and *no* are called *verdicts*. Intuitively, *yes* stands for the acceptance verdict, *no* denotes a rejection verdict and *end* is the inconclusive verdict, namely the state a monitor reaches when, based on the sequence of observations it has processed so far, it realizes that it will not be able to issue an acceptance or rejection verdict in the future. As will be formalized by the operational semantics of monitors to follow, verdicts are irrevocable. This means that once a monitor reaches a verdict, it will stick to it regardless of what further observations it makes. See, for instance, [14, 47, 104] for a detailed technical discussion.

Intuitively, a monitor of the form $a.m$ can observe action a and behave like m thereafter. On the other hand, a monitor of the form $m + n$ can behave either like m or like n.

Remark 3. *The work on which we build in this chapter considers a setting with three verdicts, two of which are 'conclusive.' There are a number of other approaches in the field of runtime verification that consider many-valued verdicts. We refer the interested reader to, for instance, [46, 45, 52, 65, 95] for further information.*

Closed monitors are those that do not contain any occurrences of variables. A (closed) *substitution* is a mapping σ from variables to (closed) monitors. We write $\sigma(m)$ for the monitor that results when applying the substitution σ to m. Note that $\sigma(m)$ is closed, if σ is a closed substitution.

Definition 4.1 (Notation). *We use $m\ [+v]$ for a verdict v to indicate that v is an optional summand of m, that is, that the term can be either m or $m + v$. In addition a monitor will be called v-free for a verdict v, when it does not contain any occurrences of v.*

For a finite index set $I = \{i_1, \ldots, i_k\}$ and indexed set of monitors $\{m_i\}_{i \in I}$, we write $\sum_{i \in I} m_i$ to stand for end if $I = \emptyset$ and for $m_{i_1} + \ldots + m_{i_k}$ otherwise. This notation is justified by the fact that $+$ is associative and commutative, and has end as a neutral element, in all of the semantics we use in this chapter.

We now associate a notion of syntactic depth with each monitor. Intuitively, the decision a monitor m takes when reading a string $s \in \textsc{Act}^*$ only depends on the prefixes of s whose length is at most the syntactic depth of m.

Definition 4.2 (Syntactic Depth). *For any closed monitor $m \in Mon_F$, we define* $\mathrm{depth}(m)$ *as follows:*

- $\mathrm{depth}(a.m) = 1 + \mathrm{depth}(m)$,
- $\mathrm{depth}(m_1 + m_2) = \max(\mathrm{depth}(m_1), \mathrm{depth}(m_2))$ *and*
- $\mathrm{depth}(v) = 0$ *for a verdict v.*

$$\frac{}{a.m \xrightarrow{a} m} \qquad \frac{m \xrightarrow{\alpha} m'}{m+n \xrightarrow{\alpha} m'} \qquad \frac{n \xrightarrow{\alpha} n'}{m+n \xrightarrow{\alpha} n'} \qquad \frac{}{v \xrightarrow{\alpha} v}$$

Table 4.1: Operational semantics of processes in Mon_F.

Semantics of monitors For each $\alpha \in \textsc{Act} \cup \{\tau\}$, we define the transition relation $\xrightarrow{\alpha} \subseteq Mon_F \times Mon_F$ as the least one that satisfies the rules in Table 4.1.

For example, $yes + x \xrightarrow{\tau} yes$ and $a.yes + end \xrightarrow{b} end$, for each $a, b \in \textsc{Act}$. A useful fact based on the above operational semantics is that if $m \xrightarrow{\tau} m'$, then $m' = v$ for some verdict v.

Note that variables have no transitions. They represent under-specification in monitor behavior. For instance, monitor $a.yes{+}x$ is one that we know can reach the verdict *yes* after having observed an a action. Further information on the behavior of that monitor can only be gleaned once the variable x has been instantiated via a (closed) substitution.

For m, m' in Mon_F and $s = a_1 \ldots a_k$ in $\textsc{Act}^*$, $k \geq 0$, $m \xrightarrow{s} m'$ holds iff there are $m_0, , \ldots, m_k$ such that

$$m = m_0 \xrightarrow{a_1} m_1 \cdots m_{k-1} \xrightarrow{a_k} m_k = m'.$$

Additionally, for $s \in \textsc{Act}^*$, we use $m \overset{s}{\Rightarrow} m'$ to mean that:

1. $m\ (\xrightarrow{\tau})^*\ m'$ if $s = \varepsilon$, where ε stands for the empty string,

2. $m \overset{\varepsilon}{\Rightarrow} m_1 \xrightarrow{a} m_2 \overset{\varepsilon}{\Rightarrow} m'$ for some m_1, m_2 if $s = a \in \textsc{Act}$ and

3. $m \overset{a}{\Rightarrow} m_1 \overset{s'}{\Rightarrow} m'$ for some m_1 if $s = a.s'$, for some $s' \neq \varepsilon$.

If $m \overset{s}{\Rightarrow} m'$ for some m', we call s a *trace* of m.

Lemma 4.1. *For all $s \in \textsc{Act}^*$, $m, n \in Mon_F$, and verdict v, $m{+}n \overset{s}{\Rightarrow} v$ iff $m \overset{s}{\Rightarrow} v$ or $n \overset{s}{\Rightarrow} v$.*

Proof. We prove both implications separately, by induction on the length of s. The details are straightforward and are therefore omitted. Here we limit ourselves to remarking that, in the proof of the implication from right to left, if $s = \varepsilon$ and $m = v$, say, then $v + n \xrightarrow{\tau} v$ by the rules in Table 4.1. □

Remark 4. *Note that the implication from right to left in Lemma 4.1 would not hold in the absence of rule $v \xrightarrow{\tau} v$ in Table 4.1.*

Verdict and ω-verdict equivalence Let m be a (closed) monitor. We define:

$$L_a(m) = \{s \in \textsc{Act}^* \mid m \overset{s}{\Rightarrow} yes\} \text{ and}$$
$$L_r(m) = \{s \in \textsc{Act}^* \mid m \overset{s}{\Rightarrow} no\}.$$

Intuitively, $L_a(m)$ denotes the set of traces that are accepted by m, whereas $L_r(m)$ stands for the set of traces that m rejects. The sets $L_a(m)$ and $L_r(m)$ will also be referred to as the acceptance and rejection set of m respectively. Note that we allow for monitors that may both accept and reject the same trace. This is necessary to maintain our monitors closed under $+$ and to work with classic total algebras rather than partial ones. Of course, in practice, one is interested in monitors that are consistent in their verdicts. One way to ensure consistency in monitor verdicts, which was considered in [104], is to restrict oneself to monitors that use only one of the conclusive verdicts *yes* and *no*. All the results that we present in the remainder of this chapter apply to such monitors.

Remark 5. *One might wonder about the connection between the languages that are accepted/rejected by recursion-free regular monitors and star-free languages [183]. A simple argument by induction on the structure of monitors shows that every recursion-free regular monitor denotes a pair of star-free languages, one for its acceptance set and one for its rejection set. Moreover, this means that recursion-free regular monitors correspond to properties that can be expressed in LTL [134]. However, there are star-free languages (and therefore LTL properties) that cannot be described by recursion-free regular monitors. For example, the language $(ab)^*$ is star-free (see, for instance, [87, page 267]) but does not correspond to any recursion-free regular monitor.*

The monitors we consider in this chapter output a positive or negative verdict after a finite number of computational steps, if they do so at all. This means that the linear-time temporal properties to which their acceptance and rejection set correspond are both 'Always Finitely Refutable' and 'Always Finitely Satisfiable' in the sense of [167], as proven in [14].

Definition 4.3. *Let m and n be closed monitors.*

- *We say that m and n are **verdict equivalent**, written $m \simeq n$, if $L_a(m) = L_a(n)$ and $L_r(m) = L_r(n)$.*
- *We say that m and n are ω**-verdict equivalent**, written $m \simeq_\omega n$, if $L_a(m) \cdot \textsc{Act}^\omega = L_a(n) \cdot \textsc{Act}^\omega$ and $L_r(m) \cdot \textsc{Act}^\omega = L_r(n) \cdot \textsc{Act}^\omega$.*

For open monitors m and n, we say that $m \simeq n$ if $\sigma(m) \simeq \sigma(n)$, for all closed substitutions σ. The relation $\simeq_\omega$ is extended to open monitors in similar fashion.

***Example* 8.** *It is easy to see that $m + end \simeq m$ holds for each $m \in Mon_F$. Moreover, since $L_a(end) = \emptyset$ and $L_r(end) = \emptyset$, $a.end \simeq end$ holds for each $a \in \textsc{Act}$.*

One can intuitively see that the notion of ω-verdict equivalence refers to a form of asymptotic behavior. Indeed, monitors m and n are ω-verdict equivalent if, and only if, they accept and reject the same infinite traces in the sense of [14]. Next we provide a lemma that clarifies the relations between the two notions of equivalence defined above.

Lemma 4.2. *The following statements hold:*

- $\simeq$ *and* $\simeq_\omega$ *are both congruences.*
- $\simeq \subseteq \simeq_\omega$ *and the inclusion is strict when* $\textsc{Act}$ *is finite.*
- *If* $\textsc{Act}$ *is infinite then* $\simeq = \simeq_\omega$.

Proof. For the first claim, it suffices to prove that $\simeq$ and $\simeq_\omega$ are equivalence relations and that they are preserved by $a.\_$ and $+$. The proof is standard and is thus omitted.

For the second claim, the inclusion $\simeq \subseteq \simeq_\omega$ is easy to check using the definitions of the two relations. The fact that the inclusion is strict when the set of actions is finite follows from the validity of the equivalence $yes \simeq_\omega \sum_{a \in \textsc{Act}} a.yes$.

However, that equivalence is not valid modulo verdict equivalence since the first monitor accepts the empty string ε, but $\sum_{a \in \textsc{Act}} a.yes$ cannot.

Finally, suppose that $\textsc{Act}$ is infinite. Assume that m and n are ω-verdict equivalent and that s is a finite trace accepted by m. We will argue that n also accepts s. To this end, note that, since $\textsc{Act}$ is infinite, there is some action a that does not occur in m and n. Since m accepts s, the infinite trace sa^ω is in $L_a(m) \cdot \textsc{Act}^\omega$. By the assumption that m and n are ω-verdict equivalent, we have that sa^ω is in $L_a(n) \cdot \textsc{Act}^\omega$. As a does not occur in n, it is not hard to see that n accepts s. Therefore, by symmetry, m and n accept the same traces. The same argument shows that $L_r(m) = L_r(n)$, and therefore $m \simeq n$. □

Equational logic An axiom system $\mathcal{E}$ over Mon_F is a collection of equations $m = n$ expressed in the syntax of Mon_F. An equation $m = n$ is derivable from an axiom system $\mathcal{E}$ (notation $\mathcal{E} \vdash m = n$) if it can be proven from the axioms in $\mathcal{E}$ using the rules of equational logic (reflexivity, symmetry, transitivity, substitution and closure under the Mon_F contexts). See Table 4.2. In the rest of this work we shall always implicitly assume, without loss of generality, that equational axiom systems are closed with respect to symmetry, i.e., that if $m = n$ is an axiom, so is $n = m$.

We say that $\mathcal{E}$ is *sound* with respect to $\simeq$ when $m \simeq n$ holds whenever $\mathcal{E} \vdash m = n$. We say that $\mathcal{E}$ is *complete* with respect to $\simeq$ when $\mathcal{E}$ can prove all the valid equations $m \simeq n$. Similar definitions apply for ω-verdict equivalence. The notion of completeness, when limited to closed terms, is referred to as *ground completeness.*

$$
(\textsc{Ref})\ m \approx m \qquad (\textsc{Sym})\ \frac{m \approx n}{n \approx m} \qquad (\textsc{Tran})\ \frac{m \approx n \quad n \approx u}{m \approx u}
$$

$$
(\textsc{Subs})\ \frac{m \approx n}{\sigma(m) \approx \sigma(n)} \qquad (\textsc{Pref})\ \frac{m \approx n}{a.m \approx a.n} \qquad (\textsc{Ch})\ \frac{m \approx m' \quad n \approx n'}{m + n \approx m' + n'}
$$

Table 4.2: Equational Laws, over Mon_F.

4.3 A Ground-Complete Axiomatization of Verdict and ω-Verdict Equivalence

Our goal in this chapter is to study the equational theory of $\simeq$ and $\simeq_\omega$ over Mon_F. Our first main result is to give a ground-complete axiomatization of verdict equivalence over Mon_F. To this end, consider the axiom system $\mathcal{E}_v$, whose axioms are listed in Table 4.3.

(A1) $x + y = y + x$	($\mathbf{E_a}$) $a.end = end$ ($a \in \textsc{Act}$)
(A2) $x + (y + z) = (x + y) + z$	($\mathbf{Y_a}$) $yes = yes + a.yes$ ($a \in \textsc{Act}$)
(A3) $x + x = x$	($\mathbf{N_a}$) $no = no + a.no$ ($a \in \textsc{Act}$)
(A4) $x + end = x$	($\mathbf{D_a}$) $a.(x + y) = a.x + a.y$ ($a \in \textsc{Act}$)

Table 4.3: The axioms of $\mathcal{E}_v$

.

Remark 6. *Note that $\mathcal{E}_v$ is finite, if so is* Act.

The subscript $_v$ in the naming scheme of the axiom set refers to the kind of equivalence that it axiomatizes, namely verdict equivalence. It will later be replaced with $_\omega$ when we study ω-verdict equivalence and used accordingly from that point forward.

We provide now the following lemma as an observation on the number of necessary axioms when Act is finite and as an example proof based on these axioms.

Lemma 4.3. *When* Act *is finite, the family of axioms (Y_a) can be replaced with*

$$
(\mathbf{Y})\ yes = yes + \sum_{a \in \textsc{Act}} a.yes.
$$

Similarly the family of axioms (N_a) can be replaced with

$$
(\mathbf{N})\ no = no + \sum_{a \in \textsc{Act}} a.no.
$$

Proof. It is not hard to see that the equation Y can be proved by using the family of equations Y_a. For the converse we can use axioms $A3$ and Y to prove any equation $yes = yes + b.yes$ of the family $\{Y_a \mid a \in \textsc{Act}\}$. Indeed, $\mathcal{E}_v$ proves

$$yes = yes + \sum_{a \in \textsc{Act}} a.yes = yes + \sum_{a \in \textsc{Act}} a.yes + b.yes = yes + b.yes.$$

□

Theorem 4.1. *$\mathcal{E}_v$ is sound modulo $\simeq$. That is, if $\mathcal{E}_v \vdash m = n$ then $m \simeq n$, for all $m, n \in Mon_F$.*

Proof. It suffices to prove soundness for each of the axioms separately. The details of the proof are standard and therefore omitted. □

In what follows, we will consider terms up to axioms $A1$-$A4$.

A fact that will be proven useful later on is the following: If $m \xrightarrow{a} n$ then $A1 - A4, E_a, Y_a, N_a \ \vdash m = m + a.n$. This follows easily by induction on the size of m and a case analysis on its form and it is thus omitted.

We will now prove that the axiom system $\mathcal{E}_v$ is ground complete for verdict equivalence.

Theorem 4.2. *$\mathcal{E}_v$ is ground complete for $\simeq$ over Mon_F. That is, if m, n are closed monitors in Mon_F and $m \simeq n$ then $\mathcal{E}_v \vdash m = n$.*

As a first step towards proving that $\mathcal{E}_v$ is complete over closed terms, we isolate a notion of normal form for monitors and prove that each closed monitor in Mon_F can be proved equal to a normal form using the equations in $\mathcal{E}_v$.

Definition 4.4. ***(Normal Form)*** *A normal form is a closed term $m \in Mon_F$ of the form:*

$$\sum_{a \in A} a.m_a \ [+yes] \ [+no]$$

for some finite $A \subseteq \textsc{Act}$, where each m_a is a term in normal form that is different from end.

Note that, by taking $A = \emptyset$ in the definition above, we obtain that *end* is a normal form. In fact, it is the normal form with the smallest size.

Lemma 4.4. *The only normal form that does not contain occurrences of yes and no is end.*

Proof. We proceed by induction on the size of a normal form m. Our base case is a verdict v. The only such verdict that does not contain an occurrence of either *yes* or *no* is *end*, which trivially satisfies the lemma. Assume now that $m = \sum_{a \in A} a.m_a$ is a normal form satisfying the statement of the lemma. Since each m_a is *yes*- and *no*-free, by inductive hypothesis, $m_a = end$. This is only possible if $A = \emptyset$. Thus $m = end$. □

Lemma 4.5. ***(Normalization)*** *Each closed term $m \in Mon_F$ is provably equal to some normal form m' with* $\text{depth}\,(m') \leq \text{depth}\,(m)$.

Proof. We prove the claim by induction on the lexicographic ordering $\prec$ over pairs $(\text{depth}\,(m)\,, size(m))$ of a monitor m, where $size(m)$ denotes the length of m in symbols. We proceed with a case analysis on the form m may have. Our induction basis will be a verdict v. If $v = end$ then the monitor is already in normal form. Otherwise: If $m = v$ for some verdict $v = yes$ or no then it is proved equal to $v + end$ (from axiom $A4$). Indeed this normal form of a non-*end* verdict has depth less or equal to that of the initial monitor.

Our induction hypothesis is that, for all monitors $m_0 \in Mon_F$ up such that $(\text{depth}\,(m_0)\,, size(m_0)) \prec (\text{depth}\,(m)\,, size(m))$, we have that $\mathcal{E}_v \vdash m_0 = m_0'$ with m_0' in normal form and $\text{depth}\,(m_0') \leq \text{depth}\,(m_0)$.

Assume that $m = a.n$ then clearly n has depth less than that of m and therefore by the inductive hypothesis $\mathcal{E} \vdash n \simeq n'$ where n' is in normal form and of depth less or equal than n. If $n' = end$ then $\mathcal{E}_v \vdash m = end$ (using E_a) which is a normal form of smaller depth. Otherwise $a.n'$ is also a normal form.

Assume that $m = m_1 + m_2$. By applying the induction hypothesis we have that

$$\mathcal{E}_v \vdash m_1 = \sum_{a \in A_1} a.m_{1a}\ [+yes][+no] \text{ and } \mathcal{E}_v \vdash m_2 = \sum_{a \in A_2} a.m_{2a}\ [+yes][+no].$$

Therefore by applying axioms from $\mathcal{E}_v$ we can rewrite m as:

$$m = \sum_{a \in A_1 \setminus A_2} a.m'_{1a} + \sum_{a \in A_2 \setminus A_1} a.m'_{2a} + \sum_{b \in A_1 \cap A_2} b.(m'_{1b} + m'_{2b})\ [+yes][+no].$$

Where by the statement of the lemma we have:

$$\text{depth}\left(\sum_{a \in A_1 \setminus A_2} a.m'_{1a}\right) \leq \text{depth}\,(m_1) \leq \text{depth}\,(m)$$

and similarly:

$$\text{depth}\left(\sum_{a \in A_2 \setminus A_1} a.m'_{2a}\right) \leq \text{depth}\,(m_2) \leq \text{depth}\,(m)\,.$$

It remains to show that the summand $\sum_{b \in A_1 \cap A_2} b.(m'_{1b} + m'_{2b})$ is equal to a normal form and that it has depth less or equal to that of m. However, this is not trivial to see, since the terms m'_{1a} and m'_{2b} have been rewritten by the normalization procedure and therefore we cannot guarantee that their summation has size less of that of m (applying the inductive hypothesis only results in terms of smaller depth but not size as we saw for instance in the case of normalization of verdicts). However we have the following:

$$\text{depth}\,(m'_{1b} + m'_{2b}) = max[\text{depth}\,(m'_{1b})\,, \text{depth}\,(m'_{2b})]$$

$$< \ max[\text{depth}\,(m'_1)\,,\text{depth}\,(m'_2)].$$

The later of the above quantities is guaranteed to be less than or equal to depth (m) by the inductive hypothesis. Therefore we still have that the monitor $m'_{1b}+m'_{2b}$ appears earlier in the lexicographic ordering and therefore $\mathcal{E}_v$ can prove it equal to a normal form of smaller depth. We will call this normal form m'_b. We have therefore that that depth $(m'_b) \leq$ depth $(m'_{1b}+m'_{2b})$. We now have the necessary result that

$$\mathcal{E}_v \vdash m = m' = \sum_{a \in A_1 \cup A_2} a.m_a \ [+yes][+no],$$

where each m_a is in normal form and of depth strictly less than that of m which means that depth $(m') \leq$ depth (m) and we are done. □

Since now we have that each term in Mon_F is provably equal to a normal form, we might attempt to prove Theorem 4.2 by arguing that the normal forms of two verdict equivalent monitors are identical. However, it turns out that this is not true. Consider, for example, the case were $m = yes$ and $n = yes + a.a.a.yes$. These two monitors are clearly verdict equivalent as $L_a(m) = L_a(n) = \text{Act}^*$ and $L_r(m) = L_r(n) = \emptyset$. However, even though they are in normal form they are not syntactically equal. Intuitively, $a.a.a.yes$ in monitor n is redundant, as it can be absorbed by yes. In what follows, we will show how to reduce the normal form of a monitor further using equations in $\mathcal{E}_v$ in order to eliminate such redundant sub-terms.

Lemma 4.6. *The following statements hold for any monitor in Mon_F:*

1. *For each action a, if m is a closed no-free term then $\mathcal{E}_v \vdash yes + a.m = yes$.*

2. *For each action a, if m is a closed monitor that contains occurrences of both yes and no then $\mathcal{E}_v \vdash yes + a.m = yes + a.n$ for some yes-free closed monitor n.*

3. *For each action a, if m is a closed yes-free term then $\mathcal{E}_v \vdash no + a.m = no$.*

4. *For each action a, if m is a closed monitor that contains occurrences of both yes and no then $\mathcal{E}_v \vdash no + a.m = no + a.n$ for some no-free closed monitor n.*

Proof. We only prove statements *1* and *2* as the proofs of *3* and *4* are similar. We will use structural induction on m.

1. If m is a verdict other than no then the claim follows using axioms E_a, Y_a and $A4$ appropriately. If $m = b.m'$ where m' is no-free then $\mathcal{E}_v$ derives:

   $$yes + a.m \overset{\mathrm{Y_a}}{=} yes + a.yes + a.m \overset{\mathrm{D_a}}{=} yes + a.(yes + b.m') \overset{\textbf{I.H.}}{=} yes + a.yes \overset{\mathrm{Y_a}}{=} yes.$$

   If m is of the form $m_1 + m_2$ where m_1, m_2 are no-free, then it suffices to apply axiom D_a and the induction hypothesis.

2. Assume that m contains occurrences of both *yes* and *no*. We will show that $\mathcal{E}_v \vdash \mathit{yes} + a.m = \mathit{yes} + a.n$ for some *yes*-free monitor n.

   If $m = v$ for some verdict v then the claim follows vacuously.

   If $m = b.m'$ for some m' that contains both *yes* and *no* then there is some *yes*-free n', such that $n = b.n'$, and $\mathcal{E}_v$ derives :

$$\mathit{yes} + a.m = \mathit{yes} + a.b.m' \overset{\mathrm{Y_a}}{=} \mathit{yes} + a.\mathit{yes} + a.b.m' \overset{\mathrm{D_a}}{=} \mathit{yes} + a.(\mathit{yes} + b.m')$$

   and for some *yes*-free n' s.t. $n = b.n'$:

$$\overset{\mathrm{I.H}}{=} \mathit{yes} + a.(\mathit{yes} + b.n') \overset{\mathrm{D_a}}{=} \mathit{yes} + a.\mathit{yes} + a.b.n' \overset{\mathrm{Y_a}}{=} \mathit{yes} + a.n.$$

   Finally if $m = m_1 + m_2$ then $\mathcal{E}_v$ can derive:

$$\mathit{yes} + a.(m_1 + m_2) \overset{\mathrm{D_a}}{=} \mathit{yes} + a.m_1 + a.m_2.$$

   We now isolate the following cases based on what verdicts the monitors $m_i,\ i \in \{1,2\}$ contain. If any $m_i,\ i \in \{1,2\}$ is both *yes*- and *no*-free it must be equal to *end* as it is in normal form and therefore $\mathcal{E}_v \vdash \mathit{yes} + a.m_i = \mathit{yes}$. If $m_i,\ i \in \{1,2\}$ contains occurrences of both *yes* and *no*, then the induction hypothesis yields that

$$\mathcal{E}_v \vdash \mathit{yes} + a.m_i = \mathit{yes} + a.n_i$$

   for some *yes*-free n_i. If $m_i, i \in \{1,2\}$ is *yes*-free we already have the result that $\mathcal{E}_v \vdash \mathit{yes} + a.m_i = \mathit{yes} + a.n_i$ for some *yes*-free monitor n_i (which in this case coincides with m_i). Finally, if some m_i is *no*-free then, by statement *1* in the lemma,

$$\mathcal{E}_v \vdash \mathit{yes} + a.m_i = \mathit{yes}.$$

   Combining these observations, we have that:

$$\mathcal{E}_v \vdash \mathit{yes} + a.m = \mathit{yes} + a.n_1 + a.n_2$$

   where both n_1 and n_2 are *yes*-free and therefore by axiom D_a:

$$\mathcal{E}_v \vdash \mathit{yes} + a.m = \mathit{yes} + a.n$$

   for some *yes*-free monitor n. □

The above lemma suggests the notion of a *reduced* normal form.

Definition 4.5. ***(Reduced normal form)*** *A reduced normal form is a term*

$$m = \sum_{a \in A} a.m_a\ [+\mathit{yes}]\ [+\mathit{no}]$$

in normal form, where if $v \in \{\mathit{yes}, \mathit{no}\}$ is a summand of m then each m_a is v-free and in reduced normal form.

Remark 7. *Note here that if* $\sum_{a \in A} a.m_a + \mathit{yes} + \mathit{no}$ *is in reduced normal form then* $A = \emptyset$.

Lemma 4.7. *Each monitor in normal form is provably equal to a monitor in reduced normal form.*

Proof. The claim follows from Lemma 4.6, using induction on the depth of the normal form. □

We are now ready to complete the proof of Theorem 4.2.

Proof of Theorem 4.2. Since each monitor is provably equal to a reduced normal form (Lemma 4.7), and by the soundness of $\mathcal{E}_v$ (Theorem 4.1), it suffices to prove the claim for verdict equivalent reduced normal forms m and n. We proceed by induction on the sum of the sizes of m and n, and a case analysis on the possible form m may have.

1. Assume that $m = \mathit{yes} + \mathit{no} \simeq n$. Since $L_a(m) = L_r(m) = \text{ACT}^*$, it follows that n has both *yes* and *no* as summands. Since n is in reduced normal form it must be the case that $n = \mathit{yes} + \mathit{no}$, and we are done.

2. Assume that $m = \sum_{a \in A} a.m_a + \mathit{yes} \simeq n$, where, for all $a \in A$, m_a is *yes*-free and in reduced normal form and $n = \sum_{b \in B} b.n_b[+\mathit{yes}][+\mathit{no}]$, where each n_b is in reduced normal form and is v-free, if v is a summand of n. Since $\varepsilon \in L_a(m) \setminus L_r(m)$, we have that *yes* is a summand of n and *no* is not. Thus $n = \sum_{b \in B} b.n_b + \mathit{yes}$, and each n_b is *yes*-free. We claim that:

   (C1) $A = B$ and

   (C2) for all $a \in A$, $m_a \simeq n_a$.

   To prove that $A = B$, we assume that $a \in A$. Since m_a is *yes*-free and different from *end*, there is some $s \in \text{ACT}^*$ such that $a.s \in L_r(m)$. As $m \simeq n$, we have that $a.s \in L_r(n)$. We conclude that $a \in B$ and $s \in L_r(n_a)$. By symmetry, claim **(C1)** follows.

   We now show that $m_a \simeq n_a$ for each $a \in A$. Since m_a and n_a are *yes*-free, $L_a(m_a) = L_a(n_a) = \emptyset$. We pick now some arbitrary $s \in L_r(m_a)$ ($L_r(m_a) \neq \emptyset$ because $m_a \neq \mathit{end}$). This means that $a.s \in L_r(m) = L_r(n)$ and therefore $s \in L_r(n_a)$. The claim follows by symmetry. By the induction hypothesis, $\mathcal{E}_v \vdash m_a = n_a$ for each $a \in A = B$. Therefore

$$m = \sum_{a \in A} a.m_a + \mathit{yes} = \sum_{b \in B} b.n_b + \mathit{yes} = n$$

   is provable from $\mathcal{E}_v$ and we are done.

3. We are left with the case where $m = \sum_{a\in A} a.m_a + no \simeq n$ and the case $m = \sum_{a\in A} a.m_a$. The proofs for those cases are similar to the one for case *2* and are thus omitted.

□

4.3.1 Axiomatizing ω-Verdict Equivalence

When $\textsc{Act}$ is infinite, by Lemma 4.2 and Theorem 4.2, $\mathcal{E}_v$ gives a ground-complete axiomatization of ω-verdict equivalence as well. However, when $\textsc{Act}$ is finite, $\mathcal{E}_v$ is not powerful enough to prove all the equalities between closed terms that are valid with respect to ω-verdict equivalence. The new axioms needed to achieve a ground complete axiomatization in this setting are:

$$(\mathbf{Y}_\omega)\quad yes = \sum_{a\in\textsc{Act}} a.yes \qquad\qquad (\mathbf{N}_\omega)\quad no = \sum_{a\in\textsc{Act}} a.no.$$

The resulting axiom system is called $\mathcal{E}_\omega$.

Remark 8. *The soundness of the new axioms is trivially shown since*

$$L_a(yes)\cdot\textsc{Act}^\omega = \textsc{Act}^*\cdot\textsc{Act}^\omega = \textsc{Act}^+\cdot\textsc{Act}^\omega = L_a(\sum_{a\in\textsc{Act}} a.yes)\cdot\textsc{Act}^\omega$$

while $L_r(yes) = L_r(\sum_{a\in\textsc{Act}} a.yes) = \emptyset$ *(and symmetrically for the* N_ω *equation).*

Theorem 4.3. *$\mathcal{E}_\omega$ is ground complete for $\simeq_\omega$ over closed terms when* $\textsc{Act}$ *is finite. That is if m, n are closed monitors in Mon_F and $m \simeq_\omega n$ then $\mathcal{E}_\omega \vdash m = n$.*

Proof. By Lemma 4.7 we may assume that m and n are in reduced normal form. We will prove the claim by induction on the sizes of m and n for two ω-verdict equivalent monitors m, n in reduced normal form.

We will proceed by a case analysis of the form m may have and limit ourselves to presenting the proof for a few selected cases that did not arise in the proof of Theorem 4.2.

- Assume that $m = yes + no \simeq_\omega \sum_{a\in A} a.n_a = n$. First of all note that $A = \textsc{Act}$. Indeed if $a \in \textsc{Act}\setminus A$ then $a^\omega \in (L_a(m)\cdot\textsc{Act}^\omega)\setminus(L_a(n)\cdot\textsc{Act}^\omega)$ which contradicts our assumption that $m \simeq_\omega n$. Moreover, it is not hard to see that, for each $a \in \textsc{Act}$, $L_a(n_a)\cdot\textsc{Act}^\omega = L_r(n_a)\cdot\textsc{Act}^\omega = \textsc{Act}^\omega$. This means that, for each $a \in \textsc{Act}$, $n_a \simeq_\omega yes + no$. By induction, for each $a \in \textsc{Act}$, we have that $\mathcal{E}_\omega \vdash n_a = yes + no$. Thus, $\mathcal{E}_\omega \vdash n = \sum_{a\in\textsc{Act}} a.(yes + no)$. From axiom D_a, $\mathcal{E}_\omega \vdash n = \sum_{a\in\textsc{Act}} a.yes + \sum_{a\in\textsc{Act}} a.no$ which from our two new axioms Y_ω, N_ω yields $\mathcal{E}_\omega \vdash n = yes + no = m$, and we are done.

- Assume that $m = \mathit{yes}+\mathit{no} \simeq_\omega \sum_{a\in A} a.n_a+\mathit{yes}$, with each n_a being *yes*-free and different from *end*. Again, reasoning as in the previous case, we have that $A = \textsc{Act}$. Moreover for each $a \in \textsc{Act}$, $L_r(n_a)\cdot\textsc{Act}^\omega = \textsc{Act}^\omega$. Following the same argument as above only for the *no* verdict we arrive at the conclusion that $\mathcal{E}_\omega \vdash n = \mathit{yes} + \sum_{a\in\textsc{Act}} a.\mathit{no} = \mathit{yes} + \mathit{no} = m$.

- The case $m = \mathit{yes} + \mathit{no} \simeq_\omega \sum_{a\in A} a.n_a + \mathit{no}$ is symmetrical to the one above.

- Assume that $m = \mathit{yes} + \sum_{a\in A} a.m_a \simeq_\omega \sum_{b\in B} b.n_b$ where both m and n are in reduced normal form. First of all, we follow an argument similar to the first case analyzed above, to the point where $\mathcal{E}_\omega \vdash n = \mathit{yes} + \sum_{b\in B'} b.n'_b$ for some *yes*-free monitors n'_b. For the proof of this final case we will use the following facts, whose validity can be easily established:

  (S1) $B = \textsc{Act}$,

  (S2) for all $b \in \textsc{Act}$, $L_a(n_b) = \textsc{Act}^\omega$, and

  (S3) for all $a \in A$, $L_r(m_a) = L_r(n_a)$.

  So, for each $a \in A$, $\mathit{yes} + m_a \simeq_\omega n_a$. Since both of these monitors have smaller depth that the original ones, we have that by induction:

  $$\mathcal{E}_\omega \vdash \mathit{yes} + m_a = n_a,\ \forall\ a \in A. \tag{4.1}$$

  For each $b \in \textsc{Act} \setminus A$, we have that $\mathit{yes} \simeq_\omega n_b$ (because $L^\omega_r(n_b) = \emptyset$). Again, we have that, by induction:

  $$\mathcal{E}_\omega \vdash \mathit{yes} = n_a,\ \forall\ b \in \textsc{Act} \setminus B. \tag{4.2}$$

  So:

  $$\mathcal{E}_\omega \vdash n = \sum_{b\in\textsc{Act}} b.n_b = \sum_{a\in A} a.n_a + \sum_{b\in\textsc{Act}\setminus A} b.\mathit{yes}$$

  By equations (4.1) and (4.2):

  $$\begin{aligned}\mathcal{E}_\omega \vdash n &= \sum_{a\in A} a.(\mathit{yes} + m_a) + \sum_{b\in\textsc{Act}\setminus A} b.\mathit{yes}\\ &= \sum_{a\in A} a.\mathit{yes} + \sum_{a\in A} a.m_a + \sum_{b\in\textsc{Act}\setminus A} b.\mathit{yes}\\ &= \sum_{a\in\textsc{Act}} a.\mathit{yes} + \sum_{a\in A} a.m_a = \mathit{yes} + \sum_{a\in A} a.m_a,\end{aligned}$$

  using axiom Y_ω, and we are done.

The above analysis can be applied symmetrically for the cases:

- $m = no + \sum_{a \in A} a.m_a \simeq_\omega \sum_{b \in B} b.n_b = n$ and
- $m = \sum_{a \in A} a.m_a \simeq_\omega \sum_{b \in B} b.n_b = n.$

This completes the proof. □

4.4 Open Terms

Thus far, we have only studied the completeness of equational axiom systems for $\simeq$ and $\simeq_\omega$ over closed terms. However, in our grammar we allow for variables and it is natural to wonder whether the ground-complete axiomatizations we have presented in Theorems 4.2 and 4.3 are also complete for verdict equivalence and ω-verdict equivalence over open terms. Unfortunately, this turns out to be false. Indeed, the equation

$$(\mathbf{O1}) \quad yes + no = yes + no + x$$

is valid with respect to $\simeq$ (as both sides trivially accept and reject all traces), but cannot be proved using the equations in $\mathcal{E}_\omega$. This is because all the equations in that axiom system have the same variables on their left- and right-hand sides. Our goal in the remainder of this section is to study the equational theory of $\simeq$ and $\simeq_\omega$ over open terms. Subsection 4.4.1 will present our results when Act is infinite as this case turns out to be more straightforward. We consider the setting of a finite set of actions in Subsection 4.4.2. In what follows, we use $\mathcal{E}'_v$ for the axiom system that results by adding $O1$ to $\mathcal{E}_v$. The superscript $'$ will be used in the name of an axiom set to denote that the axiom set is complete for one notion of equivalence over *open* terms. The absence of a superscript refers respectively to a *ground complete* axiom set.

Towards a completeness theorem, we modify the notion of normal form, to take variables into account. To that end we define:

Definition 4.6. *A term $m \in Mon_F$ is in **open normal form** if it has the form:*

$$m = \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \ [+yes] \ [+no]$$

where $\{x_i \mid i \in I\}$ is a finite set of variables, A is a finite subset of Act *and each m_a is an (open) term in open normal form that is different from end.*

Lemma 4.8. *Each open term $m \in Mon_F$ is provably equal to some open normal form m' with* $\mathrm{depth}(m') \leq \mathrm{depth}(m)$.

The proof of the above result follows the lines of the one for Lemma 4.5 for closed terms and is thus omitted.

As in the case of closed terms, we now proceed to characterize a class of open normal forms for open terms whose verdict equivalence can be detected "structurally". The following example highlights the role that equation $(O1)$ plays in that characterization.

***Example* 9.** *Consider the following monitor in open normal form:*

$$m = x + yes + a.b.(no + b.a.x).$$

Monitor m contains two occurrences of the variable x. However, because of the interplay between the two verdicts, one of them is redundant and can be removed thus:

$$\begin{aligned}
\mathcal{E}'_v \vdash m &= x + yes + a.b.(no + b.a.x)\\
&\overset{\mathrm{Y_a}}{=} x + yes + a.yes + a.b.(no + b.a.x)\\
&\overset{\mathrm{Y_b}}{=} x + yes + a.(yes + b.yes) + a.b.(no + b.a.x)\\
&\overset{\mathrm{D_a}}{=} x + yes + a.(yes + b.yes + b.(no + b.a.x))\\
&\overset{\mathrm{D_b}}{=} x + yes + a.(yes + b.(yes + no + b.a.x))\\
&\overset{\mathrm{O_1}}{=} x + yes + a.(yes + b.(yes + no))\\
&\overset{\mathrm{D_b}}{=} x + yes + a.(yes + b.yes + b.no)\\
&\overset{\mathrm{Y_b}}{=} x + yes + a.(yes + b.no)\\
&\overset{\mathrm{D_a}}{=} x + yes + a.yes + a.b.no\\
&\overset{\mathrm{Y_a}}{=} x + yes + a.b.no.
\end{aligned}$$

The above example motivates the following notion of reduced normal form for open terms.

Definition 4.7. *An* ***open reduced normal form*** *is a term*

$$m = \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \; [+yes] \; [+no]$$

where if $v \in \{yes, no\}$ is a summand of m then each m_a is v-free, different from end and in open reduced normal form. In addition:

- *if both yes and no are summands of m then m is equal to $yes + no$,*
- *if yes is a summand of m and $m \xrightarrow{s} no + m'$, for some s and m' then m' is equal to end,*
- *if no is a summand of m and $m \xrightarrow{s} yes + m'$, for some s and m' then m' is equal to end.*

In what follows we will omit the word "open" when referring to the normal form of a term that contains variables.

Lemma 4.9. *For each open monitor $m \in Mon_F$, its normal form is provably equal to a reduced normal form.*

Proof. By Lemma 4.8 we may assume that m is in in normal form. The proof is by induction on the size of m and we isolate the following cases, depending on the verdicts $v \in \{\mathit{yes}, \mathit{no}\}$ m has as summands:

1. Case $m = \sum_{a\in A} a.m_a + \sum_{i\in I} x_i$. In this case we use the induction hypothesis on the m_a monitors. These are different from *end* and have smaller size than m and therefore they are provably equal to a reduced normal form, i.e $\mathcal{E}'_v \vdash m_a = m'_a$ where m'_a is in reduced normal form. Thus $\mathcal{E}'_v$ proves $m = \sum_{a\in A} a.m'_a + \sum_{i\in I} x_i$, and we are done since $\sum_{a\in A} a.m'_a + \sum_{i\in I} x_i$ is in reduced normal form.

   By applying the congruence closure equational law we have that $\mathcal{E}'_v \vdash m = m'$, where m' is in reduced normal form.

2. Case $m = \mathit{yes} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i$. In this case by the induction hypothesis each m_a is provably equal to a reduced normal form. The extra step here is that if $m \xrightarrow{s} \mathit{no} + m'$, for some s and m' then m' is equal to *end*. In such a scenario we have that:

   If $s = \varepsilon$, then the claim follows trivially from $O1$. Otherwise $s = a.s'$ for some action $a \in \textsc{Act}$ and $m \xrightarrow{a} m_a \xrightarrow{s'} \mathit{no} + m'$. We now apply our axioms as follows:

$$m = \mathit{yes} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i \overset{\mathrm{Y_a}}{=} \mathit{yes} + \sum_{b\in A\setminus\{a\}} b.m_b + a.\mathit{yes} + a.m_a + \sum_{i\in I} x_i$$

$$\overset{\mathrm{D_a}}{=} \mathit{yes} + \sum_{b\in A\setminus\{a\}} b.m_b + a.(\mathit{yes} + a.m_a) + \sum_{i\in I} x_i.$$

   This means that since $\mathit{yes} + m_a$ has size smaller than m it is provably equal to a reduced normal form. Additionally, since it contains a *yes* summand and $m_a \xrightarrow{s'} \mathit{no} + m'$, by the induction hypothesis we have that m' is equal to *end* and we are done.

3. Case $m = \mathit{no} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i$. The proof of this case is symmetrical to Case 2 and therefore omitted.

4. Case $m = \mathit{yes} + \mathit{no} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i$. In this case we use the following simple argument. Starting for axiom O_1 we use the substitution $\sigma(x) = \sum_{a\in A} a.m_a + \sum_{i\in I} x_i$ and we get:

$$\mathit{yes} + \mathit{no} = \mathit{yes} + \mathit{no} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i,$$

   and by applying the equational law of transitivity we have that $\mathcal{E}'_v \vdash m = \mathit{yes} + \mathit{no}$. □

The normal form defined above for open terms is adjusted over the closed terms case. This is because now our syntax is allowing for variables and therefore it is convenient for proofs to take these variables into account in a controlled and consistent manner. The further reducing that occurred towards defining the open reduced normal forms was possible due to the existence of the new axiom O_1, which gave us the option to remove variable occurrences. The new axiom O_1 is the only axiom we have currently available that does not contain every variable occurrence in both of its sides and it is therefore the only rule we have available that can help us remove variables from equations. In the presence of other axioms with this property we can further reduce our normal forms, as we will see later on.

In the following subsections, we will study the full equational theory of verdict and omega-verdict equivalence over open terms.

4.4.1 Infinite Set of Actions

We begin by considering the equational theory of open monitors when the set of actions is infinite. Apart from its theoretical interest, this scenario has also some practical relevance. Indeed, as shown already by Milner in [153, 152], infinite sets of uninterpreted actions are useful when modeling system events that carry data values. Runtime monitoring of systems with data-dependent behavior has been an active field of research for over 15 years—see, for instance, the paper [44] for an early reference.

When the set of actions Act is infinite, it is easy to define a one-to-one mapping from open to closed terms that will help us prove completeness of the axiom system $\mathcal{E}'_v$.

Theorem 4.4. *(Completeness for open terms modulo $\simeq$) $\mathcal{E}'_v$ is complete for $\simeq$ over open monitors in Mon_F when* Act *is infinite. That is, for all $m, n \in Mon_F$, if $m \simeq n$, then $\mathcal{E}'_v \vdash m = n$.*

Proof. Assume $m \simeq n$. By Lemma 4.9, we may assume that m and n are in reduced normal form.

Let

$$m = \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \ [+yes] \ [+no]$$

and

$$n = \sum_{b \in B} b.n_b + \sum_{j \in J} y_j \ [+yes] \ [+no].$$

We will show that $\mathcal{E}'_v \vdash m = n$ by induction on the sum of the sizes of m and n. To this end, we will establish a strong structural correspondence between m and n. Consider a substitution σ defined as follows: $\sigma(x) = a_x.(yes + no)$ where

- for all variables x and y, $a_x = a_y$ implies $x = y$, and
- $\{a_x \mid x \in \mathcal{V}\}$ is disjoint from the set of actions occurring in m or n.

Note that such a substitution σ exists because Act is infinite. By induction on the sizes of m and n, we will prove that if $\sigma(m) \simeq \sigma(n)$ then:

(C1) v is a summand of m iff v is a summand of n, for $v \in \{yes, no\}$,

(C2) $\{x_i \mid i \in I\} = \{y_j \mid j \in J\}$,

(C3) $A = B$ and

(C4) for each $a \in A$, $\sigma(m_a) \simeq \sigma(n_a)$.

In what follows, we first show that $\mathcal{E}'_v$ proves $m = n$ assuming claims **(C1)-(C4)** and then we prove those claims. To prove that $\mathcal{E}'_v$ proves $m = n$ follows from $\sigma(m) \simeq \sigma(n)$ for reduced normal forms m and n, we proceed by induction on the sum of the sizes of m and n. By claim **C4**, we have that $\sigma(m_a) \simeq \sigma(n_a)$ and, from the induction hypothesis, $\mathcal{E}'_v \vdash m_a = n_a$. By **C1-3** we also have that $\mathcal{E}'_v \vdash \sum_{i \in I} x_i = \sum_{j \in J} y_j$ and that $\sum_{a \in A} a.m_a = \sum_{b \in B} b.n_b$, which means that by using the equational law of closure under summation we also have that $\mathcal{E}'_v \vdash m = n$.

We present now the proofs of **(C1)-(C4)**.

C1: Assume *yes* is a summand of m. Then $\varepsilon \in L_a(\sigma(m))$. Since $\sigma(m) \simeq \sigma(n)$, we have that $\varepsilon \in L_a(\sigma(n))$. Note that $\varepsilon \notin L_a(\sigma(x))$ for each x. Thus *yes* must be a summand of n. The case for $v = no$ is similar. By symmetry the claim follows.

C2: Assume that $x \in \{x_i \mid i \in I\}$. By the definition of σ, it follows that $\sigma(m)$ both accepts and rejects the trace a_x. Since $m \simeq n$, we have that $\sigma(n)$ also accepts and rejects the trace a_x. As n does not contain any occurrence of a_x and has at most one of the verdicts *yes* and *no* as a summand, it follows that $x \in \{y_j \mid j \in J\}$. Therefore, by symmetry, $\{x_i \mid i \in I\} = \{y_j \mid j \in J\}$ and we are done.

C3: Assume, towards a contradiction, that $a \in A \setminus B$. Then m cannot have both *yes* and *no* as summands, since m is in reduced normal form.

If m has none of the verdicts as a summand, we know that m_a is different from *end* since m is in reduced normal form. Therefore $\sigma(m_a)$ will either accept or reject some trace s, which implies that $\sigma(m)$ will also accept or reject as. However, $\sigma(n)$ cannot do the same because $a \notin B$, $\sigma(x) \not\xrightarrow{a}$ for each x, and neither *yes* nor *no* are summands of n. This contradicts our assumption that $m \simeq n$.

Assume now, without loss of generality, that m has only the verdict *yes* as summand. Observe that m_a is *yes*-free and different from *end*, since m is in reduced normal form. This means that $\sigma(m_a)$ can reject some trace s and, therefore, that $\sigma(m)$ will reject $a.s$. On the other hand, $\sigma(n)$ cannot do the same because $a \notin B$, $\sigma(x) \not\xrightarrow{a}$ for each x and *no* is not a summand of n. Again, this contradicts our assumption that $m \simeq n$.

The above analysis yields that $A \subseteq B$. By symmetry, $A = B$ follows.

C4: Our final claim (and the one with the most involved proof) is that $\sigma(m_a) \simeq \sigma(n_a)$, for each $a \in A$.

If the reduced normal forms of the monitors do not contain any verdict $v \in \{yes, no\}$ as a summand, then the argument is simplified significantly. Therefore, we limit ourselves to presenting here the most complicated case, where m and n both contain exactly one verdict $v \in \{yes, no\}$ as a summand. Without loss of generality, we assume that this verdict is *yes*, namely that

$$m = yes + \sum_{a \in A} a.m_a + \sum_{i \in I} x_i$$

and

$$n = yes + \sum_{b \in B} b.n_b + \sum_{j \in J} y_i.$$

Since the claims **C1-3** have already been proven, we know for m and n that:

$$m = yes + \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \text{ and } n = yes + \sum_{a \in A} a.n_a + \sum_{i \in I} x_i \ .$$

We remind the reader that our purpose is to prove that $\sigma(m_a) \simeq \sigma(n_a)$, for each $a \in A$, so that we can apply our induction hypothesis to infer that $\mathcal{E}'_v \vdash m_a = n_a$.

We first prove that the rejection sets of $\sigma(m_a)$ and $\sigma(n_a)$ are equal. To this end, assume that $s \in L_r(\sigma(m_a))$. It follows that $a.s \in L_r(\sigma(m)) = L_r(\sigma(n))$. By the form of n and from the definition of σ, we conclude that $s \in L_r(\sigma(n_a))$. Therefore, $L_r(\sigma(m_a)) \subseteq L_r(\sigma(n_a))$. By symmetry we have that $L_r(\sigma(m_a)) = L_r(\sigma(n_a))$ and we are done.

It remains to prove that the acceptance sets of $\sigma(m_a)$ and $\sigma(n_a)$ are also identical. (It is important here to point out that, since both m and n contain a *yes* verdict as a summand, the acceptance sets of $\sigma(m)$ and $\sigma(n)$ are both equal to ACT^*. However, for our inductive argument to work, we need to be able to prove that $L_a(\sigma(m_a)) = L_a(\sigma(n_a))$.) To that end and towards a contradiction, consider a shortest trace s that is accepted by monitor $\sigma(m_a)$, but not by $\sigma(n_a)$. Consequently, monitor $\sigma(m)$ accepts the trace $a.s$.

Since monitors m_a and n_a are *yes*-free, as a result of m and n being in reduced normal form, the acceptance of s must be the result of a variable x mapped to $a_x.(yes+no)$ through the substitution σ. Since s is a shortest trace that is accepted by monitor $\sigma(m_a)$, but not by $\sigma(n_a)$, none of its prefixes is accepted by $\sigma(m_a)$ and therefore the last action that is in s must be the action a_x stemming from $\sigma(x)$. This means that monitor m_a can perform the transition $m_a \overset{s'}{\Rightarrow} m'_a$, where m'_a contains x as a summand and $s = s'.a_x$. Therefore the monitor $\sigma(m_a)$ can perform the transitions:

$$\sigma(m_a) \overset{s'}{\Rightarrow} \sigma(m'_a) \xrightarrow{a_x} yes + no \xrightarrow{\tau} yes.$$

Since $s'.a_x$ is accepted by $\sigma(m_a)$, it must also be rejected by it because a_x is an action that can only be observed after the substitution of the variable x in m_a. We have already argued that the rejection sets of $\sigma(m_a)$ and $\sigma(n_a)$ are equal and therefore $\sigma(n_a)$ also rejects the trace $s'.a_x$. Since the action a_x is a unique action corresponding to the variable x, there are only two ways in which $\sigma(n_a)$ could reject the trace $s'.a_x$. The first case is that $\sigma(n_a)$ can also perform the transitions

$$\sigma(n_a) \overset{s'}{\Rightarrow} \sigma(n'_a) \xrightarrow{a_x} yes + no$$

wfor some n'_a. However, this would guarantee that $\sigma(n_a)$ accepts s, whereas we assumed that it does not.

The most complicated case is when $\sigma(n_a)$ can reject a prefix s_0 of s'. By the already proven equality of the rejection sets of the two sub-monitors, $\sigma(m_a)$ would also reject s_0. This can only happen if both n_a and m_a rejected that prefix independently of the substitution σ, since every action preceding a_x along the trace s' is not an action corresponding to the mapping of a variable through σ as explained above. This means that both m_a and n_a can perform the transitions $m_a \overset{s_0}{\Rightarrow} no + m'_a$ and $n_a \overset{s_0}{\Rightarrow} no + n'_a$, for some m'_a and n'_a. However, since m and n are in reduced normal form, this implies that m'_a and n'_a are equal to *end*. This leads us to a contradiction, as we assumed that $\sigma(m_a)$ accepts the trace s which can no longer be the case if $m_a \xrightarrow{s_0} no + end$ where s_0 is a prefix of s.

Therefore every trace accepted by $\sigma(m_a)$ is also accepted by $\sigma(n_a)$. By symmetry, we have that the acceptance sets of m_a and n_a are equal.

This means that $\sigma(m_a) \simeq \sigma(n_a)$, which completes the proof of **C4** and consequently of the whole theorem. □

Corollary 4.4.1. *$\mathcal{E}'_v$ is complete for $\simeq_\omega$ over open monitors in Mon_F when* Act *is infinite. That is, for all $m, n \in Mon_F$, if $m \simeq n$, then $\mathcal{E}'_v \vdash m = n$.*

Proof. The claim follows from Lemma 4.2. □

4.4.2 Finite Set of Actions

The study of the equational theory of $\simeq$ when Act is finite turns out to be more interesting and complicated. In this setting, we can identify equations whose validity depends on the cardinality of Act, which is not the case for any of the axioms we used so far. To see this, consider the equation

$$(\mathbf{V_1}) \quad x = x + a.x,$$

which is sound when Act $= \{a\}$ but cannot be derived by the equations in $\mathcal{E}'_v$, as it is not sound when Act $\neq \{a\}$.

As a first step in our study of the equational theory of $\simeq$ when Act is finite, we characterize some properties of sound equations.

Lemma 4.10. *Let $m \simeq n$ be a sound equation, where $m, n \in Mon_F$ and m is in reduced normal form. Assume that*

- *$m \xrightarrow{s} x + m'$, for some s in* Act**, variable x and m' in Mon_F, and*
- *$m \overset{s_p}{\not\rightarrow} x + m_{s_p}$, for each proper prefix s_p of s and $m_{s_p} \in Mon_F$.*

Then, $n \xrightarrow{s} x + n'$ for some n' in Mon_F.

Proof. Consider the substitution

$$\sigma(y) = \begin{cases} yes + no, & \text{if } y = x \\ end, & \text{if } y \neq x. \end{cases}$$

Since $m \xrightarrow{s} x + m'$ by one of the assumptions of the lemma, we have that $\sigma(m)$ will both accept and reject s. Since $m \simeq n$ is sound we have that $\sigma(n)$ must do the same. If $n \overset{s}{\not\rightarrow} x + n'$ for every n' then it is not hard to see that there are two ways in which n could accept and reject s:

1. $n \overset{s'}{\Rightarrow} yes$ and $n \overset{s'}{\Rightarrow} no$ where s' is a prefix of s (including s itself), or

2. $n \xrightarrow{s'} x + n'$ where s' is a prefix of s (so that $\sigma(n)$ would accept and reject s' and therefore s).

In the first case, consider the substitution σ_e that maps all variables to *end*. Since $n \overset{s'}{\Rightarrow} yes$ and $n \overset{s'}{\Rightarrow} no$, we have that $\sigma_e(n)$ accepts and rejects s'. From $m \simeq n$, we have that $\sigma_e(m)$ also accepts and rejects s'. It is not hard to see that this means that $m \overset{s'}{\Rightarrow} yes$ and $m \overset{s'}{\Rightarrow} no$ However, this is impossible because m is a reduced normal form and $m \xrightarrow{s} x + m'$ by the proviso of the lemma.

In the second case, even though both monitors accept and reject s, we also have that $\sigma_e(n)$ also accepts and rejects s'. Again, since the two monitors are verdict equivalent, we know that $\sigma_e(m)$ must do the same. Since m is in reduced normal form and $m \overset{s_p}{\not\Rightarrow} x + m'$ for any prefix s_p of s (and therefore neither for s') we have that $\sigma(m)$ can only accept and reject s' by performing the transitions $m \overset{s''_1}{\Longrightarrow} yes$ and $m \overset{s''_2}{\Longrightarrow} no$, for s''_1 and s''_2 prefixes of s'. This however is not allowed since it contradicts the fact that m is in reduced normal form and $m \xrightarrow{s} x + m'$. Since both cases have led to a contradiction, we can infer that there is some n' such that $n \overset{s}{\Rightarrow} x + n'$, which was to be shown. □

Corollary 4.4.2. *Let $m \simeq n$ be a sound equation, where $m, n \in Mon_F$ and m is in reduced normal form. Assume that*

- *$m \xrightarrow{s} x + m'$, for some s in* $\textsc{Act}^*$*, variable x and m' in Mon_F, and*

- *$n \overset{s}{\not\rightarrow} x + n'$, for any $n' \in Mon_F$.*

then we have that there exists an s_p prefix of s such that

- *$m \xrightarrow{s_p} x + m_{s_p}$ and $n \xrightarrow{s_p} x + n_{s_p}$ for some m_{s_p} and n_{s_p} in Mon_F, and*

- *for any prefix s_0 of s_p we have that $m \overset{s_0}{\not\Rightarrow} x + m'$ and $n \overset{s_0}{\not\Rightarrow} x + n'$ for any m' and n' in Mon_F.*

Proof. Assume a sound equation $m \simeq n$ for which we have $m \xrightarrow{s} x + m'$, for some s in $\textsc{Act}^*$, variable x and m' in Mon_F. If this is the first occurrence of x along the trace s in m (i.e. $m \overset{s_p}{\not\longrightarrow} x + m_{s_p}$, for each proper prefix s_p of s and $m_{s_p} \in Mon_F$), then by Lemma 4.10, we would have that n must be able to perform the transitions $n \xrightarrow{s} x + n'$, for some n' in Mon_F. Since this cannot be the case as the proviso of the corollary forbids it we have that there must be a prefix s_p of s such that $m \xrightarrow{s_p} x + m_{s_p}$.

Without loss of generality we assume s_p to be the shortest such trace, which means there are no other occurrences of the variable x along the trace s_p. We can therefore see that now for the trace s_p, Lemma 4.10 holds and therefore $n \xrightarrow{s_p} x{+}n_{s_p}$ for some n_{s_p} in Mon_F. Additionally since we assumed s_p t be the shortest trace of the necessary property we already have that $m \overset{s_0}{\not\Rightarrow} x + m'$ for any m' in Mon_F.

It remains to show that the same must hold for n. This can be easily seen to be the case since if we assumed the opposite where for some prefix s_0 of s_p we had $n \xrightarrow{s_0} x + n_0$ for some n_0 then by the symmetric analysis and by using the previous lemma and this corollary we would arrive at a contradiction of s_p being the shortest prefix of s for which $m \xrightarrow{s_p} x + m_{s_p}$. □

Remark 9. *In what follows, when studying open equations, we will refer to occurrences of variables such as the one mentioned in the above corollary, where only one of the monitors involved in the equation can reach a term of the form $x + m_x$ after observing a trace s, as "one-sided" variable occurrences.*

Intuitively Lemma 4.10 states that on each sound equation (including axioms) of which at least one side is in reduced normal form, the first occurrence of each variable per distinct trace leading to the variable is common for both sides of the equation.This gives us some handy intuition on what restrictions an equation that is sound must satisfy.

The following example shows Lemma 4.10 in action.

***Example* 10.** *The equation*

$$x + a.(x + a.(yes + no) + b.(yes + no)) = x + a.(a.(yes + no) + b.(yes + no))$$

is sound over the set of actions $\text{ACT} = \{a, b\}$*, but*

$$x + a.(x + a.(yes + no) + b.(yes + no)) = a.(x + a.(yes + no) + b.(yes + no))$$

is not since the first occurrence of the variable x in the second example happens after the prefix ε on the left-hand side but after the prefix a on the right. In the second equation, the earliest occurrence of the variable x (after the prefix ε) is one-sided.

Also notice here the importance of the sub-term $a. \sum_{a \in \text{ACT}} a.(yes + no)$. We will see that this type of sub-term is crucial for the soundness of the open equations with one-sided variable occurrences we encounter later on.

The following notation will be used in what follows to describe a family of sound equations that generalize the one given in Example 10.

Definition 4.8. ***(Notation)*** *Let $s \in \text{ACT}^*$.*

1. *We use $pre(s)$ to denote the set of prefixes of s (including s).*
2. *We use s^i, $i \geq 1$, to denote the trace s if $i = 1$ and ss^{i-1} otherwise.*

3. *We use $s.m$ to stand for a monitor that can perform exactly the actions along the finite trace s and then become m.*

4. *We define $\overline{s}^{\leq}(m) = \sum_{\substack{|s'|\leq|s|,\\ s'\notin pre(s)}} s'.m$. The monitor $\overline{s}^{\leq}(m)$ is one that behaves like m after having observed any trace of length at most $|s|$ that is not a prefix of s.*

5. *The term $\overline{s}(m)$ is defined thus: $\overline{s}^{\leq}(m) + s. \sum_{a\in\textsc{Act}} a.m$.*

   Intuitively $\overline{s}(yes + no)$ stands for the monitor that accepts and rejects all traces that do not *cause the acceptance or rejection of the string s. Those are exactly the traces that are shorter than s but not its prefixes, and also the ones extending s.*

6. *With the term $\overline{s}^{(k)}(m)$, for $k \geq 1$, we will mean the summation:*

   $\overline{s}(m)$ if $k = 1$ and $\sum_{1\leq i<k-1} s^i.\overline{s}^{\leq}(m) + s^{k-1}.\overline{s}(m)$ if $k \geq 2$.

   Intuitively $\overline{s}^{(k)}(yes + no)$ stands for a monitor that, after observing the fixed trace s, accepts and rejects everything except the trace s^k (and its prefixes).

We now present an example of the usage of the above notation in order to help the reader understand the equations presented later involving these new notions.

***Example* 11.** *For a set of actions $\textsc{Act} = \{a, b\}$, the monitor $m = yes + no$ and a trace $s = ab$ we have that:*

- *$pre(s) = \{\varepsilon, a, ab\}$*
- *$\overline{s}^{\leq}(m) = b.(yes + no) + a.a.(yes + no) + b.b.(yes + no) + b.a.(yes + no)$*
- *$\overline{s}(m) = b.(yes + no) + a.a.(yes + no) + b.b.(yes + no) + b.a.(yes + no) + a.b. \sum_{c\in\textsc{Act}} c.(yes + no)$*
- *and for $k = 3$ we get*

$$\overline{s}^{(3)}(m) = s.\overline{s}^{\leq}(m) + s^2.\overline{s}(m) =$$

$$a.b.(b.(yes + no) + a.a.(yes + no))+$$

$$a.b.a.b.(b.(yes + no) + a.a.(yes + no) + b.b.(yes + no)+$$

$$b.a.(yes + no) + a.b. \sum_{c\in\textsc{Act}} c.(yes + no))$$

This notation defined and presented above is very useful once one understands a very particular form equations among open monitors take when they involve one-sided variable occurrences. Consider, for instance, the following sound equation (for a fixed constant k):

$$x + a^k.x + \overline{a^k}^3(yes + no) \simeq x + \overline{a^k}^3(yes + no) \ .$$

We will formally prove the soundness of (a more general form of) this equation later on. We can intuitively see from the examples above that when an equation contains a one-sided variable occurrence, then the rest of the terms involved in the equation must have some specific form as well so that the equation will stay sound under all possible substitutions. This means that certain traces must always be accepted and rejected by both sides independently of a substitution.

The following lemma formalizes this intuition.

Lemma 4.11. *Assume $m \simeq n$, where m, n are in reduced normal form. If $m \xrightarrow{s} x + m'$ for some m' but $n \overset{s}{\not\rightarrow} x + n'$ for any n', then there exist s', s'' such that $s = s's''$ and, for all $s_b = ss_p$ where $s_p \notin pre(s'')$, either:*

- *$m \overset{s_b}{\Longrightarrow} yes$, $m \overset{s_b}{\Longrightarrow} no$, $n \overset{s_b}{\Longrightarrow} yes$ and $n \overset{s_b}{\Longrightarrow} no$ or*
- *$\exists s_0, m'', n''$ such that $m \xrightarrow{s_0} x + m''$ and $n \xrightarrow{s_0} x + n''$ and $s_0.s_b \in pre(s.s_b)$.*

Proof. We have an equation $m \simeq n$, with m and n in reduced normal form, for which we assume that: $m \xrightarrow{s} x + m'$ but $n \overset{s}{\not\rightarrow} x + n'$ for any n'. Let s be the shortest trace meeting the proviso of the lemma. It is not hard to see that $s \neq \varepsilon$ because $m \simeq n$ and m and n are in reduced normal form. This means that indeed in the monitors m, n all other earlier occurrences of x happen at both sides. By Corollary 4.4.2 we know that there is a prefix of s called s' $(s = s'.s'')$ such that both m and n can perform the transitions $m \xrightarrow{s'} x + m'_0$ and $n \xrightarrow{s'} x + n'_0$, and in addition for every prefix of s' we have that $n \overset{s'}{\not\rightarrow} x + n'$ and $m \overset{s'}{\not\rightarrow} x + m'$ for every m' and n'.

This means that there are no other one-sided occurrences of the variable x "between" s' and s'' (otherwise s would not be the shortest trace). Since $m \simeq n$ is sound, we know that under any substitution the resulting monitors are verdict equivalent.

Consider the set of traces

$$A = \{t \mid (|t| \leq |s''| \wedge t \notin pre(s'')) \vee t = s''.t', \ t' \in \text{ACT}^+\} \ .$$

We now associate with this set of traces the class $\mathcal{S}_A$ of substitutions σ as the ones that for at least one trace $s_p \in A$ we have that $\sigma(x) \overset{s_p}{\Longrightarrow} yes$ or $\sigma(x) \overset{s_p}{\Longrightarrow} no$. Note that the class of substitution $\mathcal{S}_A$ contains many substitution for each trace s_p and, additionally, since the set A is infinite, $\mathcal{S}_A$ is infinite as well.

Fix now a s_p and a substitution $\sigma \in \mathcal{S}_A$ such that $\sigma(x) \overset{s_p}{\Longrightarrow} yes$. We have therefore that $\sigma(m) \overset{s's_p}{\Longrightarrow} yes$, $\sigma(n) \overset{s's_p}{\Longrightarrow} yes$ and $\sigma(m) \overset{ss_p}{\Longrightarrow} yes$. By the construction of A, $s's_p$ is not a prefix of ss_p and therefore it is not necessary that $\sigma(n) \overset{ss_p}{\Longrightarrow} yes$. However, since $m \simeq n$ is sound we have that $\sigma(n)$ must also be able to accept $s.s_p$. One way this could happen is if both monitors, m and n accept and reject the trace $s_b = s.s_p$ where $s_p \in A$ independently of a substitution, i.e. $m \overset{s_b}{\Longrightarrow} yes$, $m \overset{s_b}{\Longrightarrow} no$, $n \overset{s_b}{\Longrightarrow} yes$ and $n \overset{s_b}{\Longrightarrow} no$. Note that if one monitor can perform these transitions independently of a substitution then the other one must do so as well since they are verdict equivalent. If this is the case then for the traces s', s'' with $s = s's''$ an for all $s_b = ss_p$ where $s_p \notin pre(s'')$ the first bullet of the lemma holds.

If this is not the case however we have that for a trace s_p and a substitution $\sigma \in \mathcal{S}_A$ such that $\sigma(x) \overset{s_p}{\Longrightarrow} yes$ the monitor n must somehow accept the trace ss_p and this is not done because $n \overset{s_b}{\Longrightarrow} yes$.

We remind to the reader here that s is the shortest we could find that satisfied the proviso of the lemma. Therefore there are no other one-sided variable occurrences along the trace s.

This means that the only way than n could accept s_b is another variable occurrence (not one-sided as s is the shortest trace satisfying the proviso of the the lemma) happening after some other prefix s_0 of s. I.e. $n \xrightarrow{s_0} x + n_1$, $m \xrightarrow{s_0} x + m_1$ for some monitors n_1 and m_1 and trace $s_0.s_p$ is a prefix of $s'.s''.s_p = s.s_p$. Note here that by Corollary 4.4.2 we know that s' is the shortest trace after which the variable x occurs. Therefore our only options for the trace s_0 would be the trace s' and its extensions which falls in the second case of the lemma as $s_0.s_p$ is a prefix of $s.s_p$.

This concludes the case analysis for the shortest s leading to a one-sided variable occurrence of a variable. We continue with a trace s_1 as the immediately longer than s. For this s_1 with $|s_1| \geq |s|$ we can generalize the result as follows:

If $s \in pre(s_1)$ then the trace s' we identified with the case analysis s is also a prefix of s_1 (i.e. $s_1 = s'.s_1'')$ and the same transitions we proved for the traces s_b are also enough for the result to hold for the trace s_1. Assume now that $s \notin pre(s_1)$. Then Corollary 4.4.2 still holds and the one-sided variable occurrence after the trace s_1 also does not have any other one-sided variable occurrences between itself and the prefix guaranteed by the corollary which means we can apply the same analysis. □

Completeness of verdict equivalence

In this section we will present our axiom system for open monitors over a finite number of actions. We start by providing an axiom set, which we prove to be sound and complete for verdict equivalence over Mon_F. In order to do so, we first use these axioms to further reduce a normal form of a term. Then, by utilizing this new reduced normal form we use structural induction to prove the completeness of our axiom set. The axiom set we provide is infinite. It is therefore natural to ask whether $\simeq$ is finitely axiomatizable over Mon_F. We answer this question negatively by proving that no complete finite axiom set exists for this algebra. This final part follows a different type of argument which we will present in Section 4.5.

When studying open equations over a finite set of actions one would hope that one of the axiom systems presented already would be complete. However, we can guarantee that the equations provided in $\mathcal{E}'_v$ are definitely unable to prove every sound open equation. To see this consider the equation used in Example 11 (where k is a constant):

$$x + a^k.x + \overline{a^k}^{(3)}(yes + no) \simeq x + \overline{a^k}^{(3)}(yes + no) \ .$$

We can clearly see that one of the sides of this equations contains a one-sided variable occurrence (remember that we are considering terms up to $A1 - A4$). The only axiom which has a similar behavior is $O1$. However for axiom $O1$ to be applied it must be the case that a variable is occurring simultaneously with a *yes and* a *no* verdict. Since this does not apply for the equation we are examining it is easy to see that no proof involving only the axioms of $\mathcal{E}'_v$ could prove it.

Towards proving this kind of equations and when Act is finite, we consider the family of axioms

$$\mathcal{O} = \{O2_{s,k} \mid s \in \textsc{Act}^*, k \geq 0\}$$

where

$$(\mathbf{O2_{s,k}}) \quad x + s.x + \overline{s}^{(k)}(yes + no) = x + \overline{s}^{(k)}(yes + no) \ .$$

We extend our finite axiom set $\mathcal{E}'_v$ for open terms to the infinite $\mathcal{E}'_v \cup \mathcal{O}$, which we will call $\mathcal{E}'_{v,f}$. The subscript f in the naming scheme states that the action set for which the axiom system is complete is finite. When the action set is a singleton, we will replace it with the subscript 1. If the cardinality of the action set is not important, or if it is infinite, then we use no subscript. Based on the naming scheme we have defined, the name of the axiom set $\mathcal{E}'_{v,f}$ denotes that we are studying verdict equivalence ($_v$), over open terms ($'$) and for a finite set of actions ($_f$).

Lemma 4.12. *$\mathcal{E}'_{v,f}$ is sound. That is, if $\mathcal{E}'_{v,f} \vdash m = n$ then $m \simeq n$, for all $m, n \in Mon_F$.*

Proof. We have to prove soundness only for the new family of equations $\mathcal{O}$ as the other equations are sound by Theorem 4.1.

First of all, note that $\sigma(x + s.x + \overline{s}^{(k)}(yes + no))$ accepts every trace accepted by $\sigma(x + \overline{s}^{(k)}(yes + no))$, and rejects every trace rejected by $\sigma(x + \overline{s}^{(k)}(yes + no))$. We are therefore left to show that

- if $\sigma(s.x)$ accepts some trace then so does $\sigma(x + \overline{s}^{(k)}(yes + no))$, and
- if $\sigma(s.x)$ rejects some trace then so does $\sigma(x + \overline{s}^{(k)}(yes + no))$.

We only detail the proof for the latter claim, as that of the former one is similar. To this end, assume that $\sigma(s.x)$ rejects some trace s'. Then $s' = ss''$ for some s'' that is rejected by $\sigma(s.x)$. If s'' is a prefix of s^k, then it is not hard to see that $\sigma(x)$ rejects s' too, and thus so does $\sigma(x + \overline{s}^{(k)}(\mathit{yes} + \mathit{no}))$. On the other hand, if s'' is not a prefix of s^k, then $s' = ss''$ is not a prefix of s^k either. Therefore, $\sigma(\overline{s}^{(k)}(\mathit{yes} + \mathit{no}))$ rejects s'. It follows that $\sigma(x + \overline{s}^{(k)}(\mathit{yes} + \mathit{no}))$ rejects s', and we are done. □

We provide here some examples of how to use the above to derive some simpler and more intuitive sound equations.

Lemma 4.13. *The following equations are derivable from $\mathcal{O}$ for each $s, s_1 \in \textsc{Act}^*$:*

1. $x + s.x + s.\overline{s_0}(\mathit{yes} + \mathit{no}) = x + s.\overline{s_0}(\mathit{yes} + \mathit{no})$, *with s_0 a prefix of s,*
2. $\mathit{yes} + x + s_1.\overline{s_2}(\mathit{no}) = \mathit{yes} + x + s_1.\overline{s_2}(\mathit{no}) + s_1.x$, *where s_2 is any prefix of s_1,*
3. $\mathit{no} + x + s_1.\overline{s_2}(\mathit{yes}) = \mathit{no} + x + s_1.\overline{s_2}(\mathit{yes}) + s_1.x$, *where s_2 is any prefix of s_1,*
4. $x + s.\sum_{a\in\textsc{Act}} a.(\mathit{no} + \mathit{yes}) = x + s.(x + \sum_{a\in\textsc{Act}} a.(\mathit{no} + \mathit{yes}))$.

Proof. We first show how to derive the first equation and then we derive the rest from it. We start by picking the equation $O2_{s,1}$ i.e.

$$x + s.x + s.\overline{s}^{\leq}(\mathit{yes} + \mathit{no}) + s.s.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no}) =$$

$$x + s.\overline{s}^{\leq}(\mathit{yes} + \mathit{no}) + s.s.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})\ .$$

In addition we have the tautology

$$s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no}) = s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})\ ,$$

for the specific prefix s_0 of s. On the two valid above equations we apply the congruence rule for + and have:

$$x + s.x + s.\overline{s}^{\leq}(\mathit{yes} + \mathit{no}) + s.s.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no}) + s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})$$

$$= x + s.\overline{s}^{\leq}(\mathit{yes} + \mathit{no}) + s.s.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no}) + s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})\ .$$

The first simplification that we perform now is by observing that the summand $s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})$ accepts and rejects a prefix of the whole summand $s.s.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})$ and therefore we can eliminate the latter from the summation:

$$x + s.x + s.\overline{s}^{\leq}(\mathit{yes} + \mathit{no}) + s.s_0.\sum_{a\in\textsc{Act}} a.(\mathit{yes} + \mathit{no})$$

$$= x + s.\overline{s}^{\leq}(yes + no) + s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no)\ .$$

In addition the term $s.\overline{s}^{\leq}$ can be rewritten as $s.\overline{s_0}^{\leq}(yes + no) + s.s_0.\overline{s_1}(yes + no)$ with $s = s_0.s_1$. To see this, consider that the traces up to length $|s|$ that do not cause a rejection of the trace s are the ones that do not cause a rejection of its prefix s_0 and the ones that start with s_0 but do not cause the rejection of its continuation s_1. Thus we have:

$$x + s.x + s.\overline{s_0}^{\leq}(yes + no) + s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no) + s.s_0.\overline{s_1}(yes + no)$$

$$= x + s.\overline{s_0}^{\leq}(yes + no) + s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no) + s.s_0.\overline{s_1}(yes + no)\ .$$

Now we have again that the summand $s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no)$ accepts and rejects a prefix of the whole summand $s.s_0.\overline{s_1}(yes + no)$ and therefore we can omit the latter. This gives us the equation:

$$x + s.x + s.\overline{s_0}^{\leq}(yes + no) + s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no) =$$

$$x + s.\overline{s_0}^{\leq}(yes + no) + s.s_0. \sum_{a \in \textsc{Act}} a.(yes + no)\ ,$$

which can be rewritten using our notation as

$$x + s.x + s.\overline{s_0}(yes + no) = x + s.\overline{s_0}(yes + no)\ ,$$

giving us the target equation.

Having presented the proof for the first family of equations in detail we give a short description for the rest. For the equations (2) and (3) it suffices to use the congruence rule for + with the equations $yes = yes$ and $no = no$ respectively and then simplify the equations by using the distribution axiom for +. For the latter equation (4) it is enough to instantiate the prefix s_0 in the the family of equations (1) as the empty string ε. This is, of course, allowed since the empty string is a prefix of any string. □

Now that we have discussed the family of axioms $\mathcal{O}$, we proceed to use them in defining a notion of reduced normal form that is suitable for monitors over a finite action set.

Definition 4.9. *A **finite-action-set reduced normal form** is a term*

$$m = \sum_{a \in A} a.m_a + \sum_{i \in I} x_i\ [+yes]\ [+no]$$

where each m_a is different from end and if $v \in \{yes, no\}$ is a summand of m then each m_a is v-free, and in reduced normal form. If both yes and no are summands *of m then m is equal to $yes + no$. In addition for every trace s, if there exists a k such that for all the traces s_0*

$$\overline{s}^{(k)}(yes + no) \xrightarrow{s_0} yes + no, \text{ implies: } m \overset{s_0}{\Rightarrow} yes \text{ and } m \overset{s_0}{\Rightarrow} no$$

then $m \overset{s}{\not\rightarrow} x_i + m'$ for all $i \in I$ and m'.

In order to use the above form of the monitors in Mon_F we need to prove that any term can be rewritten in a reduced normal form using the axioms in $\mathcal{E}'_{v,f}$. Before doing so we will prove the following useful lemma, which only uses axioms form $\mathcal{E}_v$.

Lemma 4.14. *For a monitor $m \in Mon_F$:*

- *if $m \overset{s}{\Rightarrow} yes$ then $\mathcal{E}_v \vdash m = m + s.yes$ and*
- *if $m \overset{s}{\Rightarrow} no$ then $\mathcal{E}_v \vdash m = m + s.no$.*

Proof. We prove both statements by induction on the length of the trace s and limit ourselves to presenting the proof for the first one.

- If s is the empty trace, then m accepts the empty trace. Therefore it must contain a *yes* syntactic summand and we are done.
- Assume now that $s = a.s'$. Then $m \xrightarrow{a} m_a \xrightarrow{s'} yes$ for some m_a. By induction $\mathcal{E}'_{v,f} \vdash m_a = m_a + s'.yes$.

  Now,

$$\begin{aligned}\mathcal{E}'_{v,f} \vdash m &= m + a.m_a = m + a.(m_a + s'.yes) = m + a.m_a + a.s'.yes \\ &= m + a.s'.yes\end{aligned}$$

□

We will also need a similar result, this time involving syntactic summands that contain occurrences of variables.

Lemma 4.15. *For a monitor $m \in Mon_F$, where m is in normal form for open terms, if $m \xrightarrow{s} x + m_s$ then $\mathcal{E}_v \vdash m = m' + s.x$ where $m' \overset{s}{\not\rightarrow} x + m''$ for every m''.*

Proof. We prove the claim by induction on the length of the trace s.

- If s is the empty trace then $m \xrightarrow{\varepsilon} x + m_s = m$. This means that x is a summand of m. Since m is in normal form, m_s does not have x as a summand and we are done.
- Assume now that $s = a.s'$. Since m is in normal form and $m \xrightarrow{a.s'} s + m'$, we have that $m = m' + a.m_a$ for some $m' \overset{a}{\not\rightarrow}$ and m_a in formal form such that $m_a \xrightarrow{s'} x + m'$. By the induction hypothesis, $\mathcal{E}_v \vdash m_a = m'_a + s'.x$ where $m'_a \overset{s'}{\not\rightarrow} x + m'_{s'}$ for every $m'_{s'}$.

Therefore we have:

$$m = m' + a.m_a = m + a.(m'_a + s'.x) = m' + a.m'_a + a.s'.x \ ,$$

and since $m'_a \overset{s'}{\not\rightarrow} x{+}m'_{s'}$ for every $m'_{s'}$ and $m' \overset{a}{\not\rightarrow}$ we have that $m = s.x{+}m_{rest}$ with $m_{rest} \overset{s}{\not\rightarrow} x + m''$ for every m'' and we are done. □

□

Lemma 4.16. *Each open monitor $m \in Mon_F$, is provably equal to a reduced normal form using $\mathcal{E}'_{v,f}$.*

Proof. From Lemma 4.8 we can start from a monitor m already in open normal form, as given in Definition 4.6. Therefore we have the following cases:

- $m = yes + no$.
- $m = yes + \sum\limits_{a \in A} a.m_a + \sum\limits_{i \in I} x_i$, where each m_a is *yes*-free.
- $m = no + \sum\limits_{a \in A} a.m_a + \sum\limits_{i \in I} x_i$, where each m_a is *no*-free.
- $m = \sum\limits_{a \in A} a.m_a + \sum\limits_{i \in I} x_i$.

We begin our analysis from the second case. A similar analysis can be applied to the third one and the fourth one follows by a simpler version of the same inductive argument. We have therefore a monitor $m = yes + \sum\limits_{a \in A} a.m_a + \sum\limits_{i \in I} x_i$. The extra claim for these reduced normal forms is that if for some trace s, and a k_0, for all traces s_0,

$$\overline{s}^{(k_0)}(yes + no) \xrightarrow{s_0} yes + no, \text{ implies: } m \overset{s_0}{\not\Rightarrow} yes \text{ and } m \overset{s_0}{\not\Rightarrow} no$$

then $m \overset{s}{\not\rightarrow} x_i + m_x$ for all $i \in I$ and m_x. In order to prove this extra constraint we assume the premise is true. We will show that we can reduce m to m_{red} with $\mathcal{E}_{fin} \vdash m = m_{red}$ and $m_{red} \overset{s}{\not\rightarrow} x_i + m_x$ for every $i \in I$ and every m_x.

Since m accepts and rejects all the traces that $\overline{s}^{(k_0)}(yes + no)$ accepts and rejects, we have that $m \simeq m + m'$ and that $m' \xrightarrow{s_0} yes + no$ for all of the traces s_0 that $\overline{s}^{(k_0)}(yes{+}no) \xrightarrow{s_0} yes{+}no$. We call this set of traces $\mathcal{S}$ which is finite since k_0 is fixed. Therefore by Lemma 4.14 we have that $\mathcal{E}'_{v,f} \vdash m = m + \sum\limits_{s_0 \in \mathcal{S}} s_0.(yes + no)$. Since the term $\sum\limits_{s_0 \in \mathcal{S}} s_0.(yes + no)$ is verdict equivalent to $\overline{s}^{(k_0)}(yes + no)$ and both terms are closed, we have that by Theorem 4.2, $\mathcal{E}_v \vdash \sum_{s_0 \in \mathcal{S}} s_0.(yes{+}no) = \overline{s}^{(k_0)}(yes{+}no)$. Therefore $\mathcal{E}'_{v,f} \vdash m = m + \overline{s}^{(k_0)}(yes + no)$, which means

$$\mathcal{E}'_{v,f} \vdash m = yes + \sum_{a \in A} a.m_a + \sum_{i \in I} x_i + \overline{s}^{(k_0)}(yes + no).$$

For the same monitor m we now want to argue that if $m \xrightarrow{s} x_i$ for one of the variables in $\{x_i \mid i \in I\}$ then we can eliminate this occurrence.

Since m is in reduced normal form we have by Lemma 4.15 that $m = m' + s.x_i$ where $m' \not\xrightarrow{s} x_i$. Additionally we have shown that $m = m + \overline{s}^{(k_0)}(yes + no)$ which implies $m = m' + s.x_i + \overline{s}^{(k_0)}(yes + no)$ with $m' \not\xrightarrow{s} x_i$. Since x_i is one of the variables that appear as summands of m we can successfully apply the axiom $O2_{s,k_0}$ for each variable and we have the that indeed m reduces to a monitor m_{red} such that $m_{red} \not\xrightarrow{s} x_i + m_{x_i}$ for every $i \in I$ and every m_{x_i}. □

Lemma 4.17. *If monitor $m \in Mon_F$, with $|\text{ACT}| \geq 2$ is in reduced normal form and contains an x summand and $m \xrightarrow{s} x + m'$ for some m' then there is at least one trace s_{bad} such that for every k,*

$$\overline{s}^{(k)}(yes + no) \xRightarrow{s_{bad}} yes \text{ and } \overline{s}^{(k)}(yes + no) \xRightarrow{s_{bad}} no$$

but

$$m \not\xRightarrow{s_{bad}} yes \text{ or } m \not\xRightarrow{s_{bad}} no.$$

Proof. We can easily show that for each k there exists an s_k such that $\overline{s}^{(k)}(yes + no) \xrightarrow{s_k} yes + no$ but $m \not\xRightarrow{s_k} yes$ or $m \not\xRightarrow{s_k} no$. This follows since if this were not the case then for some k_0, no such trace s_{k_0} exists. Thus the monitor would contain a summand $m' \simeq \overline{s}^{(k_0)}(yes + no)$ for this k_0 and still it would be able to perform the transition $m \xrightarrow{s} x + m'$ which contradicts the assumption that m is in reduced normal form.

We will now show that one trace s_{bad} suffices for all k. To that end, consider the term $\overline{s}^{(1)}(yes + no)$. If there is an s_1, which is not a prefix of ss and $m \not\xRightarrow{s_1} yes$ or $m \not\xRightarrow{s_1} no$ then for $s_{bad} = s_1$ we have that for all k, $\overline{s}^{(k)}(yes + no) \xRightarrow{s_{bad}} yes$ and $\overline{s}^{(k)}(yes + no) \xRightarrow{s_{bad}} no$ and we are done. If this is not the case and since the trace s_1 is guaranteed to exist (by the previous paragraph) then it must be an extension of ss. Again if s_1 is not prefix of sss then again for $s_{bad} = s_1$ we have the necessary conclusion.

Otherwise $m \not\xRightarrow{s_1} yes$ or $m \not\xRightarrow{s_1} no$ for the trace $s_1 = ssa$, where a is the first action of s. Therefore by the definition of $\overline{s}^{(k)}(yes + no)$ we have that for all s_b such that $\overline{s}^{(k)}(yes{+}no) \xrightarrow{s_b} yes{+}no$ and $k > 1$ we have that $m \not\xRightarrow{s_b} yes$ or $m \not\xRightarrow{s_b} no$. This allows us to look for an s_{bad} which will also cover the case $k = 1$ in larger terms.

We then apply the same reasoning for $k = 2, \dots$ up to a certain k_b. If at any point in the process we encounter a trace s_i which fulfills our premise then we can stop. We are just left to show that this process will eventually terminate.

This can be shown as follows. Recall that every monitor m has a finite depth depth (m) (see Def. 4.2). We now take a k_b large enough so that $s^{k_b} >$ depth (m). If the iterative procedure described above reaches this k_b we have that $m \not\overset{s_{bad}}{\Longrightarrow} yes$ or $m \not\overset{s_{bad}}{\Longrightarrow} no$ for the trace $s^{k_b+1}a$ where a is the first action of s. However since the depth of the monitor m is smaller that the length of this trace we also have that the monitor cannot accept or reject any of its extensions.

Therefore for the extension $s_{bad} = s^{k_b+1}ac$ where c is not the second action of s we have that for all $k > k_b$, $\overline{s}^{(k)}(yes + no) \overset{s_{bad}}{\Longrightarrow} yes$ and $\overline{s}^{(k)}(yes + no) \overset{s_{bad}}{\Longrightarrow} no$ while m
$not \overset{s_{bad}}{\Longrightarrow} yes$ or $m \not\overset{s_{bad}}{\Longrightarrow} no$. Additionally since the iterative procedure we described above reached this k_b we have that for all $i \leq j \leq k_b$, it is true that $\overline{s}^{(k)}(yes+no) \overset{s_j}{\Rightarrow} yes$ and $\overline{s}^{(k)}(yes + no) \overset{s_j}{\Rightarrow} no$, which concludes the proof. □

The two lemmata above play a key role in the completeness proof we will present now.

We distinguish two cases separately, namely when $|\textsc{Act}| \geq 2$ and when $\textsc{Act}$ is a singleton. This is necessary because equations such as $x = x + a.x$ are only sound when $\textsc{Act} = \{a\}$. For the proof when $|\textsc{Act}| \geq 2$ it is necessary to utilize at least two actions $a, b \in \textsc{Act}$, which is the reason why when only one action is available new cases arise.

Action set with at least two actions We have already shown the soundness of the axiom system $\mathcal{E}'_{v,f}$. We now proceed to show completeness.

For each such completeness theorem we follow a similar general strategy in order to prove that two arbitrary verdict equivalent monitors have identical reduced normal forms. To that end, we prove that they have identical variables as summands, that the sets of initial actions that each one can perform are equal and that after a common action they reach monitors that are also verdict equivalent. Unfortunately, for a finite set of actions, we were not able to define a substitution that would cover all the three above-mentioned steps like we did when the set of actions was infinite. We therefore adopted a proof strategy that focuses on each part of the proof separately.

Theorem 4.5. *$\mathcal{E}'_{v,f}$ is complete for open terms for finite Act with $|\textsc{Act}| \geq 2$. That is, if $m \simeq n$ then $\mathcal{E}'_{v,f} \vdash m = n$.*

Proof. By Lemma 4.16 we may assume that m and m are in reduced normal form. We prove the claim by induction on the sum of the sizes of m and n, and proceed with a case analysis on the form m may have.

In the case where m contains both a yes and a no summand then both m and n must be equal to $yes + no$ as they are in reduced normal form.

Assume now that

$$m = \mathit{yes} + \sum_{a\in A} a.m_a + \sum_{i\in I} x_i\ ,$$

where $\{x_i \mid i \in I\}$ is the set of variables occurring as summands of m and each m_a is *yes*-free and different from *end* (as a reduced normal form). Since $\sigma(m)$ accepts ε for each σ and $m \simeq n$, monitor n is bound to have a similar form since it must contain the verdict *yes* as a summand (but not a *no* one). Therefore:

$$n = \mathit{yes} + \sum_{b\in B} b.n_b + \sum_{j\in J} y_j$$

and we need to show that there is a way to apply our axioms to show that monitor n is provably equal to m.

We start by proving that $\{x_i \mid i \in I\} = \{y_j \mid j \in J\}$. By symmetry, it suffices to show that $\{x_i \mid i \in I\} \subseteq \{y_j \mid j \in J\}$. To this end, assume $x \in \{x_i \mid i \in I\}$. Consider the substitution σ mapping x to *no* and every other variable to *end*, i.e:

$$\sigma(y) = \begin{cases} \mathit{no}, & \mathit{if}\ y = x \\ \mathit{end}, & \mathit{otherwise.} \end{cases}$$

Then, $\sigma(m)$ rejects the empty trace ε. Since $\sigma(m) \simeq \sigma(n)$, we have that $\sigma(n)$ must also reject ε. By the form of n and the definition of σ, this is only possible if n has x as a summand, and we are done. Therefore the set of variables of m is a subset of the variables of n.

Next, we prove that the action sets A, B are identical. Assume that $a \in A$. Since $\textsc{Act}$ contains at least two actions, there is some action $b \neq a$. Consider the substitution σ_1 defined by $\sigma_1(x) = b.\mathit{no}$ for each $x \in \mathcal{V}$. Since $a \in A$ and m_a is *yes*-free and different from *end*, it is easy to see that there exists an $s \in \textsc{Act}^*$ such that $as \in L_r(\sigma_1(m))$. Since $m \simeq n$ we have that $\sigma_1(m) \simeq \sigma_1(n)$ and therefore $\sigma_1(n)$ must also reject as. By the form of n and the definition of σ, this is only possible if $n \xrightarrow{a} n_a$ for some n_a and therefore $a \in B$. Hence, $A \subseteq B$ and the claim follows by symmetry.

For the final part of the proof we must show that $m_a \simeq n_a$ for each $a \in A$, which is enough to complete the proof, by the induction hypothesis. Towards a contradiction we will assume that the two monitors m_a, n_a are not verdict equivalent. Therefore there exists a substitution σ_0 that separates them, that is without loss of generality, there is a trace s_0 such that $s_0 \in L_r(\sigma_0(m_a)), s_0 \notin L_r(\sigma_0(n_a))$ or there is some $s_0 \in L_a(\sigma_0(m_a)), s_0 \notin L_a(\sigma_0(n_a))$.

We will analyze first the case of rejection of the string s_0. The substitution σ_0 must be a closed one for m_a, n_a i.e. it must map to a closed monitor all variables in $(Var(m_a) \cup Var(n_a))$. We will use this substitution to create a new one σ_{bad} that would also separate the original monitors m, n.

The first step towards this is:

$$\sigma_{bad}(x) = \begin{cases} \mathit{end},\ \text{if } x \in Var(m) \setminus (Var(m_a) \cup Var(n_a)), \\ \sigma_0(x),\ \mathit{otherwise.} \end{cases}$$

Now since $s_0 \in L_r(\sigma_0(m_a))$ and $\sigma_{bad}(m_a) = \sigma_0(m_a)$ we also know that $a.s_0 \in L_r(\sigma_{bad}(a.m_a))$. Our aim is to show that $a.s_0 \notin L_r(\sigma_{bad}(n))$. Following the definition $\sigma_{bad}(n_a) = \sigma_0(n_a)$ and therefore $s_0 \notin L_r(\sigma_{bad}(n_a))$.

Hence, the only way for $\sigma_{bad}(n)$ to reject $a.s_0$, like $\sigma_{bad}(m)$ does, is if it was rejected by the mapping of one the variables contained in the set $\{x_i \mid i \in I\}$.

It is useful to make here apparent that in order for $\sigma_{bad}(n)$ to reject $a.s_0$, it must do so completely independently of the summand $\sigma_{bad}(a.n_a)$, since the latter cannot reject any of the prefixes of $a.s_0$ as well. Even in the case where s_0 starts with a, and $\sigma_0(n_a)$ rejects some $a.s_1.s_2.\ldots s_{n-i}$ it would still be impossible for $\sigma_0(n_0)$ to reject $a.s_0$ since the assumption that $a.s_0 = a.a.s_1.s_2.\ldots s_{n-1}$ would automatically imply that $\sigma_0(n_a)$ rejects some prefix of s_0 which is a contradiction.

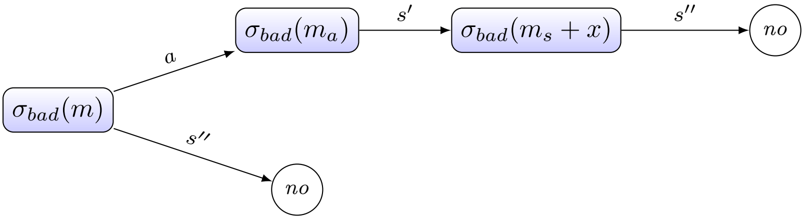

Figure 4.1: Transitions the monitor $\sigma_{bad}(m)$ can perform

By the definition of σ_{bad}, the variables that did not appear at all in n_a or m_a were mapped to *end* and therefore cannot reject any string. Therefore the only way for n to reject $a.s_0$ is for one of the variables appearing in $Var(n_a) \cup Var(m_a)$ to have been mapped to a closed term that can reject $a.s_0$. (Note that this does not contradict the fact that $\sigma_{bad}(n_a)$ does not reject s_0). Therefore there is at least one $x_0 \in Var(m_a) \cup Var(n_a)$ and $x_0 \in \{x_i \mid i \in I\}$ such that $as_0 \in L_r(\sigma_{bad}(x_0))$.

This leads to the case were m, n reject a prefix of as_0 because of the mapping of x_0. However this implies that we have the following situation:

$$m = yes + x_0 + a.m_a + \sum_{b \in A \setminus \{a\}} b.m_b + \sum_{i \in I \setminus \{0\}} x_i \simeq$$

$$yes + x_0 + a.n_a + \sum_{b \in A \setminus \{a\}} b.n_b + \sum_{i \in I \setminus \{0\}} x_i = n$$

and that the monitor m_a can perform the transitions: $m_a \xrightarrow{s'} m'_a + x_0$ and the monitor $\sigma_0(x_0) = \sigma_{bad}(x_0)$ respectively can perform the transitions: $\sigma_{bad}(x_0) \xrightarrow{s''}$ *no* , where s' is a prefix of s_0 (i.e. $s_0 = s'.s''$) and in addition $n_a \overset{s'}{\not\rightarrow} x + n'$ for any n'. This means respectively that $m \xrightarrow{as'} m'_a + x_0$ and $\sigma_{bad}(m'_a + x_0) \xrightarrow{s''} no$.

By Lemma 4.17 we have that there exists at least one trace s_b such that $m \overset{s_b}{\not\Rightarrow}$ *yes* or $m \overset{s_b}{\not\Rightarrow}$ *no* but $s_b \in L_r(\overline{as'}^{(k)}(yes + no))$ for all $k \geq 0$. Since m contains a *yes* summand we have that it must be the case that $m \overset{s_b}{\not\Rightarrow} no$. We now, further modify σ_{bad} to map the variable x_0 to $s_b.no$ and any other variable $y \neq x_0$ to *end*. We have then that s_b and $as'.s_b \in L_r(\sigma_{bad}(m))$. In addition $s_b \in L_r(\sigma_{bad}(n))$. However the traces that are rejected by the term $\overline{as'}^{(k)}$, by definition, are exactly the traces such that their rejection does not cause a rejection of the as' trace. This means that under the modified substitution σ_{bad}, monitor n *cannot* reject the trace $as'.s_b$. This deems the monitors m, n not verdict equivalent, which contradicts our assumption. We conclude then that the rejection set of m_a is equal to the rejection set of n_a for each $a \in A$.

It remains to show that m_a and n_a also have identical acceptance sets. Towards a contradiction, assume they do not and take a trace s that under some substitution σ_0 separates them, i.e. $s \in L_a(\sigma_0(m_a))$ and $s \notin L_a(\sigma_0(n_a))$. In addition, assume that s is of minimum length, meaning that no prefix of s (under any substitution) has the property of separating the acceptance sets of m_a and n_a. This fact in addition to m_a and n_a being *yes*-free (as a result of m and n being in reduced normal form) means that the acceptance of s by m_a is the result of a variable x occurring in m_a as $m_a \xrightarrow{s} x + m'$ for some m'. Since however the assumption is that $s \notin L_a(\sigma_0(n_a))$ we have that $n_a \not\xrightarrow{s} x + n'$ for any n'. We know that this is exactly the case since if the variable x occurred earlier in m_a then by mapping it to *yes* we would have a shorter trace being accepted by $\sigma_0(m_a)$ but not $\sigma_0(n_a)$.

We are sure now that monitor $\sigma_0(n_a)$ cannot perform the transition $\sigma_0(n_a) \overset{s}{\Longrightarrow}$ *yes*, which means that not only it does not arrive at the variable x after reading the trace s, but also does not arrive to the *yes* verdict for any of its prefixes (say s') as that would imply that it can reach the *yes* verdict for s as well.

Finally, by n being in reduced normal form, and by m_a not arriving at a *no* verdict for any of the prefixes s' of s (as this would mean that if becomes a *no* and therefore cannot perform the transitions $m_a \xrightarrow{s} x$) we know that n_a does not arrive to the *no* verdict after reading the trace s or any of its prefixes either.

Given all of the above we can now construct the substitution σ_{bad} that would separate the rejection sets of n_a, m_a which is enough to prove the contradiction as the case where such a substitution exists and separates the rejection sets of the two sub-monitors has already been covered. The situation we have at hand is as follows:

Monitor $\sigma_0(m_a)$ can arrive to the verdict *yes* after reading the trace s while $\sigma_0(n_a)$ cannot and also neither n_a nor m_a can produce a *no* verdict for the trace s. Therefore if we switch the mapping of x to *no* in σ' and the verdicts of all other variables that where mapped to a *no* verdict to *end* we have produced a substitution that causes s to be rejected by $\sigma'(m_a)$ but not from $\sigma'(n_a)$. By utilizing our previous construction there exists another one that separates the monitors n, m as well which is a contradiction.

We have concluded then that the $L_a(m_a) = L_a(n_a)$ and $L_r(m_a) = L_r(n_a)$ which means that they are verdict equivalent. Therefore we can apply the inductive hypothesis and have that $\mathcal{E}'_{v,f} \vdash m_a = n_a$. Using now congruence rules we have that $\mathcal{E}'_{v,f} \vdash m = n$. All other possible forms of monitors m, n are sub-cases that the relative analysis can be applied symmetrically and therefore they are omitted. □

Singleton Action Set We proceed now with the analysis of the completeness result when $\textsc{Act} = \{a\}$.

As we mentioned earlier, when a is the only action, the equation

$$(V_1) \quad x = x + a.x$$

is sound, but cannot be proved from the equations in $\mathcal{E}'_{v,f}$ over $\{a\}$. Indeed, unlike V_1, all the equations in E_v are sound regardless of the cardinality of the action set and those in the family $\mathcal{O}$ introduce subterms of the form $yes + no$, which can never be removed in equational derivations.

Theorem 4.6. *The finite axiom system $\mathcal{E}'_{v,1} = \mathcal{E}'_v \cup \{V_1\}$ is complete for verdict equivalence over open monitors when* $\textsc{Act} = \{a\}$*. That is, if $m \simeq n$ then $\mathcal{E}'_{v,1} \vdash m = n$. Hence, verdict equivalence is finitely based when* $\textsc{Act} = \{a\}$.

Proof. Before we start the main proof we note that the new axiom V_1 can prove the equation $x = a^n.x + x$ for each $n \geq 0$. This is done as follows: if $n = 0$ then this is the axiom $A3$. Assume we can prove that equation for n. Then we can show it for $n + 1$ thus:

$$x \overset{\mathrm{V_1}}{=} x + a.x \overset{\mathrm{I.H.}}{=} x + a.(a^n.x + x) \overset{\mathrm{D_a}}{=} x + a.x + a^{n+1}.x \overset{\mathrm{V_1}}{=} x + a^{n+1}.x \quad .$$

Note here that this means that $\mathcal{E}'_v \cup \{V_1\}$ proves all the equations in $\mathcal{O}$ over $\{a\}$, which means that even though $\mathcal{E}'_v \cup \{V_1\}$ is finite, it can prove the infinite family $\mathcal{E}'_{v,f}$ over $\{a\}$.

Let $m \simeq n$. By Lemma 4.16, we can assume that m and n are in reduced normal form. We will present the argument only for the case where $m = yes + a.m_a + \sum_{i \in I} x_i$, where each m_a is yes-free, as every other case is either trivial or a sub-case of this one.

By following the reasoning of previous proofs, we have that $n = yes\ [+a.n_a] + \sum_{i \in I} x_i$.

Let us first consider the case that $a.n_a$ is not a summand of n. (Note that this is possible, as witnessed by axiom V_1.) That is

$$m = yes + a.m_a + \sum_{i \in I} x_i \simeq yes + \sum_{i \in I} x_i = n \quad .$$

Observe that, for each $s \in \textsc{Act}^*$, we have $m_a \overset{s}{\not\Rightarrow} no$. Indeed, $m_a \overset{s}{\Rightarrow} no$ would imply that m and n are not verdict equivalent under the substitution $\sigma_{end}(x) = end$ for all x. This means that m_a is both yes- and no-free. Moreover, note that the set of variables occurring in m_a is included in $\{x_i \mid i \in I\}$. To see this, assume that x occurs in m_a, but is not contained in $\{x_i \mid i \in I\}$. Consider the substitution that maps x to no and all the other variables to end. Again, we have that m rejects some trace starting with a while n cannot reject any trace, which contradicts our assumption that $m \simeq n$.

For each monitor m', we define $\mathcal{V}(m')$ as the set of pairs (s, x) such that $m' \xrightarrow{s} x + m''$ for some m''. By structural induction on m' and Lemma 4.15, one can easily prove that, when m' is *yes*- and *no*-free, $\mathcal{E}_v$ proves $m' = \sum_{(s,x)\in\mathcal{V}(m')} s.x$.

Therefore $m = yes + a. \sum_{(s,x)\in\mathcal{V}(m_a)} s.x + \sum_{i\in I} x_i$. Since the only available action in Act is a and the variables occurring in m_a also occur in $\{x_i \mid i \in I\}$, we have that by applying the equations we proved earlier by using axiom V_1 we can prove $m = yes + \sum_{i\in I} x_i = n$, and we are done.

Assume now that $a.n_a$ is a summand of n. We proceed to prove that that $m_a \simeq n_a$. In this case we have

$$m_a = \sum_{(s,x)\in\mathcal{V}(m_a)} s.x\ [+a^h.no] \text{ and } n_a = \sum_{(s,x)\in\mathcal{V}(n_a)} s.x\ [+a^k.no]\ ,$$

for some h, k.

By mapping all variables to *end* we can see that $h = k$. Additionally, for each variable s and by using the axiom V_1 we can reduce both of the above summations so that only the shortest s leading to x is kept. By Lemma 4.10, we have that, for each variable, this s is identical for both sides of the equality $m \simeq n$ and we are done. □

Completeness of ω-verdict equivalence

This section presents a complete axiomatization for ω-verdict equivalence over Mon_F. We have already presented the necessary axioms that capture ω-verdict equivalence over closed terms, as well as the necessary ones to capture equivalence of terms that include variables. We will show here that the combination of the two axiom systems is enough for completeness of ω-verdict equivalence over open terms and there is no need for extra axioms to be added. First we look at the case for a singleton action set, i.e. Act $= \{a\}$. In this case, the equation

$$(\mathbf{V1}_\omega)\ x = a.x$$

is sound and we therefore we can shrink the axiom system to:

$$\mathcal{E}'_{\omega,1} = \{A1 - A4\} \cup \{V1_\omega\} \cup \{O1\},$$

for which we prove:

Theorem 4.7. *$\mathcal{E}'_{\omega,1}$ is complete for ω-verdict equivalence for open terms for a finite* Act*, with* $|\text{Act}| = 1$*. That is, if $m \simeq_\omega n$ then $\mathcal{E}'_{\omega,1} \vdash m = n$.*

Proof. The proof of the above follows easily since, by using those equations, every term can be proved equal to one of the form $\sum_{i\in I} x_i\ [+yes]\ [+no]$, where I is empty if both *yes* and *no* are summands, and two terms of that form are ω-verdict equivalent iff they are equal modulo $A1 - A4$. Note that, in this case, there are only four congruence classes of terms, namely the ones asymptotically equivalent to subsets of *yes* and *no*, so the quotient algebra is very small and equationally well behaved. □

In the case where there the action set contains more than one action but is still finite we have a more interesting situation. We therefore define:

$$\mathcal{E}'_{\omega,f} = \mathcal{E}_\omega \cup \mathcal{E}'_{v,f} \ ,$$

for which we prove:

Theorem 4.8. *$\mathcal{E}'_{\omega,f}$ is complete for ω-verdict equivalence over open terms when* $\textsc{Act}$ *is finite and* $|\textsc{Act}| \geq 2$. *That is, if $m \simeq_\omega n$ then $\mathcal{E}'_{\omega,f} \vdash m = n$.*

The rest of this section is devoted to the proof of the above theorem. We start by showing a lemma that tells us that if two monitors are ω-verdict equivalent then they can only disagree on finitely many finite traces.

Lemma 4.18. *For two monitors in Mon_F, we have that $m \simeq_\omega n$ if and only if, for any substitution σ, the set*

$$\begin{aligned}\mathcal{S}_{m,n,\sigma} = {} & (L_a(\sigma(m)) \setminus L_a(\sigma(n))) \cup (L_r(\sigma(m)) \setminus L_r(\sigma(n))) \\ & \cup (L_a(\sigma(n)) \setminus L_a(\sigma(m))) \cup (L_r(\sigma(n)) \setminus L_r(\sigma(m)))\end{aligned}$$

is finite.

Proof. We prove both implications separately by establishing their contrapositive statements. For the implication from left to right, assume that $\mathcal{S}_{m,n,\sigma}$ is infinite. It follows that there are some σ and trace s such that $s \in \mathcal{S}_{m,n,\sigma}$ with

$$|s| > max\{\text{depth}\,(\sigma(m))\,, \text{depth}\,(\sigma(n))\}.$$

Assume, without loss of generality, that $\sigma(m)$ accepts s, but $\sigma(n)$ does not. Let $a \in \textsc{Act}$. Then sa^ω is in $L_a(\sigma(m)) \cdot \textsc{Act}^\omega$. We claim that sa^ω is not in $L_a(\sigma(n)) \cdot \textsc{Act}^\omega$. Indeed, $\sigma(n)$ does not accept any prefix of s, since it does not accept s itself, and it does not accept sa^i for any $i \geq 0$ because $|s| > \text{depth}\,(\sigma(n))$. For the implication from right to left, assume, without loss of generality, that there are some substitution sigma and some $t \in \textsc{Act}^{\{}omega$ such that t is in $L_a(\sigma(m)) \cdot \textsc{Act}^\omega$, but not in $L_a(\sigma(n)) \cdot \textsc{Act}^\omega$. Since t is in $L_a(\sigma(m)) \cdot \textsc{Act}^\omega$, we have that there are some $s \in L_a(\sigma(m))$ and u in $\textsc{Act}^\omega$ such that $t = su$. It follows that $ss' \in L_a(\sigma(m))$ for each finite prefix s' of u, but none of the ss' is contained in $L_a(\sigma(n))$. Therefore, $\mathcal{S}_{m,n,\sigma}$ is infinite, and we are done. □

We are now ready to present the proof of the main theorem of this section (Theorem 4.8).

Proof. By Lemma 4.16 we may assume without loss of generality that the monitors m and n are in finite-action-set reduced normal form (Definition 4.9).

We proceed by a case analysis on the form m and n might have and by induction on the sum of the sizes of m and n.

- Assume that $m = \mathit{yes} + \mathit{no} \simeq_\omega \sum_{a \in A} a.n_a + \sum_{j \in J} y_j = n$. First of all, note that $A = \textsc{Act}$. Indeed, assume $a \notin A$. Then, under a substitution that maps every variable to end, all infinite traces starting with a are neither accepted nor rejected by n since n cannot take an a transition and it also does not accept and reject ε. However all infinite traces (including those starting from a) are both accepted and rejected by m, which is a contradiction as we have assumed that the two monitors are ω-verdict equivalent.

  Moreover, it is not hard to see that $n_a \simeq_\omega \mathit{yes} + \mathit{no}$ holds for each $a \in \textsc{Act}$. By the induction hypothesis, $\mathcal{E}'_{\omega,f}$ proves $n_a = \mathit{yes} + \mathit{no}$, for each $a \in \textsc{Act}$. Therefore,

$$\mathcal{E}'_{\omega,f} \vdash n = \sum_{a \in \textsc{Act}} a.(\mathit{yes} + \mathit{no}) + \sum_{j \in J} y_j \overset{\mathrm{D_a}}{=} \sum_{a \in \textsc{Act}} a.\mathit{yes} + \sum_{a \in \textsc{Act}} a.\mathit{no} + \sum_{j \in J} y_j$$

$$\overset{\mathrm{Y}_\omega,\mathrm{N}_\omega}{=} \mathit{yes} + \mathit{no} + \sum_{j \in J} y_j \overset{\mathrm{O1}}{=} \mathit{yes} + \mathit{no} \quad ,$$

  and we are done.

- Now, we assume that $m = \mathit{yes} + \mathit{no} \simeq_\omega \sum_{a \in A} a.n_a + \sum_{j \in J} y_j + \mathit{yes} = n$, with each n_a being *yes*- and *end*-free. As above $A = \textsc{Act}$. Moreover, for each $a \in \textsc{Act}$, $L_r(n_a) \cdot \textsc{Act}^\omega = \textsc{Act}^\omega$. Following the same argument as above only for the *no* verdict we conclude that

$$\mathcal{E}'_{\omega,f} \vdash n = \mathit{yes} + \sum_{a \in \textsc{Act}} a.\mathit{no} + \sum_{j \in J} y_j = \mathit{yes} + \mathit{no} = m.$$

- The case $m = \mathit{yes} + \mathit{no} \simeq_\omega \sum_{a \in A} a.n_a + \sum_{j \in J} y_j + \mathit{no} = n$ is symmetrical to the previous one.

- The final case whose proof we present in detail is when

$$m = \mathit{yes} + \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \simeq_\omega \sum_{b \in B} b.n_b + \sum_{j \in J} y_j \ [+\mathit{yes}] \ [+\mathit{no}] = n \ ,$$

  where each side is in reduced normal form. To deal with this case, we note, first of all, that by mimicking the argument in the first case of the proof, we can prove that $\mathcal{E}'_{\omega,f} \vdash n = \mathit{yes} + \sum_{b \in B'} b.n'_b + \sum_{j \in J} y_j$, where now each n'_b is *yes*-free. By the same argument as for the verdict equivalence case (Proof of Theorem 4.5) and by defining the appropriate substitutions σ we can infer that $A = B'$ and $\{x_i \mid i \in I\} = \{y_j \mid j \in J\}$. In other words, we have:

$$m = \mathit{yes} + \sum_{a \in A} a.m_a + \sum_{i \in I} x_i \simeq_\omega \mathit{yes} + \sum_{a \in A} a.n'_a + \sum_{i \in I} x_i \quad ,$$

where m and n are in finite-action-set reduced normal form for open terms. It remains to show that under every substitution σ we have that $\sigma(m_a) \simeq_\omega \sigma(n'_a)$ so that we can apply our induction hypothesis and complete the proof.

Towards a contradiction assume that this is not the case. Therefore there exists a substitution σ for which there is at least one infinite trace s such that, without loss of generality, $s \in L_r(\sigma(m_a)) \cdot \text{ACT}^\omega$ but $s \notin L_r(\sigma(n'_a)) \cdot \text{ACT}^\omega$ or $s \in L_a(\sigma(m_a)) \cdot \text{ACT}^\omega$ but $s \notin L_a(\sigma(n'_a)) \cdot \text{ACT}^\omega$. We examine first the case of the rejection sets. Since $\sigma(m_a)$ rejects the infinite trace s, there is some finite prefix s_0 of s that is rejected by $\sigma(m_a)$. Note that $\sigma(m_a)$ will also reject all the finite prefixes of s that extend s_0. On the other hand, $\sigma(n'_a)$ does not reject any of those because it does not reject s.

As we saw in the proof of Theorem 4.5 this substitution and any such trace s_0 can be modified to a new substitution σ' such that $\sigma'(m) \not\simeq \sigma'(n)$ and consequently m is not verdict equivalent to n. Specifically from the proof of Theorem 4.5 we have that:

- Under the substitution σ', all variables except x are mapped to *end*.
- $m_a \xrightarrow{s'} x + m'_a$ for some m'_a and a trace s' that is a prefix of s_0.
- $n'_a \overset{s'}{\not\longrightarrow} x + n''_a$ for any n''_a
- The variable x is mapped to $s_b.no$ for a trace s_b such that m rejects the trace $as's_b$, but n does not.

By Lemma 4.18 we have that the only way m can be ω-verdict equivalent to n is if the number of traces they disagree on, under any substitution (including σ'), is finite. Since monitor m is ω-verdict equivalent to n, both monitors must disagree on finitely many extensions of $as's_b$. This however can be done only if m_a and n'_a also disagree on finitely many extensions of $s's_b$. This is because we have seen that under σ', only the variable x can contribute to the rejections sets of the monitors and it does so by being mapped to $s_b.no$. However, as s_b is not a prefix of $as's_b$ we know that also none of its extensions are prefixes of $as's_b$. Therefore the rejection of s_b does not cause the rejection of any of the prefixes and extensions of $as's_b$. This implies that the infinite trace s is only rejected by $\sigma'(m_a)$ but not $\sigma'(n'_a)$, which implies that the monitors m_a and n'_a still disagree on infinitely many extensions of s_0 under the new substitution σ' which is a contradiction.

It is now easy to see that for each $a \in A$ and for each substitution σ we have that $L_r(a.\sigma(m_a)) \cdot \text{ACT}^\omega = L_r(a.\sigma(n'_a)) \cdot \text{ACT}^\omega$ which implies $L_r(\sigma(m_a)) \cdot \text{ACT}^\omega = L_r(\sigma(n'_a)) \cdot \text{ACT}^\omega$. It remains to see that $L_a(\sigma(m_a)) \cdot \text{ACT}^\omega = L_a(\sigma(n'_a)) \cdot \text{ACT}^\omega$.

To this end, assume, towards a contradiction, that there exist a substitution σ and an infinite trace s such that $s \in L_a(\sigma(m_a)) \cdot \text{ACT}^\omega$ but $s \notin L_a(\sigma(n'_a)) \cdot \text{ACT}^\omega$. Following the argument for the rejection sets, we can infer that there is a finite trace s_0 accepted by $\sigma(m_a)$ but not by $\sigma(n'_a)$. Again by using the proof of Theorem 4.5 , we can transform σ into a σ' that causes a disagreement over the rejection of a trace s'_0 for $\sigma(m_a)$ and $\sigma(n'_a)$ i.e. $s'_0 \in L_r(\sigma'(m_a))$ but $s'_0 \notin L_r(\sigma'(n'_a))$. This, in turn, means we can apply the same reasoning as before for the rejection of a trace to reach a contradiction, namely that m and n are not ω-verdict equivalent.

We can therefore conclude that $\sigma(m_a) \simeq_\omega \sigma(n'_a)$ under any substitution σ and therefore we can apply our induction hypothesis to obtain $\mathcal{E}'_{\omega,f} \vdash m_a = n'_a$. Using the congruence rules, we have $\mathcal{E}'_{\omega,f} \vdash m = n$, and we are done. □

Table 4.4 summarizes the equational axiom systems we have obtained.

4.5 A Non-Finite-Axiomatizability Result

Observe that the family of axioms $\mathcal{O} = \{O2_{s,k} \mid s \in \mathit{Act}^*, k \geq 0\}$, which is included in $\mathcal{E}'_{v,f}$, is infinite. Thus it is natural to wonder whether verdict equivalence has a finite equational axiomatization over Mon_F. In the remainder of this section, we will provide a negative answer to that question by showing that no finite subset of $\mathcal{E}'_{v,f}$ is enough to prove all the equations in $\mathcal{O}$.

Intuitively, the proof of the above claim proceeds as follows. Let $\mathcal{E}$ be an arbitrary finite subset of $\mathcal{E}'_{v,f}$. First of all, we isolate a property of equations that is satisfied by all the equations that are provable from $\mathcal{E}$. We then show that there are equations in the family $\mathcal{O}$ that do not have the given property. This means that those equations are not provable from $\mathcal{E}$ and, therefore, that $\mathcal{E}$ cannot be complete for verdict equivalence.

An arbitrary finite axiom set vs. a finite subset of $\mathcal{E}'_{v,f}$ In Section 4.4.2, in Theorem 4.5, we proved that $\mathcal{E}'_{v,f}$ is complete for open terms over a finite action set modulo verdict equivalence. Therefore, without loss of generality, we can assume that any basis would be, in fact, a subset of the equations in $\mathcal{E}'_{v,f}$. To see this, consider any sound equation that could be involved in an arbitrary axiom set. Since $\mathcal{E}'_{v,f}$ is complete this equation is derivable from it. In addition, since every proof is finite, there is a finite number of axioms of $\mathcal{E}'_{v,f}$ involved in this proof. Therefore, any finite family of equations is derivable from a finite subset of the equations in $\mathcal{E}'_{v,f}$. This means that if another finite family of equations was complete, there would also be a finite subset of equations from $\mathcal{E}'_{v,f}$ which would also be complete. From now on, when considering a finite equational basis we will always mean a subset of the equations in $\mathcal{E}'_{v,f}$. This reasoning is referred to as a compactness argument.

We remind our readers that we assume that all axiom systems that we consider are closed under symmetry. This preserves finiteness and allows us to simplify our arguments, since the symmetry rule does not need to be used in equational proofs.

Definition 4.10 (Notation). *For a finite, non empty set of equations $\mathcal{E}$ we denote as* depth $(()\,\mathcal{E})$ *the quantity:*

$$max\{\text{depth}\,(m) \mid m = n \in \mathcal{E}\}.$$

(A1) $x + y = y + x$	$(\mathbf{E_a})$ $a.\mathit{end} = \mathit{end}$ $(a \in \textsc{Act})$
(A2) $x + (y + z) = (x + y) + z$	$(\mathbf{Y_a})$ $\mathit{yes} = \mathit{yes} + a.\mathit{yes}$ $(a \in \textsc{Act})$
(A3) $x + x = x$	$(\mathbf{N_a})$ $\mathit{no} = \mathit{no} + a.\mathit{no}$ $(a \in \textsc{Act})$
(A4) $x + \mathit{end} = x$	$(\mathbf{D_a})$ $a.(x + y) = a.x + a.y$ $(a \in \textsc{Act})$

The axioms of $\mathcal{E}_v$, which are ground complete for $\simeq$ (Theorem 4.2).

$$(\mathbf{Y}_\omega)\ \mathit{yes} = \sum_{a \in \textsc{Act}} a.\mathit{yes} \qquad\qquad (\mathbf{N}_\omega)\ \mathit{no} = \sum_{a \in \textsc{Act}} a.\mathit{no}$$

The axiom system $\mathcal{E}_\omega = \mathcal{E}_v \cup \{Y_\omega, N_\omega\}$ is ground complete for $\simeq_\omega$ when Act is finite (Theorem 4.3).

$$(\mathbf{O1})\ \mathit{yes} + \mathit{no} = \mathit{yes} + \mathit{no} + x$$

The axiom system $\mathcal{E}'_v = \mathcal{E}_v \cup \{O1\}$ is complete for $\simeq$ when Act is infinite (Theorem 4.4).

$$\mathcal{O} = \{O2_{s,k} \mid s \in \textsc{Act}^*, k \geq 0\} \text{ where}$$

$$(\mathbf{O2_{s,k}})\ x + s.x + \overline{s}^{(k)}(\mathit{yes} + \mathit{no}) = x + \overline{s}^{(k)}(\mathit{yes} + \mathit{no})$$

The axiom system $\mathcal{E}'_{v,f} = \mathcal{E}'_v \cup \mathcal{O}$ is complete for $\simeq$ when Act is finite and $|\textsc{Act}| \geq 2$ (Theorem 4.5).

$$(\mathbf{V_1})\ a.x + x = x$$

The axiom system $\mathcal{E}'_{v,1} = \mathcal{E}'_v \cup \{V_1\}$ is complete for $\simeq$ when $|\textsc{Act}| = 1$ (Theorem 4.6).

$$(\mathbf{V1}_\omega)\ x = a.x$$

The axiom system $\mathcal{E}'_{\omega,1} = \{A1, \ldots, A4, V1_\omega, O1\}$ is complete for $\simeq_\omega$ when $|\textsc{Act}| = 1$ (Theorem 4.7).

The axiom system $\mathcal{E}'_{\omega,f} = \mathcal{E}_\omega \cup \mathcal{E}'_{v,f}$ is complete for $\simeq_\omega$ when Act is finite and $|\textsc{Act}| \geq 2$ (Theorem 4.8).

Table 4.4: Our axiom systems.

The depth of an axiom system turns out to be a very important aspect of it when proving open equations. We refer the reader to all the axioms we have defined so far (Figure 4.4) and particularly to the family $\mathcal{O}$. Take an instance of the family of equations $\mathcal{O}$, namely

$$x + a^k.x + \overline{a^k}^3(yes + no) \simeq x + \overline{a^k}^3(yes + no) \ ,$$

for some k. What we will focus on for equations like this one is the fact that every trace starting with s^k followed by any trace of length larger than $3k+1$ (which is the depth of this equation), is both accepted and rejected by both sides of the equation for any closed substitution. This fact is exactly the intuition behind the property that we will use. We now proceed to formulate this property formally:

Lemma 4.19. *Let $\mathcal{E}$ be a finite subset of $\mathcal{E}'_{v,f}$ and let $m = n$ be an equation in $\mathcal{E}$. Assume that for some string s:*

- *$m \xrightarrow{s} m' + x$, for some monitor m' and variable x and*
- *$n \overset{s}{\not\rightarrow} n' + x$ for any n'.*

Then, for every trace of the form $s.s'$ where $|s'| \geq \mathrm{depth}(\mathcal{E})$, we have that $ss' \in L_a(\sigma(m))$ and $ss' \in L_r(\sigma(m))$ for every substitution σ.

Proof. It suffices to examine each member of $\mathcal{E}'_{v,f}$ separately.

- Each axiom in $\mathcal{E}_v$ does not have any one-sided occurrence of a variable as the ones stated and therefore the lemma holds vacuously.
- For the axiom $O1$ we have that both sides accept and reject all traces for each σ and therefore the claim follows trivially.
- We are left to discuss the family of equations $\mathcal{O}$. Let us select an arbitrary member of this family, i.e. for some $s_0 \in Act^*$ and some $k \geq 0$, the equation

  $$x + s_0.x + \overline{s_0}^{(k)}(yes + no) = x + \overline{s_0}^{(k)}(yes + no) \ .$$

  We see that the depth of x is 1, the depth of $s_0.x$ is $|s_0| + 1$ and the depth of the term $\overline{s_0}^k(yes + no)$ is $(k+1)|s_0| + 1$ (which follows by the definition of the term $\overline{s}^k(m)$). We can also see that the term $\overline{s_0}^k(yes + no)$ accepts and rejects all traces of the form $s_0 s'$, where the length of s' is strictly bigger than $(k-1)|s_0|$, which is enough for the statement to hold. □

Now that we have defined the property we were looking for over a finite subset $\mathcal{E}$ of $\mathcal{E}'_{v,f}$, we proceed to show that the property itself is preserved by equational proofs from $\mathcal{E}$.

Theorem 4.9. *Let $\mathcal{E}$ be a finite subset of $\mathcal{E}'_{v,f}$ and let $m = n$ be an equation such that $\mathcal{E} \vdash m = n$. Assume that:*

- *$m \xrightarrow{s} m' + x$ for some string s, monitor m' and variable x and*

- *$n \overset{s}{\not\rightarrow} n' + x$ for any n'.*

Then, for every trace of the form $s.s'$ where $|s'| \geq \mathrm{depth}\,(\mathcal{E})$, we have that $ss' \in L_a(\sigma(m))$ and $ss' \in L_r(\sigma(m))$ for every substitution σ.

Proof. We will use induction over the length of the proof that results in an arbitrary equation $m = n$. Our base case is a proof of length one, where the the only equations we can prove are the axioms themselves and therefore the property holds by Lemma 4.19.

Assume now we have shown that all proofs of length up to ℓ preserve the property. We will show that proofs of length up to $\ell + 1$ do so as well. The final step of a proof can be performed by applying:

- The congruence rule for $+$,
- The congruence rule for action prefixing $a.\_$,
- A variable substitution (for an open substitution σ), or
- Transitivity.

Note here that, as we mentioned earlier, the axiom system $\mathcal{E}'_{v,f}$ is closed with respect to symmetry and therefore there is no need to use the symmetry rule in proofs. We proceed by considering each of the above-mentioned proof steps.

- The congruence rule for $+$ must be applied as so: Assume two equations $m_1 = n_1$ and $m_2 = n_2$, two already proven equations for which the statement of the theorem holds (inductive hypothesis). By applying the congruence rule for $+$ we have proven the equation $m = m_1 + m_2 = n_1 + n_2 = n$. Assume that $m \xrightarrow{s} m' + x$ for some string s, monitor m' and variable x and $n \overset{s}{\not\rightarrow} n' + x$ for any n'. By the operational semantics of Mon_F we have that either $m_1 \xrightarrow{s} m' + x$ or $m_2 \xrightarrow{s} m' + x$. Without loss of generality assume $m_1 \xrightarrow{s} m' + x$. Moreover we have that $n_1 \overset{s}{\not\rightarrow} n'_1 + x$ for any n'_1 since $n \overset{s}{\not\rightarrow} n' + x$ for any n'. By inductive hypothesis then for every trace of the form $s.s'$ where $|s'| \geq \mathrm{depth}\,(\mathcal{E})$ we have that $s.s' \in L_a(\sigma(m_1))$ and $s.s' \in L_r(\sigma(m_1))$ for every substitution σ. This in turn implies that $s.s' \in L_a(\sigma(m))$ and $s.s' \in L_r(\sigma(m))$ for every substitution σ and we are done.

- We now consider the case of applying the congruence rule for action prefixing. Assume a proven equation $m_0 = n_0$ on which we apply the axiom prefixing congruence rule for an action $a \in Act$, that is, $m = a.m_0 = a.no = n$. Assume now that $m \xrightarrow{s} m_s + x$ for some string s, monitor m_s and variable x and $n \overset{s}{\not\rightarrow} n_s + x$ for any n_s. Since $m = a.m_0$, it follows that $s = as_0$ and $m_0 \xrightarrow{s_0} m'_0 + x$ for some m'_0 and $n_0 \overset{s_0}{\not\rightarrow} n'_0 + x$ for any n'_0. Therefore by inductive hypothesis we have that all traces of the form $s_0.s'$ where $|s'| \geq \mathrm{depth}\,(\mathcal{E})$ are accepted and rejected by m_0 under any substitution. Consequently all traces of the form $as_0.s' = ss'$ are both accepted and rejected by m under any substitution and we are done.

- Consider now variable substitution. Note that we will consider open substitutions, in order to capture the more general case. The case of closed substitutions is of course trivial as after one of them is applied there are no variable occurrences left in any equation and therefore the result holds vacuously. We have now that $\mathcal{E} \vdash m' = n'$ for some open monitors m' and n' and that we apply the open substitution σ_0 in order to prove the open equation $m = \sigma_0(m') = \sigma_0(n') = n$. Assume now that $\sigma_0(m) \xrightarrow{s} m_s + x$ for some string s, monitor m_s and variable x and $\sigma_0(n) \not\xrightarrow{s} n_s + x$ for any n_s. We can easily see that every such one-sided occurrence of a variable in the new equation must have resulted from a one-sided variable occurrence in $m' = n'$. This is because if there were no one-sided variable occurrences in the old equation, then under no substitution could one have introduced a variable in only one side without also introducing it on the other side. This means that there exists some variable y (which could be the same as x) such that $m' \xrightarrow{s_0} m'_{s_0} + y$ for some string s_0 where s_0 a prefix of s, monitor m'_{s_0} and variable y and $n' \not\xrightarrow{s_0} n'_{s_0} + y$ for any n'_{s_0}. The reason why s_0 must be a prefix of s is that an open substitution can only expand the traces that lead to a variable occurrence in the original term. By applying our inductive hypothesis on $m' = n'$, we have that both m' and n' must accept and reject all traces of the form $s_0.s'$ where $|s'| \geq \text{depth}(\mathcal{E})$ under any substitution σ. This, in turn, implies that $\sigma_0(m') = m$ accepts and rejects traces of the form ss' under any closed substitution σ. In fact $\sigma(\sigma_0(m')) = \sigma_0(\sigma_0(m'))$ which means that m and n reject the traces of the form $s_0.s$ as well. Since s_0 is a prefix of s we have that for every extension of s of length at least depth$(\mathcal{E})$ there exists an extension of s_0 of length at least depth$(\mathcal{E})$ that is a prefix of it. Since all traces $s_0.s'$ of this length are both accepted and rejected under any substitution, the same applies for the traces $s.s'$ and we are done.

- The case of transitivity is also straightforward though the following inductive argument. We start by $\mathcal{E} \vdash m = m'$ and $\mathcal{E} \vdash m' = n$ and we apply the transitivity rule to prove $m = n$. Assume that $m \xrightarrow{s} m_s + x$ for some trace s, variable x and monitor m_s, while $n \not\xrightarrow{s} n_s + x$ for any n_s. We have that either: $m' \xrightarrow{s} m'_s + x$ for some m'_s or $m' \not\xrightarrow{s} m'_s + x$. In the first case we have that the equation $m' = n$ which has already been proven by $\mathcal{E}$ satisfies the premises of the theorem and therefore by induction hypothesis all traces of the form $s.s'$ where $|s'| \geq \text{depth}(\mathcal{E})$ are both accepted and rejected by both m' and n. Since $n \simeq m$ by the soundness of $\mathcal{E}'_{v,f}$ and thus $\mathcal{E}$, we have that m also accept and rejects all of these traces and we are done. In the second case and via a similar argument we have the same result.

This concludes the case analysis for our inductive proof and we are done. □

As we can see, if we start from any finite subset $\mathcal{E}$ of $\mathcal{E}'_{v,f}$, we are bound to only prove equations that have the property in the statement of Theorem 4.9. We now argue that for each $\mathcal{E}$ there will always exist sound equations in $\mathcal{E}'_{v,f}$ that do not satisfy the above property and therefore the axiom set $\mathcal{E}$ is not enough to prove them.

Lemma 4.20. *Let $\mathcal{E}$ be a finite subset of $\mathcal{E}'_{v,f}$. There exists a sound equation $m = n$ in $\mathcal{O}$ such that $m \xrightarrow{s} m' + x$ for some string s, monitor m' and variable x and $n \overset{s}{\not\rightarrow} n' + x$ for any n' and there is at least one trace of the form $s.s'$ where $|s'| \geq \text{depth}\,(\mathcal{E})$ and $s.s' \notin L_a(\sigma(m))$ and $s.s' \notin L_a(\sigma(m))$ for the one substitution $\sigma_{end} = end$, for every x.*

Proof. It suffices to give an example from the members of the family $\mathcal{O}$. Namely we consider the equation:

$$x + a^n.x + \overline{(a^n)}^3(yes + no) = x + \overline{(a^n)}^3(yes + no)\ ,$$

where $n > \text{depth}\,(\mathcal{E})$.

We can clearly see that first of all the occurrence of x after the trace a^n is one-sided in the left hand side of the equation. However there is a substitution (namely $\sigma(x) = end$) under which the trace a^{2n+1} is neither accepted nor rejected by the two monitors even though the length of $a^{(n+1)}$ is strictly larger than depth $(\mathcal{E})$. □

Theorem 4.10. *There is no finite complete set of axioms for verdict equivalence over Mon_F over a finite, non-unary set of actions.*

Proof. Let $\mathcal{E}$ be a finite subset $\mathcal{E}'_{v,f}$. Then, by the above lemma, $\mathcal{E}$ cannot prove the sound equation

$$x + a^n.x + \overline{(a^n)}^3(yes + no) = x + \overline{(a^n)}^3(yes + no)\ ,$$

for $n > \text{depth}\,(\mathcal{E})$ and we are done. □

4.6 Conclusions

In this chapter, we have studied the equational theory of recursion-free, regular monitors from [14, 15, 104] modulo two natural notions of monitor equivalence, namely verdict and ω-verdict equivalence. We have provided complete axiomatizations for those equivalences over closed and open terms. The axiomatizations over closed terms are finite when so is the set of actions monitors can process. On the other hand, even when the set of actions is finite, whether those equivalences have finite bases over open terms depends on the cardinality of the action set. For instance, we have shown that verdict equivalence has no finite equational axiomatization when the set of actions contains at least two actions.

Since verdict and ω-verdict equivalence are trace-based behavioral equivalences, our axiomatizations, which are summarized in Table 4.4, share a number of equations with those for trace and completed trace equivalence over BCCSP [197] and for equality of regular expressions [78, 140, 182]. However, the presence of the *yes*, *no* and *end* verdicts yields a number of novelties and technical complications, which are most evident in the axiomatization results over open terms and in the negative result we present in Section 4.5. By way of example, we remark here that, as mentioned in [71], trace and completed trace equivalence are finitely based over BCCSP when the set of actions is finite, unlike the notions we study in this chapter over monitors. Moreover, unlike the one given in this chapter, proofs of non-finite-axiomatizability results for regular expressions rely on families of equations that exploit the interplay between Kleene star and concatenation, such as

$$a^* = (a^n)^*(1 + a + \cdots + a^{n-1}) \quad (n > 0).$$

See, for instance, [3, 78, 177].

The results presented in this chapter deal with a minimal language for monitors that is mainly of theoretical interest and set the stage for further research. An interesting and natural avenue for future work is to study the complexity of the equational theory of verdict and ω-verdict equivalence. Moreover, one could investigate axiomatizations of those behavioral equivalences over extensions of recursion-free monitors with the parallel operators considered in [14] and/or with recursion [104]. As shown in [14](Proposition 3.8), every ‘reactive parallel monitor’ is verdict equivalent to a regular one. This opens the tantalizing possibility that verdict equivalence affords an elegant equational axiomatization over such monitors. However, the proof of Proposition 3.8 in [14] relies on a non-trivial automata-theoretic construction, which would have to be simulated equationally to transform ‘reactive parallel monitors’ into regular ones. We leave this interesting problem for further study.

Chapter 5

The Axiomatizability of Open CCS Terms Modulo Rooted Weak Bisimilarity

In the previous chapters we have considered the τ action as an observable move by a process. We now switch from strong to weak semantics: in this new setting, a τ-move corresponds to a *silent* (or *hidden*, *invisible*) step in the behavior of a process. In detail, we are interested in studying an equational characterization of the parallel composition operator modulo *rooted weak bisimilarity.* That is because weak bisimilarity in its original version is not a congruence, and thus our equational theory cannot be applied to it. We begin with a few important definitions.

5.1 Background

We remind the reader the definition, and associated notation of weak bisimilarity in Chapter 2. Similarly to those definitions, here $\textsc{Act}$ stands for the set of actions including τ and it is ranged over by α.

Definition 5.1 (Rooted weak bisimilarity). *Let* $(\mathbf{P}, \textsc{Act}, \rightarrow)$ *be a LTS.* Weak bisimilarity*, denoted by* $\sim_{\texttt{WB}}$*, is the largest binary symmetric relation over* $\mathbf{P}$ *such that whenever* $p \sim_{\texttt{WB}} q$ *and* $p \xrightarrow{\alpha} p'$*, then either*

- $\alpha = \tau$ *and* $p' \overset{\tau}{\Rightarrow} q$*, or*
- *there is a processes* q' *such that* $q \overset{\alpha}{\Rightarrow} q'$ *and* $p' \mathcal{R} q'$*.*

Then, rooted weak bisimilarity*, denoted by* $\sim_{\texttt{RWB}}$*, is the binary symmetric relation over* $\mathbf{P}$ *such that whenever* $p \sim_{\texttt{RWB}} q$ *and* $p \xrightarrow{\alpha} p'$*, then there is a process* q' *such that* $q \overset{\alpha}{\Rightarrow} q'$ *and* $p' \sim_{\texttt{WB}} q'$*.*

It is well known that rooted weak bisimilarity is an equivalence relation, and that the root condition is necessary to guarantee the compositionality with respect to the nondeterministic choice operator (as well as the left merge), see, e.g., [56, 200], and thus that $\sim_{\texttt{RWB}}$ is a congruence over CCS.

Moller proved that the use of auxiliary operators is not only sufficient to obtain a finite equational charasterization of $\|$, but it is *necessary* indeed.

Theorem 5.1 (Moller [157, 158, 160]). *Bisimilarity has no finite, complete axiomatization over* CCS.

To prove this result, in [160], Moller considered the following family of equations $\{M_n\}_{n\geq 1}$:

$$(x+y) \parallel \sum_{i=1}^{n} z_i + \sum_{i=1}^{n} \Big(x \parallel z_i + y \parallel z_i\Big) \approx$$

$$x \parallel \sum_{i=1}^{n} z_i + y \parallel \sum_{i=1}^{n} z_i + \sum_{i=1}^{n} \Big((x+y) \parallel z_i\Big) \qquad (M_n)$$

and he argued that all these equations should be sound modulo any behavioral congruence that is *reasonable*, including bisimilarity. Roughly speaking, for each $n \geq 1$, the terms in the two sides of M_n can match exactly their single step behavior and, at the same time, the equation does not introduce any causal dependency between the behavior of the single components of each term. Moller then considered a particular family of instantiations $\{I_n\}_{n\geq 1}$ of $\{M_n\}_{n\geq 1}$, consisting only of closed terms:

$$(a+aa) \parallel \sum_{i=1}^{n} a^i + \sum_{i=1}^{n} \Big(a \parallel a^i + aa \parallel a^i\Big) \approx$$

$$a \parallel \sum_{i=1}^{n} a^i + aa \parallel \sum_{i=1}^{n} a^i + \sum_{i=1}^{n} \Big((a+aa) \parallel a^i\Big) \qquad (I_n)$$

and he argued that no finite set of equations, that are sound modulo bisimilarity, can derive I_n for each $n \geq 1$.

To this end, he applied the following proof strategy, which has been later referred to as the *proof-theoretic* approach to negative results. Whenever an equation $t \approx u$ is provable from an axiom system E, then there is a *proof* of it, i.e., a sequence of equations $t_i \approx u_i$, for $i = 1, \ldots, n$, such that $t = t_n$, $u = u_n$, and each equation $t_i \approx u_i$ is in turn derivable from $\mathsf{E} \cup \{t_j \approx u_j \mid j < i\}$. The aim in the proof-theoretic approach is to show that no such sequence exists, so that the considered equation cannot be derived from E. To this end, we need to identify a specific property of terms, say P_n for $n \geq 0$, that, when n is *large enough*, is preserved by provability from finite, sound axiom systems. Roughly, this means that if:

- E is a finite set of axioms that are sound modulo $\sim$,
- the equation $p \approx q$ is provable from E, and
- $n > \text{size}(t)$ for any term t in the equations in E,

then either both p and q satisfy P_n, or none of them does. Then, we exhibit an infinite family of sound equations in which P_n is not preserved, namely it is satisfied only by one side of each equation.

Using this method, Moller proved that whenever n is larger than the size of any term occurring in the equations in a finite, sound, axiom system E, then the instance I_n cannot be derived from E. (In this case, the property P_n was to have a summand bisimilar to $(a + aa) \parallel \sum_{i=1}^{n} a^i$.)

5.2 The negative Result in the Weak Setting

In this part we present our original contribution, namely a negative answer to the following problem:

> *Can we obtain a finite axiomatization of parallel composition* modulo rooted weak bisimilarity *over* CCS? (Q3)

Our aim is to prove the following theorem:

Theorem 5.2. *Rooted weak bisimilarity has no finite, complete axiomatization over CCS.*

To this end, we exploit the family of equations $\{M_n\}_{n\geq 1}$, from [160], introduced in Section 5.1. First of all, notice that the equations M_n are all sound modulo rooted weak bisimilarity, as bisimilarity is included in rooted weak bisimilarity, i.e. $t \sim_{\mathtt{B}} u$ implies $t \sim_{\mathtt{RWB}} u$ for all CCS terms t, u. Then, we remark that in [160] Moller obtained his result over a fragment of the language CCS that have considered in the paper. In particular, he considered the *purely interleaving parallel composition* operator, i.e., parallel composition without communication. Notice that if we restrict the set of actions one with no actions and co-actions then there is no difference between full parallel composition and interleaving, since the lack of actions co-names prevents any form of synchronization between CCS terms. The CCS terms considered by Moller were indeed built over the set of actions without co-actions, and so we will from now on use $_{\mathrm{ACT}}$ to denote the fragment of CCS considered by Moller in [160].

Informally, the core of our proof consists in showing that any equation over CCS that is sound modulo $\sim_{\mathtt{RWB}}$ and that does not contain any occurrence of the prefixing operator, is *also* sound modulo $\sim_{\mathtt{B}}$ over $\mathrm{CCS}_{\mathrm{ACT}}$. Then we show that, since all the terms occurring in the family $\{M_n\}_{n\geq 1}$ do not contain any occurrence of prefixing, any proof of an equation M_n from an axiom system sound modulo $\sim_{\mathtt{RWB}}$, uses *only* equations over terms that do not contain any occurrence of prefixing. Consequently, any finite axiom system that is sound modulo $\sim_{\mathtt{RWB}}$ over CCS, and can prove all the equations in the family $\{M_n\}_{n\geq 1}$, would also be sound modulo $\sim_{\mathtt{B}}$ over $\mathrm{CCS}_{\mathrm{ACT}}$. As this contradicts the negative result obtained by Moller in [160], we can conclude that rooted weak bisimilarity has no finite, complete axiomatization over CCS.

We devote the remainder of this section to a formalization of the intuitions given above. We remark that, although we formally discuss only the case of $\sim_{\mathtt{RWB}}$, our negative result can be extended to any weak congruence $\sim$ such that: M_n is sound modulo $\sim$ for all $n \geq 1$, $\sim$ coincides with $\sim_{\mathtt{B}}$ over $\mathrm{CCS}_{\mathrm{ACT}}$, and whenever $p \sim q$ then any initial τ-step by p is matched by q and vice versa. In particular, our result holds for the *rooted* versions of *branching bisimilarity*, *delay bisimilarity* and *η-bisimilarity.*

Firstly, we introduce the notion of *action-free* terms, i.e., CCS terms that do not contain any occurrence of prefixing.

Definition 5.2 (Action-free term)**.** *Let t be a* CCS *term. We say that t is* action-free *if $t \overset{\alpha}{\nrightarrow}$ for all $\alpha \in$* ACT.

An equation $t \approx u$ is action-free *if t and u are action-free.*

A fundamental property of action-free equations is that their soundness modulo rooted weak bisimilarity over CCS implies soundness modulo bisimilarity over $\mathrm{CCS}_{\mathrm{ACT}}$.

Proposition 5.1. *Let t, u be action-free* CCS *terms. If $t \approx u$ is sound modulo $\sim_{\mathtt{RWB}}$ over* CCS*, then it is also sound modulo $\sim_{\mathtt{B}}$ over* CCS $_{\mathrm{ACT}}$.

Proof. Assume that $t \approx u$ is action-free and sound modulo $\sim_{\mathtt{RWB}}$. Let σ be any closed substitution mapping variables to a $\mathrm{CCS}_{\mathrm{ACT}}$ processes. We remark that processes in $\mathrm{CCS}_{\mathrm{ACT}}$ do not contain any occurrence of action τ. By the soundness of $t \approx u$, we have that $\sigma(t) \sim_{\mathtt{RWB}} \sigma(u)$. Since $\sim_{\mathtt{B}}$ coincides with $\sim_{\mathtt{RWB}}$ over τ-free processes, we obtain that $\sigma(t) \sim_{\mathtt{B}} \sigma(u)$. Hence, by the arbitrariness of σ, we can conclude that $\sigma_{\mathrm{ACT}}(t) \sim_{\mathtt{B}} \sigma_{\mathrm{ACT}}(u)$ for all closed substitutions σ_{ACT} over $\mathrm{CCS}_{\mathrm{ACT}}$, thus giving that $t \sim_{\mathtt{B}} u$ over $\mathrm{CCS}_{\mathrm{ACT}}$. □

We identify a particular substitution, denoted by $\sigma_{\mathtt{nil}}$, that maps each variable to the null process. Formally, the substitution $\sigma_{\mathtt{nil}}$ is defined as $\sigma_{\mathtt{nil}}(x) = \mathtt{nil}$, for all $x \in \mathcal{V}$.

Lemma 5.1. *Let t be a* CCS *term.*

1. *If t is action-free, then $\sigma_{\mathtt{nil}}(t) \sim_{\mathtt{B}} \mathtt{nil}$.*
2. *If t is not action-free, then there exists an action $\alpha \in$* ACT *such that $\sigma(t) \xrightarrow{\alpha}$, for any substitution σ.*

Proof. In both cases, the proof follows by induction over the structure of the term t. □

We now proceed to show that proofs of action-free equations, from an axiom system that is sound modulo rooted weak bisimilarity, use only action-free equations.

Proposition 5.2. *Let* E *be an axiom system sound modulo $\sim_{\mathtt{RWB}}$.*

1. *If $t \approx u$ is sound modulo $\sim_{\mathtt{RWB}}$ and t is action-free, then also u is action-free.*
2. *If $\mathsf{E} \vdash t \approx u$ and t is action-free, then a proof of $t \approx u$ from* E *uses only action-free equations.*

Proof. We start from the first item. As $t \approx u$ is sound modulo $\sim_{\mathtt{RWB}}$, we get that $\sigma_{\mathtt{nil}}(t) \sim_{\mathtt{RWB}} \sigma_{\mathtt{nil}}(u)$. Moreover, as t is action-free, by Lemma 5.1.1 we have that $\sigma_{\mathtt{nil}}(t)$ cannot perform any action. Hence, $\sigma_{\mathtt{nil}}(u)$ cannot perform any action either since, by the root condition, any possible initial τ-transition from $\sigma_{\mathtt{nil}}(u)$ would have to be matched by a τ-transition from $\sigma_{\mathtt{nil}}(t)$. By Lemma 5.1.2, we can then conclude that u is action free.

Let us now deal with the second item. First of all, we notice that since $t \approx u$ is provable from E, then it is sound modulo $\sim_{\mathtt{RWB}}$. Hence, as t is action-free, we can apply Proposition 5.2.1 and obtain that u is action-free as well. The proof then proceeds by induction on the length of the proof of $t \approx u$ from E, where the inductive step is carried out by a case analysis on the last rule of equational logic that is used in the proof. We expand only the case in which the last rule applied is an instance of the substitution rule. The other cases are standard.

Assume that $t = \sigma(t')$ and $u = \sigma(u')$ for some substitution σ and CCS terms t', u' such that $t' \approx u' \in \mathsf{E}$. Since t and u are both action-free, from Lemma 5.1.2 we can infer that t' and u' are action-free as well. In fact, if t' was not action-free, we could directly infer that $\sigma(t') \xrightarrow{\alpha}$ for some $\alpha \in \textsc{Act}$, thus giving a contradiction with $t = \sigma(t')$ being action-free. Similarly for u'. Hence $t' \approx u' \in \mathsf{E}$ is action-free. Notice now that since t and u are both action-free, we can infer that the substitution rule used in the last step of the proof of $t \approx u$ is action-free. Therefore, we can conclude that the proof of $t \approx u$ from E uses only action-free equations. □

Theorem 5.2 can then be obtained as a direct consequence of the following result:

Theorem 5.3. *Assume that E is a finite axiom system over* CCS *that is sound modulo $\sim_{\mathtt{RWB}}$. Then there exists some $n \geq 1$ such that $\mathsf{E} \not\vdash M_n$, where M_n is the n-th member of the family $\{M_n\}_{n \geq 1}$ introduced in Section 5.1.*

Proof. Assume, towards a contradiction, that $\mathsf{E} \vdash M_n$ for all $n \geq 1$. Since M_n is an action-free equation for each $n \geq 1$, by Proposition 5.2.2 we have that all the equations that are used in the proof from E of M_n are action-free as well. Moreover, for each $n \geq 1$, M_n can be proved by using finitely many action-free equations, as E is finite by the proviso of the theorem. By Proposition 5.1 we have that all these equations are also sound modulo $\sim_{\mathtt{B}}$ over $\mathrm{CCS}_{\textsc{Act}}$. Therefore, we can conclude that the finite axiom set E allows us to prove M_n, for all $n \geq 1$, over $\mathrm{CCS}_{\textsc{Act}}$. This contradicts the negative result obtained by Moller in [157], and we can therefore conclude that there is at least one $n \geq 1$ such that M_n is not provable from E. □

5.3 Conclusion

The original contribution presented here, paves the way to the following research question:

Are there general techniques for lifting negative results from strong to weak congruences? (FW4)

An answer to this question will allow us to solve many problems that have been already solved for strong semantics, but that are still open for weak semantics.

Overall, the question answered is indeed a step towards understanding better the nature of weak semantics and axiomatizations. However, since this result concerns open equations, it could be the case that closed CCS terms modulo rooted weak bisimilarity afford a finite equational axiomatization. One can easily verity that the equations necessary for this hardness result would collapse into a closed one that can be proven via the expansion laws (which are still infinitely many, but at the time of studying this setting we had no guarantee yet that in the weak setting the expansion laws cannot be replaced by finitely many other equations). It turns out that another weak congruence, namely *rooted branching bisimilarity* affords no finite, ground-complete axiomatization over CCS (see [30]). This result is produced by extending Moller's technique to account for τ steps in the proofs. Moreover the infinite family of equations used is provable via the expansion laws, and thus the negative result is produced without gaining any insight on the kind of (potentially infinitely many) unknown equations that could be valid in the weak setting.

It is our hope that discovering such new infinite families of equations would eventually lead to possessing infinite complete equational bases over open terms. Such achievement would then enable us to use compactness to extend the non finite axiomatizability results in the following way: Assume an (infinite) complete set of axioms, called $\mathcal{E}$, over some weak equivalence, we could infer that if a finite one $\mathcal{E}'$, for either open or closed terms existed then it should be provable by a finite subset of $\mathcal{E}$, as it is complete. Thus, we would not need to prove that *arbitrary* finite sets of equations cannot axiomatize our equivalence at hand, but instead, proving it for subsets of $\mathcal{E}$ would suffice. Later on, in our future work, we will present one of our initial attempts in this direction.

Chapter 6

Non-Finite Axiomatizability Results via Reductions

6.1 Introduction

Some of Frits Vaandrager's early seminal contributions were firmly rooted in the theory and applications of process algebras and their semantics. Having been brought up in the tradition of Bergstra and Klop's Algebra of Communicating Processes (ACP) [36, 55, 56], Frits Vaandrager studied semantic models of algebraic process description languages [198, 113], equational axiomatizations of process equivalences [38] and their application in verification (see, for instance, [199, 192]). Moreover, together with Aceto and Bloom, in [1] he initiated the study of methods for generating finite, ground-complete, equational axiomatizations of bisimilarity [152, 166] from operational specifications given in the GSOS format [62]. The techniques proposed in [1] can be used to synthesize auxiliary operators, such as Bergstra and Klop's left- and communication-merge operators, that make finite axiomatizations possible and paved the way to several further studies in the literature—see, for instance, the developments presented in [11, 100, 111, 151].

The use of auxiliary operators to obtain finite, equational, ground-complete axiomatizations of bisimilarity, even for very inexpressive process algebras, was justified by Moller in [157, 160, 158], where he showed that bisimilarity has no finite axiomatization over minimal fragments of Milner's Calculus of Communicating Systems (CCS) [152] and Bergstra and Klop's ACP. (Henceforth, we will consider the recursion, relabeling and restriction free fragment of CCS, which, for simplicity, we still denote as CCS.) Moller's above-mentioned, path-breaking, negative results have been followed by a wealth of research on non-finitely-based fragments of process algebras—see, for instance, [2, 5, 6, 9, 19, 20, 22, 25, 72].

Our Contribution

In this work, we celebrated Frits Vaandrager's early contributions to the study of algebraic process description languages by answering the two questions that Rob van Glabbeek[1] asked Luca Aceto after his invited talk at LICS 2021[2]:

> Would Moller's non-finite axiomatizability result for CCS remain true if we replaced CCS parallel composition with the parallel operators from Hoare's Communicating Sequential Processes (CSP) [129]? And what if we added the restriction operator or the relabeling operator to CCS instead?

Our first contributions concern the existence of finite, ground-complete axiomatizations of bisimilarity over process algebras that extend the language BCCSP [126, 193, 152] with parallel operators from CSP. (BCCSP is a common fragment of Milner's CCS and Hoare's CSP suitable for describing finite process behavior.)

For each set of actions A, the CSP parallel operator $|_A$ behaves like interleaving parallel composition for all actions that are not contained in A, but requires transitions of its operands that are labeled with some action $a \in A$ to synchronise. The result of such a synchronization is an a-labeled transition of the composite parallel process, which can itself synchronize further with a-labeled steps from its environment. Therefore, unlike CCS parallel composition that is based on hand-shaking communication, the parallel operators from CSP support multi-way synchronization and span the whole spectrum from pure interleaving parallel composition (the operator $|_\emptyset$) to synchronous composition (the operator $|_{\mathrm{Act}}$, where Act is the whole collection of actions that processes may perform).

We start our investigations by considering the languages $\mathrm{BCCSP}_A^{\mathrm{p}}$, which extend BCCSP with the parallel operator $|_A$ for some subset A of the whole set of actions $\textsc{Act}$, and $\mathrm{BCCSP}_\tau^{\mathrm{p}}$, which contains the parallel operator $|_A$ for each $A \subseteq \textsc{Act}$, and the τ-prefixing operator for a distinguished action $\tau \not\in \textsc{Act}$. We show that Moller's non-finite axiomatizability result for bisimilarity still holds over $\mathrm{BCCSP}_A^{\mathrm{p}}$, when A is a strict subset of $\textsc{Act}$, and $\mathrm{BCCSP}_\tau^{\mathrm{p}}$. On the other hand, bisimilarity affords a finite, ground-complete axiomatization over $\mathrm{BCCSP}_{\textsc{Act}}^{\mathrm{p}}$.

The proofs of the above-mentioned negative results for $\mathrm{BCCSP}_A^{\mathrm{p}}$, when A is a strict subset of $\textsc{Act}$, and $\mathrm{BCCSP}_\tau^{\mathrm{p}}$ employ a reduction-based technique proposed in [12] for showing new, non-finite axiomatizability results over process algebras from already-established ones. In our setting, such reductions are translations from terms in the languages $\mathrm{BCCSP}_A^{\mathrm{p}}$ $(A \subset Act)$ and $\mathrm{BCCSP}_\tau^{\mathrm{p}}$ to those in the fragment of CCS studied by Moller that

- preserve sound equations and equational provability over the source language, and
- reflect an infinite family of equations responsible for the non-finite axiomatizability of the target language.

No reduction from $\mathrm{BCCSP}_{\textsc{Act}}^{\mathrm{p}}$ to CCS satisfying the former property modulo bisimilarity reflects Moller's family of equations witnessing his negative result over CCS. Therefore, the reduction technique cannot be applied to $\mathrm{BCCSP}_{\textsc{Act}}^{\mathrm{p}}$. Indeed, we present a finite, ground-complete axiomatization of bisimilarity over $\mathrm{BCCSP}_{\textsc{Act}}^{\mathrm{p}}$.

[1] Rob van Glabbeek was one of Frits Vaandrager's early collaborators and fellow doctoral student at CWI.

[2] See `https://www.youtube.com/watch?v=2PxM3f0QWDM` for a recording of that talk.

We also show that, if we consider the language BCCSP$^{\mathrm{p}}$, namely BCCSP with a parallel operator $|_A$ for each $A \subseteq \textsc{Act}$, then no reduction that is structural, i.e. that does not introduce new variables and it is defined compositionally over terms, can reflect Moller's family of equations. However, we conjecture that bisimilarity does not admit a finite, ground-complete axiomatization over BCCSP$^{\mathrm{p}}$.

For our final contribution, we consider the languages CCS$^{\mathrm{r}}$ and CCS$^{\ell}$, namely CCS enriched with *restriction operators* of the form $\cdot\backslash R$, and CCS enriched with *relabeling operators*, of the form $\cdot[f]$. Informally, $R \subseteq \textsc{Act}$ is a set of actions that are restricted, meaning that the execution of a-labeled transitions (and of their "complementary actions") is prevented in $t\backslash R$ for all $a \in R$, while $f : \textsc{Act} \to \textsc{Act}$, is a mapping of actions to actions, meaning that whenever a is is an action occurring in t, the action $f(a)$ will be executed in $t[f]$. By exploiting the reduction technique described above, we show that Moller's negative result can be lifted to CCS$^{\mathrm{r}}$, giving thus that bisimilarity admits no finite, ground-complete axiomatization over CCS with restriction.

Our contributions can then be summarized as follows:

1. We consider BCCSP$^{\mathrm{p}}_A$, i.e., BCCSP enriched with *one* CSP-style parallel composition operator $|_A$, with $A \subset \textsc{Act}$, and we show that, over that language, bisimilarity admits no finite, ground-complete axiomatization (Theorem 6.4).

2. We consider BCCSP$^{\mathrm{p}}$, i.e., BCCSP enriched with *all* CSP-style parallel composition operators $|_A$, and we show that there is no structural reduction from BCCSP$^{\mathrm{p}}$ to CCS that can reflect the family of equations used by Moller to prove the negative result for bisimilarity over CCS (Theorem 6.5).

3. We consider BCCSP$^{\mathrm{p}}_\tau$, i.e, BCCSP$^{\mathrm{p}}$ enriched with the τ-prefixing, and we show that this algebra admits no finite, ground-complete axiomatization modulo bisimilarity (Theorem 6.6).

4. We consider BCCSP$^{\mathrm{p}}_{\textsc{Act}}$, i.e., BCCSP enriched with the CSP-style parallel composition operator $|_{\textsc{Act}}$, and we present a finite, ground-complete axiomatization for it, modulo bisimilarity (Theorem 6.7).

5. We consider CCS$^{\mathrm{r}}$, i.e., CCS with the restriction operator $\backslash$, and we show that bisimilarity has no finite, ground-complete axiomatization over it (Theorem 6.8).

6. We consider CCS$^{\ell}$, i.e., CCS with the relabeling operator $[f]$, and we show that bisimilarity has no finite, ground-complete axiomatization over it (Theorem 6.9).

Organization of Contents

In Section 6.2 we review basic notions on process semantics, behavioral equivalences, and equational logic. We also briefly recap Moller's negative result for bisimilarity over CCS. In Section 6.3 we give a bird's-eye view of the reduction technique from [12]. In Section 6.4, we present the lifting of Moller's negative result to BCCSP$^{\mathrm{p}}_A$ (for $A \subset \textsc{Act}$) and BCCSP$^{\mathrm{p}}$, and then we discuss the collapse of that result in the case of $\mathrm{BCCSP}^{\mathrm{p}}_{\mathrm{ACT}}$. In Section 6.5.1, we use the reduction technique to prove the non-finite axiomatizability result for $\mathrm{CCS}^{\mathrm{r}}$. We conclude by discussing some directions for future work in Section 6.6.

6.2 Preliminaries

In this section we present some background notions on process algebras and equational logic. To make our contribution self-contained, we also briefly recap Moller's work on the nonexistence of finite axiomatizations modulo bisimilarity over the recursion, relabeling, and restriction free fragment of CCS (henceforth simply referred to as CCS).

In what follows, we assume that the set of actions Act is finite and non-empty. We let $p, q, \ldots$ range over $\mathcal{P}$, and $a, b, \ldots$ over Act. Moreover, as usual, we use $p \xrightarrow{a} p'$ in lieu of $(p, a, p') \in \rightarrow$. For each $p \in \mathcal{P}$ and $a \in \text{Act}$, we write $p \xrightarrow{a}$ if $p \xrightarrow{a} p'$ holds for some p', and $p \overset{a}{\not\rightarrow}$ otherwise.

The Language BCCSP

In this part we will consider several algebraic process description languages, each characterized by the presence of a particular operator, or sets of operators. As all those languages are extensions of BCCSP [126], consisting of the basic operators from CCS [152] and CSP [129], in this section we use that language to introduce some general notions and notations on term algebras that will be useful throughout the remainder of the chapter.

BCCSP terms are defined by the following grammar:

$$t ::= \quad \mathtt{nil} \quad | \quad x \quad | \quad a.t \quad | \quad t + t \ , \qquad \text{(BCCSP)}$$

where x is drawn from a countable set of variables $\mathcal{V}$, a is an action from Act, $a.(\cdot)$ is the prefix operator, defined for each $a \in \text{Act}$, and $\cdot + \cdot$ is the nondeterministic choice operator. We shall use the meta-variables $t, u, \ldots$ to range over process terms. The *size* of a term t, denoted by $\text{size}(t)$, is the number of operator symbols in t. A term is *closed* if it does not contain any variables. Closed terms, or *processes*, will be denoted by $p, q, \ldots$. In particular, we denote the set of all BCCSP terms by $\mathbb{T}(\text{BCCSP})$, and the set of closed BCCSP terms (or BCCSP processes) by $\mathcal{P}(\text{BCCSP})$. This notation can be directly extended to all the languages that we will consider. Moreover, we omit trailing nil's from terms and we use a *summation* $\sum_{i=1}^{k} t_i$ to denote the term $t = t_1 + \cdots + t_k$, where the empty sum represents nil. Henceforth, for each action $a \in \text{Act}$ and natural number $m \geq 0$, we let a^0 denote nil and a^{m+1} denote $a.(a^m)$.

We use the *Structural Operational Semantics* (SOS) framework [168] to equip processes with an operational semantics. The SOS rules (also called inference rules, or deduction rules) for the BCCSP operators given above are reported in Table 6.1. A (*closed*) *substitution* σ is a mapping from process variables to (closed) terms. Substitutions are extended from variables to terms, transitions, and rules in the usual way. Note that $\sigma(t)$ is closed, if so is σ. The inference rules in Table 6.1

$$\text{(act)}\ \frac{}{a.t \xrightarrow{a} t} \qquad \text{(lSum)}\ \frac{t \xrightarrow{a} t'}{t + u \xrightarrow{a} t'} \qquad \text{(rSum)}\ \frac{u \xrightarrow{a} u'}{t + u \xrightarrow{a} u'}$$

Table 6.1: The SOS rules for BCCSP operators ($a \in \textsc{Act}$).

allow us to derive valid transitions between closed BCCSP terms. The operational semantics for BCCSP is then modeled by the LTS whose processes are the closed terms in $\mathcal{P}$(BCCSP), and whose labeled transitions are those that are provable from the SOS rules. The same approach will be applied to all the extensions of BCCSP that we will consider. The SOS rules of each language will be presented in the respective sections.

We call an equivalence relation a *congruence* over a language if it is compositional with respect to the operators of the language, i.e., the replacement of a component with an equivalent one does not affect the overall behavior. Formally, the congruence property for bisimilarity over BCCSP, and its extensions, consists in verifying whether, given any n-ary operator f,

$$f(p_1, \dots, p_n) \sim f(q_1, \dots, q_n) \text{ whenever } p_i \sim q_i \text{ for all } i = 1, \dots, n.$$

Since all the operators considered in this chapter are defined by inference rules in the de Simone format [86], by [195, Theorem 4] we have that bisimilarity is a *congruence* over BCCSP and over all the languages that we will study, similarly to bisimulation over CCS.

Equational Logic in the Context of BCCSP and Extensions

Here, as we study many languages that extend BCCSP, we remind the reader how one can define the equational laws of equation over languages of a signature $\mathcal{F}$, as discussed in Chapter 2. Then, a classic question is whether an algebra modulo the chosen notion of behavioral congruence (in this work, bisimilarity, as defined in Chapter 2, Definition 2.2, and over several flavors) affords a finite equational axiomatization. For example, as shown by Hennessy and Milner in [126], the equations in Table 6.2 are a finite axiomatization of bisimilarity over BCCSP. We denote by E_0 the axiom system consisting of the equations in Table 6.2. Later on, we will extend this set of axioms to present our positive result for $\text{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$.

Moller's Result Over CCS

In his thesis [157], Moller gave a celebrated non-finite axiomatizability result in the field of process algebra, namely:

Theorem 6.1. *Bisimilarity admits no finite, ground-complete axiomatization over CCS.*

$$
\begin{array}{ll}
\text{(A1)} & x \approx x + x \\
\text{(A2)} & x + y \approx y + x \\
\text{(A3)} & (x + y) + z \approx x + (y + z) \\
\text{(A4)} & x + \mathtt{nil} \approx x
\end{array}
$$

Table 6.2: Finite equational basis for BCCSP modulo bisimilarity.

$$
\text{(lPar)}\ \frac{t \xrightarrow{a} t'}{t \parallel u \xrightarrow{a} t' \parallel u} \qquad \text{(rPar)}\ \frac{u \xrightarrow{a} u'}{t \parallel u \xrightarrow{a} t \parallel u'}
$$

Table 6.3: The SOS rules for CCS_a interleaving parallel composition.

Specifically, Moller considered the language CCS_a with interleaving parallel composition, defined over $\textsc{Act} = \{a\}$ by the following syntax:

$$
t ::= \quad \mathtt{nil} \quad | \quad x \quad | \quad a.t \quad | \quad t + t \quad | \quad t \parallel t \qquad\qquad (\mathrm{CCS}_a)
$$

where $x \in \mathcal{V}$ and $\parallel$ denotes the interleaving parallel composition operator.

The SOS rules for CCS_a operators are given by the rules in Table 6.1, plus the rules for the interleaving parallel operator presented in Table 6.3.

In detail, for his result, Moller applied the following proof strategy, later referred to as the proof-theoretic approach to negative results [7]. He considered the infinite family of equations Φ with

$$
\begin{aligned}
&\Phi = \{\varphi_n \mid n \geq 0\} \\
&\varphi_n \ : \ a \parallel (\sum_{i=1}^{n} a^i) \approx a.(\sum_{i=1}^{n} a^i) + (\sum_{i=2}^{n+1} a^i) \qquad\qquad (n \geq 0)
\end{aligned}
$$

and he proved that whenever n is larger than the size of any term occurring in the equations in a finite, sound axiom system E, then equation φ_n cannot be derived from E.

Hence, Theorem 6.1 specialized to the following result, which will play a fundamental role in the technical development of our contributions:

Theorem 6.2 (Moller's negative result [157, Theorem 5.2.12]). *No finite axiom system that is sound modulo bisimilarity over* CCS_a *can prove the whole family of equations* Φ*. Thus no finite, ground-complete axiom system can exist for* CCS_a *modulo bisimilarity.*

6.3 The Proof Strategy: Reduction Mappings

The non-finite axiomatizability results that we will present in this chapter are all obtained by means of a proof technique, proposed in [12], that allows for transferring this kind of negative results across process languages. Even though we only apply that technique out-of-the-box towards establishing new results, we decided to give, in this section, an overview of the terminology and results presented in [12], to improve the readability of our contributions. As our studies are focused on the axiomatizability of bisimilarity, we consider only this behavioral congruence in the presentation below.

We consider two processes description languages defined over the same set of variables: L_{neg} and L_{new}. L_{neg} is known to be non-finitely axiomatisable modulo bisimilarity, whereas L_{new} is the language for which we want to prove this negative result. The aim of the proof technique proposed in [12] is to establish whether it is possible to *lift* the known result for L_{neg} to L_{new}. This approach is based on a variation of the classic idea of *reduction mappings* that, in this setting, are *translations* from $\mathbb{T}(L_{\text{new}})$ to $\mathbb{T}(L_{\text{neg}})$ that preserve soundness and provability.

Given a translation mapping $\widehat{\cdot} : \mathbb{T}(L_{\text{new}}) \to \mathbb{T}(L_{\text{neg}})$ and a collection E of equations over L_{new} terms, we let $\widehat{E} = \{\widehat{t} \approx \widehat{u} \mid t \approx u \in E\}$. The notion of *reduction* is then formalized as follows:

Definition 6.1 (Reduction). *A mapping* $\widehat{\cdot} \colon \mathbb{T}(L_{\text{new}}) \to \mathbb{T}(L_{\text{neg}})$ *is a reduction from* $\mathbb{T}(L_{\text{new}})$ *to* $\mathbb{T}(L_{\text{neg}})$*, when for all* $t, u \in \mathbb{T}(L_{\text{new}})$*:*

1. $t \sim u \implies \widehat{t} \sim \widehat{u}$*, i.e.,* $\widehat{\cdot}$ *preserves sound equations, and*
2. $E \vdash t \approx u \implies \widehat{E} \vdash \widehat{t} \approx \widehat{u}$*, for each axiom system* E *over* L_{new}*, i.e,* $\widehat{\cdot}$ *preserves provability.*

Interestingly, in [12, Theorem 2] it is proved that if a mapping is *structural*, then it automatically satisfies Definition6.1.2. Hence, the notion of structural mapping will be crucial in the development of our results, as it allows for a significant simplification of the technical proofs.

Definition 6.2 (Structural mapping). *A mapping* $\widehat{\cdot} \colon \mathbb{T}(L_{\text{new}}) \to \mathbb{T}(L_{\text{neg}})$ *is structural if:*

- *It is the identity function over variables, i.e.,* $\widehat{x} = x$ *for each variable* x*.*
- *It does not introduce new variables, i.e., the set of variables occurring in the term* $\widehat{f(x_1, \ldots, x_n)}$ *is included in* $\{x_1, \ldots, x_n\}$*, for each operator* f *in* L_{new} *and sequence of distinct variables* $x_1, \ldots, x_n$*.*
- *It is defined compositionally for each operator* f *in* L_{new}*, i.e.,* $\widehat{f(t_1, \ldots, t_n)} = \widehat{f(x_1, \ldots, x_n)}[\widehat{t_1}/x_1, \ldots, \widehat{t_n}/x_n]$*, for a sequence of distinct variables* $x_1, \ldots, x_n$ *and a sequence of terms* $t_1, \ldots, t_n$*. (Here* $[\widehat{t_1}/x_1, \ldots, \widehat{t_n}/x_n]$ *stands for the substitution mapping each variable* x_i *to* $\widehat{t_i}$ $(1 \leq i \leq n)$*, and acting like the identity function on all the other variables.)*

Given a substitution $\sigma\colon \mathcal{V} \to \mathbb{T}(L_{\mathrm{new}})$, we let $\widehat{\sigma}\colon \mathcal{V} \to \mathbb{T}(L_{\mathrm{neg}})$ denote the substitution that maps each variable x to $\widehat{\sigma(x)}$.

Proposition 6.1. *Assume that* $\widehat{\cdot}\colon \mathbb{T}(L_{\mathrm{new}}) \to \mathbb{T}(L_{\mathrm{neg}})$ *is a structural mapping. Then*

- $\widehat{\sigma(t)} = \widehat{\sigma}(\widehat{t})$, *for each term* $t \in \mathbb{T}(L_{\mathrm{new}})$, *and for each substitution* $\sigma\colon \mathcal{V} \to \mathbb{T}(L_{\mathrm{new}})$.
- *A structural mapping satisfies Definition6.1.2.*

Assume now that we have an infinite collection E of equations that are sound modulo bisimilarity, but that are not derivable from any finite, sound axiom system over L_{neg}. The idea in [12] is then that if a structural mapping $\widehat{\cdot}$ is a reduction from $\mathbb{T}(L_{\mathrm{new}})$ to $\mathbb{T}(L_{\mathrm{neg}})$ that contains all the equations in E in its range, then the "malicious" collection of equations that map to those in E cannot be derivable from any finite, sound axiom system over L_{new}. In fact, if those derivations were possible, then the equational properties of $\widehat{\cdot}$ would allow us to write derivations (obtained via the translations of the equational proofs) of the equations in E from a finite, sound axiom system over L_{neg}. As this contradicts the established negative result over L_{neg}, the non-finite axiomatizability result over L_{new} follows.

The intuitions above are formalized in the following definition and theorem.

Definition 6.3 (E-reflection)**.** *Let E be an axiom system over L_{neg}. A reduction* $\widehat{\cdot}$ *is E-*reflecting, *when for each* $t \approx u \in E$, *there are terms* $t', u' \in \mathbb{T}(L_{\mathrm{new}})$ *such that the equation* $t' \approx u'$ *is sound modulo* $\sim$, $\widehat{t'} = t$ *and* $\widehat{u'} = u$. *A reduction is* ground *E-reflecting, if the conditions above are satisfied over closed equations.*

Theorem 6.3 (The lifting theorem)**.** *Assume that there is a set of (closed) equations E over L_{neg} that is sound modulo $\sim$ and that is not derivable from any finite sound axiom system over L_{neg}. If there exists a (ground) E-reflecting reduction from L_{new} to L_{neg}, then there exists no sound and (ground-)complete finite axiom system for $\sim$ over L_{new}.*

We remark that the notion of (ground) E-reflecting reduction requires that only the equations in E are reflected. This means that to establish the negative result over L_{neg} it is enough to identify a particular family of equations that is reflected, disregarding the effects of the reduction on other sound equations. For our purposes, it will be enough to consider the family of equations Φ used by Moller to prove Theorem 6.2. Hence, our target language will always be CCS_a, and we will use the lifting technique presented in this section to prove negative results for the languages $\mathrm{BCCSP}^{\mathtt{p}}_A$ (Section 6.4.2), $\mathrm{BCCSP}^{\mathtt{p}}$ (Section 6.4.3), and $\mathrm{CCS}^{\mathtt{r}}$ (Section 6.5.1).

6.4 Results for CSP Parallel Composition

In this section we investigate the existence of finite, ground-complete axiomatizations of bisimilarity over the process description languages $\mathrm{BCCSP}^{\mathtt{p}}_A$ (for all

$A \subset \textsc{Act}$), $\mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}$, $\mathrm{BCCSP}^{\mathrm{p}}$ and $\mathrm{BCCSP}^{\mathrm{p}}_{\tau}$. In detail, we apply the reduction technique presented in Section 6.3 to lift Moller's negative result to $\mathrm{BCCSP}^{\mathrm{p}}_{A}$, for each $A \subset \textsc{Act}$, and to $\mathrm{BCCSP}^{\mathrm{p}}_{\tau}$ (Theorem 6.4 and Theorem 6.6, respectively). In between, we show that the reduction technique cannot be applied to $\mathrm{BCCSP}^{\mathrm{p}}$ (Theorem 6.5). Conversely, we establish a positive result for $\mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}$, providing a finite, ground-complete axiomatization for bisimilarity over this language (Theorem 6.7).

6.4.1 The Languages $\mathrm{BCCSP}^{\mathrm{p}}_{A}$, $\mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}$, $\mathrm{BCCSP}^{\mathrm{p}}$ and $\mathrm{BCCSP}^{\mathrm{p}}_{\tau}$

The languages that we consider in this section are obtained by extending BCCSP with instances of the CSP-like parallel composition operator $|_A$, where $A \subseteq \textsc{Act}$ is the set of actions that must be performed synchronously by the parallel components. For this reason, we shall henceforth refer to A in $|_A$ as to the *synchronization* set. The operator then behaves like interleaving parallel composition on the complement of A.

In detail, the languages are defined by the following grammar

$$t ::= \quad \mathtt{nil} \quad | \quad x \quad | \quad a.t \quad | \quad t + t \quad | \quad t \,|_A\, t \ ,$$

with $x \in \mathcal{V}$ and $a \in \textsc{Act}$, and they differ in the choice of the synchronization set(s) $A \subseteq \textsc{Act}$ as follows:

$\mathrm{BCCSP}^{\mathrm{p}}_{A}$ The parallel operator $|_A$ is defined only over the fixed set $A \subset \textsc{Act}$ (notice that the inclusion is strict).

$\mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}$ The only synchronization set is the entire set of actions $\textsc{Act}$.

$\mathrm{BCCSP}^{\mathrm{p}}$ There are no restrictions on the choice of synchronization sets, i.e. the signature of the language contains the operator $|_A$ for all $A \subseteq \textsc{Act}$.

$\mathrm{BCCSP}^{\mathrm{p}}_{\tau}$ This is like $\mathrm{BCCSP}^{\mathrm{p}}$ with the additional property that the prefixing operator is of the form $\alpha.t$, with α potentially being a silent action τ for a special action label $\tau \notin \textsc{Act}$, similarly to the definition CCS 2.3 (see Section 6.4.3 for further details).

The SOS rules for the CSP-like parallel composition operator $|_A$ are given in Table 6.4. The operational semantics of each of the above-mentioned languages is then given by the rules in Table 6.1 and those in Table 6.4, in which A is instantiated according to the considered language.

Let $L \in \{\mathrm{BCCSP}^{\mathrm{p}}_{A}, \mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}, \mathrm{BCCSP}^{\mathrm{p}}, \mathrm{BCCSP}^{\mathrm{p}}_{\tau}\}$. Since in the technical results to follow we will need to distinguish between transitions over L processes and transitions over CCS_a processes, to avoid possible confusion we will denote the transition relation over $\mathcal{P}(L)$ induced by the rules in Tables 6.1 and 6.4 by $\rightarrow_{\mathrm{p}}$. Similarly, we can properly instantiate the definition of bisimilarity over L processes:

Definition 6.4 (Bisimilarity over $\mathrm{BCCSP}^{\mathrm{p}}_{A}$, $\mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}$, $\mathrm{BCCSP}^{\mathrm{p}}$ and $\mathrm{BCCSP}^{\mathrm{p}}_{\tau}$)**.** *Let L be any of* $\mathrm{BCCSP}^{\mathrm{p}}_{A}, \mathrm{BCCSP}^{\mathrm{p}}_{\textsc{Act}}, \mathrm{BCCSP}^{\mathrm{p}}, \mathrm{BCCSP}^{\mathrm{p}}_{\tau}$*. Bisimulation relations over L processes are defined by applying Definition 2.2 to the LTS* $(\mathcal{P}(L), \textsc{Act}, \rightarrow_{\mathrm{p}})$ *induced by the SOS rules in Tables 6.1 and 6.4. We use the symbol* $\sim_{\mathrm{p}}$ *to denote bisimilarity over L processes.*

$$\text{(lParA)}\ \frac{t \xrightarrow{a} t'}{t \mid_A u \xrightarrow{a} t' \mid_A u}\ a \notin A \qquad \text{(rParA)}\ \frac{u \xrightarrow{a} u'}{t \mid_A u \xrightarrow{a} t \mid_A u'}\ a \notin A$$

$$\text{(syncA)}\ \frac{t \xrightarrow{a} t', \quad u \xrightarrow{a} u'}{t \mid_A u \xrightarrow{a} t' \mid_A u'}\ a \in A$$

Table 6.4: SOS rules for the parallel operator $|_A$, $A \subseteq \text{Act}$.

It is worth noticing that, as briefly outlined above, when the parallel components t, u in $t \mid_A u$ contain only actions that are not in A, then the semantics of $|_A$ coincides with the semantics of CCS interleaving parallel composition. On the other hand, when t and u contain only actions in A, then $|_A$ behaves like "synchronous" parallel composition. The following example highlights these observations.

***Example* 12.** *Let $A \subseteq \text{Act}$ and $b \in A$. It is not difficult to see that*

$$b \mid_A \sum_{i=1}^{n} b^i \sim_{\mathrm{p}} b \qquad\qquad (n \geq 1)$$

and therefore

$$b \mid_A \sum_{i=1}^{n} b^i \sim_{\mathrm{p}} b \mid_A \sum_{j=1}^{m} b^j \qquad\qquad (n, m \geq 1).$$

In particular, we have that the axiom

$$b.x \mid_A (b.y + z) \approx (b.x \mid_A b.y) + (b.x \mid_A z) \qquad \textit{if } b \in A$$

is sound modulo $\sim_{\mathrm{p}}$ over the languages considered in this section.

Conversely, if we pick an action $a \notin A$, then we have

$$a \mid_A \sum_{i=1}^{n} a^i \sim_{\mathrm{p}} a.\sum_{i=1}^{n} a^i + \sum_{j=1}^{n} a^{j+1} \qquad\qquad (n \geq 0)$$

and thus

$$a \mid_A \sum_{i=1}^{n} a^i \not\sim_{\mathrm{p}} a \mid_A \sum_{j=1}^{m} a^j \qquad\qquad (n \neq m).$$

Notice that, for $a \notin A$, if we let

$$\varphi_A^n : a \mid_A \sum_{i=1}^{n} a^i \approx a.\sum_{i=1}^{n} a^i + \sum_{j=1}^{n} a^{j+1} \qquad (n \geq 0), \qquad\qquad (6.1)$$

then the family of equations $\Phi_A = \{\varphi_A^n \mid n \in \mathbb{N}\}$ can be thought of as the counterpart in $\text{BCCSP}_A^{\mathrm{p}}$ of the family Φ used by Moller to prove Theorem 6.2. As we will see, this correspondence will be instrumental in applying the reduction technique to those languages.

6.4.2 The Negative Result for $\mathrm{BCCSP}^{\mathtt{p}}_A$

We start our investigations with $\mathrm{BCCSP}^{\mathtt{p}}_A$, for a given set $A \subset \textsc{Act}$. In particular, by applying the proof methodology discussed in Section 6.3, we prove that:

Theorem 6.4. $\mathrm{BCCSP}^{\mathtt{p}}_A$ *does not have a finite, ground-complete axiomatization modulo bisimilarity.*

Our first step consists in defining a mapping allowing us to rewrite $\mathrm{BCCSP}^{\mathtt{p}}_A$ terms into CCS_a terms. As the target language is built over a specific action, it is natural to have a definition of our mapping that is parametric in that action. Hence, choose an action $a \in \textsc{Act}\backslash A$. Notice that the requirement that the inclusion $A \subset \textsc{Act}$ be strict guarantees that such an action a exists.

Definition 6.5 (The mapping $\mathtt{p}^A_a$)**.** *The mapping* $\mathtt{p}^A_a \colon \mathbb{T}(\mathrm{BCCSP}^{\mathtt{p}}_A) \to \mathbb{T}(\mathrm{CCS}_a)$ *is defined inductively over the structure of terms as follows:*

$$\mathtt{p}^A_a(\mathtt{nil}) = \mathtt{nil} \qquad \mathtt{p}^A_a(x) = x \qquad \mathtt{p}^A_a(t+u) = \mathtt{p}^A_a(t) + \mathtt{p}^A_a(u)$$

$$\mathtt{p}^A_a(b.t) = \begin{cases} a.\mathtt{p}^A_a(t) & \text{if } b = a, \\ \mathtt{nil} & \text{otherwise.} \end{cases} \qquad \mathtt{p}^A_a(t \mid_A u) = \mathtt{p}^A_a(t) \parallel \mathtt{p}^A_a(u).$$

By Definition 6.5, for each $t \in \mathbb{T}(\mathrm{BCCSP}^{\mathtt{p}}_A)$, the only action occurring in $\mathtt{p}^A_a(t)$ is a.

In order to lift the negative result in Theorem 6.2 to $\mathrm{BCCSP}^{\mathtt{p}}_A$, we need to prove that the proposed mapping $\mathtt{p}^A_a$ is a ground Φ-reflecting reduction. Let us first focus on showing that $\mathtt{p}^A_a$ is a reduction, i.e., we need to show that it satisfies the two constraints in Definition 6.1.

Remark 10. *For simplicity, we shall sometimes extend the mapping notation from terms to equations. For instance, if $e : t \approx u$ is an equation over* $\mathrm{BCCSP}^{\mathtt{p}}_A$ *terms, we shall write $\mathtt{p}^A_a(e)$ to denote the equation over* CCS_a *terms $\mathtt{p}^A_a(t) \approx \mathtt{p}^A_a(u)$.*

The following lemma is immediate from Definition 6.5.

Lemma 6.1. *The mapping $\mathtt{p}^A_a$ is structural.*

Hence, in light of Proposition 6.1, the mapping $\mathtt{p}^A_a$ satisfies Definition 6.1.2. Our order of business will now be to show that $\mathtt{p}^A_a$ preserves sound equations.

Lemma 6.2. *For all $p \in \mathcal{P}(\mathrm{BCCSP}^{\mathtt{p}}_A)$ and $q \in \mathcal{P}(\mathrm{CCS}_a)$, if $\mathtt{p}^A_a(p) \xrightarrow{a} q$, then there exists a* $\mathrm{BCCSP}^{\mathtt{p}}_A$ *process p' such that $p \xrightarrow{a}_{\mathtt{p}} p'$ and $\mathtt{p}^A_a(p') = q$.*

Proof. We proceed by structural induction over p.

- Case $p = \mathtt{nil}$. This is vacuous, since $\mathtt{p}^A_a(p)$ has no outgoing transition.
- Case $p = b.p_0$. By Definition 6.5 and the assumption that $\mathtt{p}^A_a(p) \xrightarrow{a} q$, we have that $b = a \notin A$ and $\mathtt{p}^A_a(p_0) = q$. As $p \xrightarrow{a}_{\mathtt{p}} p_0$, the claim follows.

- Case $p = p_1 \,|_A\, p_2$. By Definition 6.5, we have that $\mathtt{p}_a^A(p) = \mathtt{p}_a^A(p_1) \parallel \mathtt{p}_a^A(p_2)$. Moreover, by the proviso of the lemma, $\mathtt{p}_a^A(p_1) \parallel \mathtt{p}_a^A(p_2) \xrightarrow{a} q$, for some CCS$_a$ process q. This follows by an application of either rule (lPar) or rule (rPar) from Table 6.3. We can assume, without loss of generality, that rule (lPar) was applied. (The case of an application of rule (rPar) follows from a similar reasoning.) Hence $\mathtt{p}_a^A(p_1) \xrightarrow{a} q'$ for some CCS$_a$ process q' such that $q' \parallel \mathtt{p}_a^A(p_2) = q$. By the induction hypothesis, we obtain that $p_1 \xrightarrow{a}_{\mathtt{p}} p_1'$ for some $p_1' \in \mathcal{P}(\mathrm{BCCSP}_A^{\mathtt{p}})$ such that $\mathtt{p}_a^A(p_1') = q'$. Hence, as $p_1 \xrightarrow{a}_{\mathtt{p}} p_1'$ and $a \notin A$, we can apply rule (lParA) from Table 6.4 and obtain that $p = p_1 \,|_A\, p_2 \xrightarrow{a}_{\mathtt{p}} p_1' \,|_A\, p_2$. Since $\mathtt{p}_a^A(p_1' \,|_A\, p_2) = \mathtt{p}_a^A(p_1') \parallel \mathtt{p}_a^A(p_2) = q' \parallel \mathtt{p}_a^A(p_2) = q$, the claim follows.

- Case $p = p_1 + p_2$. This case is similar to the case of parallel composition discussed above. The only difference is that rules (lSum) and (rSum) from Table 6.1 are applied in place of rules (lParA) and (rParA), respectively. □

Lemma 6.3. *For all $p, p' \in \mathcal{P}(\mathrm{BCCSP}_A^{\mathtt{p}})$, if $p \xrightarrow{a}_{\mathtt{p}} p'$ then $\mathtt{p}_a^A(p) \xrightarrow{a} \mathtt{p}_a^A(p')$.*

Proof. We proceed by induction on the size of the proof for the transition $p \xrightarrow{a}_{\mathtt{p}} p'$. We distinguish three cases, according to the last inference rule from Tables 6.1 and 6.4 that is applied in the proof. (Notice that the analysis of symmetric rules is omitted.) We remark that since $a \notin A$, rule (syncA) cannot be applied as the last rule in the proof for $p \xrightarrow{a}_{\mathtt{p}} p'$.

- Rule (act). In this case, we have that $p = a.p'$ and $p \xrightarrow{a}_{\mathtt{p}} p'$. By Definition 6.5, we have that $\mathtt{p}_a^A(a.p') = a.\mathtt{p}_a^A(p')$, and, thus, we can apply rule (act) and obtain that $\mathtt{p}_a^A(p) = \mathtt{p}_a^A(a.p') = a.\mathtt{p}_a^A(p') \xrightarrow{a} \mathtt{p}_a^A(p')$. Hence the claim follows in this case.

- Rule (lSum). In this case, we have that $p = p_0 + p_1$, $p_0 \xrightarrow{a}_{\mathtt{p}} p'$, and $\mathtt{p}_a^A(p) = \mathtt{p}_a^A(p_0) + \mathtt{p}_a^A(p_1)$. By the inductive hypothesis we get that $\mathtt{p}_a^A(p_0) \xrightarrow{a} \mathtt{p}_a^A(p')$. By applying now rule (lSum), we conclude that $\mathtt{p}_a^A(p) = \mathtt{p}_a^A(p_0) + \mathtt{p}_a^A(p_1) \xrightarrow{a} \mathtt{p}_a^A(p')$.

- Rule (lParA). In this case, as $a \notin A$, we have that $p = p_0 \,|_A\, p_1$, $p_0 \xrightarrow{a}_{\mathtt{p}} p_0'$ for some $p_0' \in \mathcal{P}(\mathrm{BCCSP}_A^{\mathtt{p}})$, and $p' = p_0' \,|_A\, p_1$. By induction, we obtain that $\mathtt{p}_a^A(p_0) \xrightarrow{a} \mathtt{p}_a^A(p_0')$. Hence, by applying rule (lPar) from Table 6.3 to $\mathtt{p}_a^A(p)$, we get that $\mathtt{p}_a^A(p) = \mathtt{p}_a^A(p_0) \parallel \mathtt{p}_a^A(p_1) \xrightarrow{a} \mathtt{p}_a^A(p_0') \parallel \mathtt{p}_a^A(p_1) = \mathtt{p}_a^A(p_0' \,|_A\, p_1) = \mathtt{p}_a^A(p')$. □

We can now proceed to prove that $\mathtt{p}_a^A$ satisfies Definition 6.1.1 as well. Moreover, we show that it is also ground Φ-reflecting.

Proposition 6.2. *The mapping $\mathtt{p}_a^A$ satisfies the following properties:*

1. *For all $t, u \in \mathbb{T}(\mathrm{BCCSP}_A^{\mathtt{p}})$, if $t \sim_{\mathtt{p}} u$ then $\mathtt{p}_a^A(t) \sim \mathtt{p}_a^A(u)$.*
2. *The mapping $\mathtt{p}_a^A$ is ground Φ-reflecting.*

Proof. We prove the two items separately.

1. First, observe that for every (closed) term t in CCS_a there is a (closed) term $t_a^{\mathtt{p},A}$ in $\mathrm{BCCSP}_A^{\mathtt{p}}$ such that $\mathtt{p}_a^A(t_a^{\mathtt{p},A}) = t$. The term $t_a^{\mathtt{p},A}$ is defined as follows:

   $$\begin{array}{lll} \mathtt{nil}_a^{\mathtt{p},A} = \mathtt{nil} & x_a^{\mathtt{p},A} = x & (a.t)_a^{\mathtt{p},A} = a.t_a^{\mathtt{p},A} \\ (t+u)_a^{\mathtt{p},A} = t_a^{\mathtt{p},A} + u_a^{\mathtt{p},A} & & (t \parallel u)_a^{\mathtt{p},A} = t_a^{\mathtt{p},A} \,|_A\, u_a^{\mathtt{p},A}. \end{array}$$

   Given a CCS_a substitution σ, we define $\sigma_a^{\mathtt{p},A}$ to be the $\mathrm{BCCSP}_A^{\mathtt{p}}$ substitution given by $\sigma_a^{\mathtt{p},A}(x) = (\sigma(x))_a^{\mathtt{p},A}$. By Proposition 6.1 and since the mapping $\mathtt{p}_a^A$ is structural (Lemma 6.1), we have that

   $$\mathtt{p}_a^A(\sigma_a^{\mathtt{p},A}(t)) = \mathtt{p}_a^A(\sigma_a^{\mathtt{p},A})(\mathtt{p}_a^A(t)) = \sigma(\mathtt{p}_a^A(t)),$$

   for all $t \in \mathbb{T}(\mathrm{BCCSP}_A^{\mathtt{p}})$.

   To prove the claim, it is then enough to show that the following relation

   $$\mathcal{R} = \{(\sigma(\mathtt{p}_a^A(t)), \sigma(\mathtt{p}_a^A(u))) \mid t \sim_{\mathtt{p}} u \text{ and } \sigma \colon \mathcal{V} \to \mathcal{P}(\mathrm{CCS}_a)\}$$

   is a bisimulation relation over CCS_a processes.

   Notice, first of all, that since $\sim_{\mathtt{p}}$ is symmetric, then so is $\mathcal{R}$. Assume now that $\sigma(\mathtt{p}_a^A(t)) \,\mathcal{R}\, \sigma(\mathtt{p}_a^A(u))$, where $t, u \in \mathbb{T}(\mathrm{BCCSP}_A^{\mathtt{p}})$ and σ is a closed CCS_a substitution. By the definition of $\mathcal{R}$, we have that $t \sim_{\mathtt{p}} u$. Assume now that $\sigma(\mathtt{p}_a^A(t)) \xrightarrow{a} q$ for some $q \in \mathcal{P}(\mathrm{CCS}_a)$. By the observation above, this means that $\mathtt{p}_a^A(\sigma_a^{\mathtt{p},A}(t)) \xrightarrow{a} q$. By Lemma 6.2, we get that $\sigma_a^{\mathtt{p},A}(t) \xrightarrow{a}_{\mathtt{p}} p'$ for some $p' \in \mathcal{P}(\mathrm{BCCSP}_A^{\mathtt{p}})$ such that $\mathtt{p}_a^A(p') = q$. As $t \sim_{\mathtt{p}} u$ implies that $\sigma_a^{\mathtt{p},A}(t) \sim_{\mathtt{p}} \sigma_a^{\mathtt{p},A}(u)$, we have that $\sigma_a^{\mathtt{p},A}(u) \xrightarrow{a}_{\mathtt{p}} p''$, for some $p'' \in \mathcal{P}(\mathrm{BCCSP}_A^{\mathtt{p}})$ such that $p' \sim_{\mathtt{p}} p''$. Additionally, by Lemma 6.3 we have that $\sigma(\mathtt{p}_a^A(u)) = \mathtt{p}_a^A(\sigma_a^{\mathtt{p},A}(u)) \xrightarrow{a} \mathtt{p}_a^A(p'')$. We can then conclude by noticing that, since $p' \sim_{\mathtt{p}} p''$, by definition of $\mathcal{R}$ it holds that $q = \mathtt{p}_a^A(p') \,\mathcal{R}\, \mathtt{p}_a^A(p'')$, i.e., $\mathcal{R}$ is a bisimulation relation over CCS_a processes.

2. In order to show that $\mathtt{p}_a^A$ is ground Φ-reflecting, it is enough to argue that the family Φ_A consisting of the closed equations φ_A^n defined in Equation 6.1 is mapped exactly onto Φ. Since $a \notin A$ we have that $\mathtt{p}_a^A$ simply replaces all the occurrences of $|_A$ in each equation φ_A^n with $\parallel$. Hence, we have that $\mathtt{p}_a^A(\varphi_A^n) = \varphi_n$, for each $n \geq 0$. □

From Lemma 6.1 and Proposition 6.2, we can infer that $\mathtt{p}_a^A$ is a well-defined reduction as in Definition 6.1, and it is also ground Φ-reflecting. Theorem 6.4 then follows by Theorem 6.2 and Theorem 6.3.

6.4.3 The Case of $\mathrm{BCCSP}^{\mathtt{p}}$ and the Negative Result for $\mathrm{BCCSP}_\tau^{\mathtt{p}}$

Given the negative result over $\mathrm{BCCSP}_A^{\mathtt{p}}$, it is natural to wonder what happens when we extend that language to $\mathrm{BCCSP}^{\mathtt{p}}$, namely BCCSP enriched with an operator $|_A$, for each $A \subseteq \mathrm{Act}$.

One might expect that bisimilarity does not have a finite, ground-complete axiomatization over BCCSP$^{\text{p}}$ and indeed we conjecture that such a results holds. However, the reduction method cannot be applied to prove such a claim.

Specifically, consider the language BCCSP$^{\text{p}}$ over $\text{A}\textsc{ct} = \{a\}$. We can prove the following result:

Theorem 6.5. *There is no structural reduction from* BCCSP$^{\text{p}}$ *to* CCS$_a$ *that is ground* Φ*-reflecting.*

Proof. To simplify notation, let us use $|_a$ in place of $|_{\{a\}}$.

Assume that $\widehat{\cdot}$ is a structural reduction from BCCSP$^{\text{p}}$ to CCS$_a$. Our aim is to prove that $\widehat{\cdot}$ is not ground Φ-reflecting.

To this end, we start by recalling that, since $\widehat{\cdot}$ is structural (Definition 6.2), then:

$$\begin{align}
\widehat{at} &= \widehat{ax}[\widehat{t}/x], \text{ for each } t \in \mathbb{T}(\text{BCCSP}^{\text{p}}) \tag{6.2}\\
\widehat{t_1 \odot t_2} &= \widehat{x_1 \odot x_2}[\widehat{t_1}/x_1, \widehat{t_2}/x_2], \text{ for each } t_1, t_2 \in \mathbb{T}(\text{BCCSP}^{\text{p}}) \tag{6.3}\\
&\qquad\qquad \text{and binary operator } \odot \in \{+, |_\emptyset, |_a\}.
\end{align}$$

Moreover, as $\widehat{\cdot}$ preserves sound equations (Definition 6.1), we have that:

$$\begin{align}
&\widehat{ax \,|_a\, \mathtt{nil}} \sim \widehat{\mathtt{nil}} \sim \widehat{\mathtt{nil} \,|_a\, ax}; \tag{6.4}\\
&\widehat{a^n \,|_a\, a^n} \sim \widehat{a^n}, \text{ for all } n \geq 0; \tag{6.5}\\
&\widehat{\mathtt{nil} + \mathtt{nil}} \sim \widehat{\mathtt{nil}}; \tag{6.6}\\
&\widehat{\mathtt{nil} \,|_\emptyset\, \mathtt{nil}} \sim \widehat{\mathtt{nil}}. \tag{6.7}
\end{align}$$

Assume now that

$$\widehat{x_1 |_a x_2} = t \tag{6.8}$$

for some $t \in \mathbb{T}(\text{CCS}_a)$ with $\text{var}(()t) \subseteq \{x_1, x_2\}$ (as $\widehat{\cdot}$ is structural).

We can distinguish two cases, according to whether t is a closed term or not. In both cases, we shall show that $\widehat{\cdot}$ is not ground Φ-reflecting.

- Case 1: t is a closed CCS$_a$ term. In this case, for each $n \geq 0$, we have that

$$t \sim \widehat{a^n} \sim \mathtt{nil}. \tag{6.9}$$

  Indeed,

$$\begin{align}
\widehat{a^n} &\sim \widehat{a^n \,|_a\, a^n} && \text{(by 6.5)}\\
&\sim t[\widehat{a^n}/x_1, \widehat{a^n}/x_2] && \text{(by 6.3 and 6.8)}\\
&\sim t[\widehat{\mathtt{nil}}/x_1, \widehat{\mathtt{nil}}/x_2] && \text{(since } t \text{ is closed)}\\
&\sim \widehat{\mathtt{nil}} && \text{(by 6.3 and 6.5 with } n = 0).
\end{align}$$

  We now claim that

Claim 1: *For each* $p \in \mathcal{P}(\mathrm{BCCSP}^{\mathrm{p}})$*, it holds that* $\widehat{p} \sim \widehat{\mathtt{nil}}$*.*

Before proving Claim 1 above, we observe that by using it we can immediately show that the mapping $\widehat{\cdot}$ is not ground Φ-reflecting. Indeed, since

$$a \parallel a \not\sim a \parallel (a + a^2),$$

by Claim 1 there cannot be two processes $p, q \in \mathcal{P}(\mathrm{BCCSP}^{\mathrm{p}})$ such that $\widehat{p} = a \parallel a$ and $\widehat{q} = a \parallel (a + a^2)$. Let us now prove Claim 1.

Proof of Claim 1: We proceed by induction on the structure of process p.

- The case $p = \mathtt{nil}$ is trivial.
- Case $p = aq$. We have

$$\begin{aligned}
\widehat{p} &= \widehat{ax}[\widehat{q}/x] && \text{(by 6.2)}\\
&\sim \widehat{ax}[\widehat{\mathtt{nil}}/x] && \text{(by induction and $\sim$ is a congruence)}\\
&\sim \widehat{a\mathtt{nil}} && \text{(by 6.2)}\\
&\sim \mathtt{n\hat{i}l} && \text{(by 6.9).}
\end{aligned}$$

- Case $p = p_1 \odot p_2$ for some binary operator $\odot \in \{+, |_{\emptyset}, |_a\}$. In this case,

$$\begin{aligned}
\widehat{p} &= \widehat{x_1 \odot x_2}[\widehat{p_1}/x_1, \widehat{p_2}/x_2] && \text{(by 6.3)}\\
&\sim \widehat{x_1 \odot x_2}[\widehat{\mathtt{nil}}/x_1, \widehat{\mathtt{nil}}/x_2] && \text{(by induction and $\sim$ is a congruence)}\\
&\sim \widehat{\mathtt{nil} \odot \mathtt{nil}} && \text{(by 6.3)}\\
&\sim \widehat{\mathtt{nil}} && \text{(by 6.5–6.7 according to $\odot$).}
\end{aligned}$$

This concludes the proof of Claim 1.

The proof of Case 1 is now complete.

- Case 2: t is an open CCS$_a$ term. Assume, without loss of generality, that t contains at least an occurrence of x_1. (The cases of $x_2 \in \mathrm{var}(()t)$ and $x_1, x_2 \in \mathrm{var}(()t)$ can be treated in a similar fashion and are therefore omitted.) Firstly, we observe that for each $p \in \mathcal{P}(\mathrm{BCCSP}^{\mathrm{p}})$

$$\begin{aligned}
\widehat{\mathtt{nil}} &\sim \widehat{ap \,|_a\, \mathtt{nil}} && \text{(by 6.4)}\\
&= t[\widehat{ap}/x_1, \widehat{\mathtt{nil}}/x_2] && \text{(by 6.3 and 6.8).}
\end{aligned}$$

Moreover, we recall that for every $u \in \mathrm{CCS}_a$ and $y \in \mathcal{V}$, it holds that whenever $y \in \mathrm{var}(()u)$ then $\mathrm{depth}(\sigma(y)) \leq \mathrm{depth}(\sigma(u))$ for every closed substitution σ. Hence, since $t \in \mathbb{T}(\mathrm{CCS}_a)$ and $x_1 \in \mathrm{var}(()t)$, we have that

$$\mathrm{depth}(\widehat{ap}) \leq \mathrm{depth}(\widehat{ap \,|_a\, \mathtt{nil}}) = \mathrm{depth}(\widehat{\mathtt{nil}}). \tag{6.10}$$

We claim that

Claim 2: *For each $n \geq 0$ and processes $p_1, \ldots, p_n \in \mathcal{P}(\mathrm{BCCSP}^{\mathtt{p}})$, it holds that* $\mathrm{depth}(\widehat{\sum_{i=1}^{n} ap_i}) \leq \mathrm{depth}(\widehat{\mathtt{nil}})$.

Proof of Claim 2: We proceed by induction on $n \geq 0$.

- The case $n = 0$ is trivial.
- For the inductive step, we have that:

$$
\begin{aligned}
& \mathrm{depth}(\widehat{\sum_{i=1}^{n+1} ap_i}) \\
&= \mathrm{depth}(\widehat{\sum_{i=1}^{n} ap_i + ap_{n+1}}) \\
&= \mathrm{depth}(\widehat{x_1 + x_2}[\widehat{\sum_{i=1}^{n} ap_i}/x_1, \widehat{ap_{n+1}}/x_2]) && \text{(by 6.3)} \\
&\leq \mathrm{depth}(\widehat{x_1 + x_2}[\widehat{\mathtt{nil}}/x_1, \widehat{\mathtt{nil}}/x_2]) && \text{(by induction and 6.10)} \\
&= \mathrm{depth}(\widehat{\mathtt{nil} + \mathtt{nil}}) && \text{(by 6.3)} \\
&= \mathrm{depth}(\widehat{\mathtt{nil}}) && \text{(by 6.6 and Remark 1).}
\end{aligned}
$$

This concludes the proof of Claim 2.

Claim 3: *For each $p \in \mathcal{P}(\mathrm{BCCSP}^{\mathtt{p}})$ it holds that* $\mathrm{depth}(\widehat{p}) \leq \widehat{(\mathtt{nil})}$.

Proof of Claim 3: First of all, we notice that each $\mathrm{BCCSP}^{\mathtt{p}}$ process can be rewritten into *head normal form* up to bisimilarity. This means that, given any $p \in \mathcal{P}(\mathrm{BCCSP}^{\mathtt{p}})$, we have that $p \sim \sum_{i=1}^{n} ap_i$ for some $n \geq 0$ and $p_1, \ldots, p_n \in \mathcal{P}(\mathrm{BCCSP}^{\mathtt{p}})$.

Since $\widehat{\cdot}$ preserves sound equations, we have

$$\widehat{p} \sim \widehat{\sum_{i=1}^{n} ap_i}.$$

Hence, by Claim 2 above, it follows that

$$\mathrm{depth}(\widehat{p}) = \mathrm{depth}(\widehat{\sum_{i=1}^{n} ap_i}) \leq \mathrm{depth}(\widehat{\mathtt{nil}}). \tag{6.11}$$

This concludes the proof of Claim 3.

We can now proceed to show that $\widehat{\cdot}$ is not ground Φ-reflecting. Let $k = \mathrm{depth}(\widehat{\mathtt{nil}})$. We have that equation $\varphi_k \in \Phi$ is of the form:

$$a \parallel (\sum_{i=1}^{k} a^i) \approx a.(\sum_{i=1}^{k} a^i) + \sum_{i=2}^{k+1} a^i.$$

In particular, the depth of $a \parallel (\sum_{i=1}^{k} a^i)$ is $k+1$. Therefore, by 6.11, there is no $p \in \mathcal{P}(\text{BCCSP}^{\text{p}})$ such that $\widehat{p} = a \parallel (\sum_{i=1}^{k} a^i)$.

The proof of Case 2 is now concluded.

This completes the proof of the Theorem 6.5. $\square$

Although we proved Theorem 6.5 in the simplified setting of $\text{Act} = \{a\}$, it is not difficult to see that the proof can be extended to the general case $\{a\} \subset \text{Act}$ in a straightforward manner.

Since the reduction method cannot be applied, one might show the non-existence of a finite, ground-complete axiomatization of bisimilarity over BCCSP^{p} by adapting the strategy employed by Moller in his proof of Theorem 6.2. However, since that proof would require several pages of technical results, we leave it as an avenue for future research, and we deal with the presence of all the operators $|_A$ in a simplified setting.

The basic idea behind the reduction defined for $\text{BCCSP}^{\text{p}}_A$ is that we can always identify an action $a \in \text{Act} \setminus A$ such that the parallel operator $|_A$ always allows for interleaving of a-moves of its arguments. Clearly, if we add an operator $|_A$ for each $A \subseteq \text{Act}$ to the language, it is no longer possible to identify such an action. There is, however, a special action that is not used to build syntactically CSP terms, but it is however necessary to express their semantics: the silent action $\tau \notin \text{Act}$. CSP terms are defined over Act, which means that the language does not offer a τ-prefixing operator; however, in order to properly define the operational semantics of the internal choice operator, the set of action labels in the LTS is $\text{Act} \cup \{\tau\}$. In particular, as explained in [67], the operational semantics of the parallel operators always allow for the interleaving of τ-moves of their arguments.

Hence, we now consider $\text{BCCSP}^{\text{p}}_\tau$, i.e., the extension of BCCSP^{p} that includes the τ-prefixing operator, and we prove the following result:

Theorem 6.6. $\text{BCCSP}^{\text{p}}_\tau$ *does not afford a finite, ground-complete axiomatization modulo bisimilarity .*

To this end, we apply the same proof technique that we used in Section 6.4.2 for $\text{BCCSP}^{\text{p}}_A$. The reduction mapping for $\text{BCCSP}^{\text{p}}_\tau$ is almost identical to the mapping $\mathtt{p}^A_a$ defined for $\text{BCCSP}^{\text{p}}_A$, the only difference being that now we consider the language CCS_τ as target language, i.e., CCS_a with $a = \tau$.

Remark 11. *Theorem 6.2 remains true over* CCS_τ*. In fact, as we are considering strong bisimilarity, there is no difference between τ and any other observable action $a \in \text{Act}$. Specifically, if we let Φ_τ be the family of equations in Φ in which each occurrence of a is replaced by τ, then we can repeat Moller's arguments in a step-by-step fashion to obtain that no finite axiom system, that is sound modulo bisimilarity, can prove the whole family of equations Φ_τ.*

Definition 6.6 (The mapping $\mathtt{p}_a$)**.** *The mapping* $\mathtt{p}_a \colon \mathbb{T}(\text{BCCSP}^{\text{p}}_\tau) \to \mathbb{T}(\text{CCS}_\tau)$ *is defined inductively over the structure of* $\text{BCCSP}^{\text{p}}_\tau$ *terms as follows:*

$$\mathtt{p}_a(\mathtt{nil}) = \mathtt{nil} \qquad \mathtt{p}_a(x) = x \qquad \mathtt{p}_a(t+u) = \mathtt{p}_a(t) + \mathtt{p}_a(u)$$

$$\mathtt{p}_a(\alpha.t) = \begin{cases} \tau.\mathtt{p}_a(t) & \textit{if } \alpha = \tau, \\ \mathtt{nil} & \textit{otherwise;} \end{cases} \qquad \mathtt{p}_a(t \mid_A u) = \mathtt{p}_a(t) \parallel \mathtt{p}_a(u).$$

Intuitively, we use the mapping $\mathtt{p}_a$ to eliminate any action $b \neq \tau$ from terms, so that a process $\mathtt{p}_a(p \mid_A q)$ can perform a transition $\mathtt{p}_a(p \mid_A q) \xrightarrow{\tau} p'$, for some CCS$_a$ process p', if and only if $b = \tau$. (Recall that, by construction $\tau \notin A$ for each $A \subseteq$ Act.)

First, we note that this mapping is a structural mapping.

Lemma 6.4. *The mapping* $\mathtt{p}_a$ *is structural.*

We will now state and prove the results over BCCSP$^{\mathtt{p}}_\tau$, that correspond to Lemma 6.2 and Lemma 6.3 over BCCSP$^{\mathtt{p}}_A$.

Lemma 6.5. *For all* $p \in \mathcal{P}(\text{BCCSP}^{\mathtt{p}}_\tau)$ *and* $q \in \mathcal{P}(\text{CCS}_\tau)$*, if* $\mathtt{p}_a(p) \xrightarrow{\tau} q$*, then there exists a* BCCSP$^{\mathtt{p}}_\tau$ *process* p'*, such that* $p \xrightarrow{\tau}_{\mathtt{p}} p'$ *and* $\mathtt{p}_a(p') = q$*.*

Proof. The proof is by structural induction over p. We omit it since it is similar to that of Lemma 6.2. □

Lemma 6.6. *For all* $p, p' \in \mathcal{P}(\text{BCCSP}^{\mathtt{p}}_\tau)$*, if* $p \xrightarrow{\tau}_{\mathtt{p}} p'$ *then* $\mathtt{p}_a(p) \xrightarrow{\tau} \mathtt{p}_a(p')$*.*

Proof. The proof proceeds by induction over the size of the proof for $p \xrightarrow{\tau}_{\mathtt{p}} p'$. It is analogous to the proof of Lemma 6.3, and it is therefore omitted. □

The following result, which extends Proposition 6.2 to BCCSP$^{\mathtt{p}}_\tau$, allows us to prove that $\mathtt{p}_a$ is a well-defined reduction mapping that is also ground Φ_τ-reflecting.

Proposition 6.3. *The following properties hold for the mapping* $\mathtt{p}_a$*:*

1. *For all* $t, u \in \mathbb{T}(\text{BCCSP}^{\mathtt{p}}_\tau)$*, if* $t \sim_{\mathtt{p}} u$*, then* $\mathtt{p}_a(t) \sim \mathtt{p}_a(u)$*.*
2. *The mapping* $\mathtt{p}_a$ *is ground* Φ_τ*-reflecting.*

Proof.

1. We start by observing that for every (closed) term t in CCS$_\tau$ there is a (closed) term $t^{\mathtt{p}}_\tau$ in BCCSP$^{\mathtt{p}}_\tau$ such that $\mathtt{p}_a(t^{\mathtt{p}}_\tau) = t$. The term $t^{\mathtt{p}}_\tau$ is defined as follows:

$$\mathtt{nil}^{\mathtt{p}}_\tau = \mathtt{nil} \qquad x^{\mathtt{p}}_\tau = x \qquad (\tau.t)^{\mathtt{p}}_\tau = \tau.t^{\mathtt{p}}_\tau$$
$$(t + u)^{\mathtt{p}}_\tau = t^{\mathtt{p}}_\tau + u^{\mathtt{p}}_\tau \qquad (t \parallel u)^{\mathtt{p}}_\tau = t^{\mathtt{p}}_\tau \mid_\emptyset u^{\mathtt{p}}_\tau.$$

   Then, given a CCS$_\tau$ substitution σ, we define the BCCSP$^{\mathtt{p}}_\tau$ substitution $\sigma^{\mathtt{p}}_\tau$ by $\sigma^{\mathtt{p}}_\tau(x) = (\sigma(x))^{\mathtt{p}}_\tau$. By Lemma 6.4 and Proposition 6.1, we have that $\mathtt{p}_a(\sigma^{\mathtt{p}}_\tau(t)) = \mathtt{p}_a(\sigma^{\mathtt{p}}_\tau)(\mathtt{p}_a(t)) = \sigma(\mathtt{p}_a(t))$ for all $t \in \mathbb{T}(\text{BCCSP}^{\mathtt{p}}_\tau)$.

   The proof of this statement then proceeds as that of the corresponding statement in Proposition 6.2, and it is therefore omitted.

2. Consider the family of equations $\Phi_{\tau,\emptyset} = \{\varphi^n_{\tau,\emptyset} \mid n \in \mathbb{N}\}$, where the closed equations $\varphi^n_{\tau,\emptyset}$ are defined as in Equation 6.1, using the set $\emptyset$ as synchronization set, and replacing each occurrence of a with τ. It is straightforward to prove that $\mathtt{p}_a(\varphi^n_{\tau,\emptyset}) = \varphi_{\tau,n}$ for each $n \in \mathbb{N}$. Hence, $\mathtt{p}_a$ is ground Φ_τ-reflecting. □

$$
\begin{array}{ll}
\text{(P1)} & x \mid_{\textsc{Act}} y \ \approx \ y \mid_{\textsc{Act}} x \\
\text{(P2)} & (x+y) \mid_{\textsc{Act}} z \ \approx \ (x \mid_{\textsc{Act}} z) + (y \mid_{\textsc{Act}} z) \\
\text{(P3)} & (x \mid_{\textsc{Act}} \mathtt{nil}) \ \approx \ \mathtt{nil} \\
\text{(P4)} & (a.x \mid_{\textsc{Act}} a.y) \ \approx \ a.(x \mid_{\textsc{Act}} y), \text{for each } a \in \textsc{Act} \\
\text{(P5)} & (a.x \mid_{\textsc{Act}} b.y) \ \approx \ \mathtt{nil}, \text{for } b \neq a, \ \text{ and } a, b \in \textsc{Act}.
\end{array}
$$

Table 6.5: Additional axioms for $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$.

Theorem 6.6 is then obtained as a direct consequence of Lemma 6.4, Proposition 6.3, Theorem 6.3, and Theorem 6.2.

6.4.4 The Case of $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$

We now argue that the requirement that the inclusion $A \subset \textsc{Act}$ be strict, used in Section 6.4.2, is indeed necessary for Theorem 6.4 to hold. We also notice that a similar requirement is not explicitly expressed for the validity of Theorem 6.6, proved in Section 6.4.3, because having $|_A$ defined for all $A \subseteq \textsc{Act}$ automatically guaranteed the existence of at least one synchronization set A such that $a \notin A$ for some action $a \in \textsc{Act}$, namely the synchronization set $A = \emptyset$. Moreover, as discussed in Example 12, given a synchronization set A, the requirement $a \notin A$ is crucial to guarantee the soundness modulo bisimilarity of equation φ^n_A, for any $n \in \mathbb{N}$ (see Equation 6.1).

In this section, we handle the border case of the language $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$, which includes only the parallel operator $|_{\textsc{Act}}$, and we show that for this special case a positive result holds: we provide a *finite, ground-complete axiomatization* of bisimilarity over this language. Let us consider the axiom system $\mathsf{E}_{\mathsf{p}} = \mathsf{E}_0 \cup \{\mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$, where E_0 consists of the axioms in Table 6.2, and axioms P1–P5 are reported in Table 6.5. Notice that the axiom schemata P4 and P5 generate only finitely many axioms. More precisely, P4 generates $|\textsc{Act}|$ axioms, and P5 generates $|\textsc{Act}| \times (|\textsc{Act}| - 1)$ axioms. We will now prove the following result:

Theorem 6.7. *E_{p} is a finite, ground-complete axiomatization of $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ modulo bisimilarity.*

The idea behind the proof of Theorem 6.7 is that the axioms in Table 6.5 allow us to eliminate all occurrences of the parallel operator $|_{\textsc{Act}}$ from $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ processes. Hence, every $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ process can be proven equal to a BCCSP process using E_{p} . The ground-completeness of E_{p} then follows from that of E_0 proven in [126]. To that end, we first show:

Lemma 6.7. *For all closed* BCCSP *terms p and q, there exists a closed* BCCSP *term r such that $\mathsf{E}_{\mathsf{p}} \vdash p \mid_{\textsc{Act}} q \approx r$.*

Proof. The proof is by induction on size($p \mid_{\textsc{Act}} q$). First of all we notice that, given any closed BCCSP term p, we can assume, without loss of generality, that

$p = \sum_{i \in I} a_i p_i$ for some finite index set I, actions $a_i \in \textsc{Act}$, and closed BCCSP terms p_i, for $i \in I$. In fact, in case p is not already in this shape, then by applying axioms A2 and A4 in Table 6.2 we can remove superfluous occurrences of `nil` summands. In particular, we remark that this transformation does not increase the number of operator symbols occurring in p. Thus we proceed under the assumption that

$$p = \sum_{i \in I} a_i.p_i \quad \text{and} \quad q = \sum_{j \in J} b_i.q_j .$$

We proceed by a case analysis on the cardinality of the sets of indexes I and J.

- If either $I = \emptyset$ or $J = \emptyset$, then $p = \mathtt{nil}$ or $q = \mathtt{nil}$. In light of P1, without loss of generality, we can assume that $q = \mathtt{nil}$ and we have that $p \,|_{\textsc{Act}}\, q = p \,|_{\textsc{Act}}\, \mathtt{nil}$. Thus by applying axiom P3, we get $\mathsf{E_p} \vdash p \,|_{\textsc{Act}}\, q \approx \mathtt{nil}$ and we are done.

- If both I and J are singletons, then we have that $p = a.p'$ and $q = b.q'$, for some $a, b \in \textsc{Act}$ and BCCSP processes p' and q'.

  If $a = b$, then we use axiom P4 to get $\mathsf{E_p} \vdash a.p' \,|_{\textsc{Act}}\, a.q' \approx a.(p' \,|_{\textsc{Act}}\, q')$. Since the size of $p' \,|_{\textsc{Act}}\, q'$ is smaller than that of $p \,|_{\textsc{Act}}\, q$, by the induction hypothesis, there exists a BCCSP process r' such that $\mathsf{E_p} \vdash p' \,|_{\textsc{Act}}\, q' \approx r'$. Thus we have $\mathsf{E_p} \vdash a.p' \,|_{\textsc{Act}}\, a.q' \approx a.r'$, which is a BCCSP process.

  In the case that $a \neq b$, then we can use axiom P5, to infer $\mathsf{E_p} \vdash a.p' \,|_{\textsc{Act}}\, b.q' \approx \mathtt{nil}$ and we are done.

- We can now assume, without loss of generality, that $|I| > 1$ and $|J| \geq 1$. This means that we can express p as the summation of two summands of smaller size that are different from `nil`, i.e. $p = p_1 + p_2$, for some BCCSP processes p_1 and p_2. Then, we use axiom P2 to get $\mathsf{E_p} \vdash (p_1 + p_2) \,|_{\textsc{Act}}\, q \approx (p_1 \,|_{\textsc{Act}}\, q) + (p_2 \,|_{\textsc{Act}}\, q)$. Since both $p_1 \,|_{\textsc{Act}}\, q$ and $p_2 \,|_{\textsc{Act}}\, q$ have size less than that of $p \,|_{\textsc{Act}}\, q$, by the induction hypothesis, we have that there exist BCCSP processes r' and r'' such that $\mathsf{E_p} \vdash p_1 \,|_{\textsc{Act}}\, q \approx r'$ and $\mathsf{E_p} \vdash p_2 \,|_{\textsc{Act}}\, q \approx r''$. We thus have that $\mathsf{E_p} \vdash p \,|_{\textsc{Act}}\, q \approx r' + r''$, which is a BCCSP process, and we are done. □

The above lemma is the key step in the elimination of $|_{\textsc{Act}}$ from closed terms. Namely:

Proposition 6.4. *For every closed* $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ *process p there exists a closed* BCCSP *process q such that $\mathsf{E_p} \vdash p \approx q$.*

Proof. The proof is straightforward by structural induction on p and using Lemma 6.7 in the case that p is of the form $p_1 \,|_{\textsc{Act}}\, p_2$, for some $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ processes p_1, p_2. □ □

The ground-completeness of $\mathsf{E_p}$ over $\mathrm{BCCSP}^{\mathsf{p}}_{\textsc{Act}}$ follows from Proposition 6.4 and the ground-completeness of E_0 over BCCSP [126].

6.5 Axiomatizability Results for CCS Full Merge

In this section we apply the reduction technique described in Section 6.3 to show that bisimilarity does not have a finite, ground-complete equational axiomatization over the BCCSP extended with the full merge operator from CCS and either restriction or relabeling.

To this end, as already done in Subsections 6.4.2 and 6.4.3, we exploit the reduction technique from [12] and Moller's non-finite axiomatizability result from CCS_a (Theorem 6.2). In detail:

- We select a particular action $a \in \textsc{Act}$.
- We consider the language CCS_a and the instantiation of the equations φ_n in the family Φ over processes defined using only that action.
- We provide translation mappings from $\mathrm{CCS}^{\mathtt{r}}$ to CCS_a, and CCS^{ℓ} to CCS_a, denoted by $\mathtt{r}_a$, and ℓ_a respectively, whose definition will be parametric in the chosen action a. The first mapping will allow us to eliminate all $\mathrm{CCS}^{\mathtt{r}}$ terms in which the execution of a is restricted, while ensuring the possibility to perform any a-transition that is unrestricted. The second works by abstracting away of all action names and replacing them with a, while also completely ignoring all relabelings, since now specific action names are indistinguishable.

It will be then enough to show that the mappings $\mathtt{r}_a$ and ℓ_a are structural, that they preserves the soundness of equations from $\mathrm{CCS}^{\mathtt{r}}$ to CCS_a (CCS^{ℓ} to CCS_a respectively), and that they are ground Φ-reflecting, to obtain the validity of the lifting of the negative result in Theorem 6.2 to $\mathrm{CCS}^{\mathtt{r}}$ and CCS^{ℓ}.

In this section we will use a slightly augmented notation from the previous ones. The reason is that for the technical contributions of this section it was imperative for clarity to distinguish between the set of *labels* (of observable actions) and sets of actions that include internal steps.

In detail, we assume a finite set of action names $\textsc{Act}$, and we let $\overline{\textsc{Act}}$ denote the set of action co-names, i.e., $\overline{\textsc{Act}} = \{\overline{a} \mid a \in \textsc{Act}\}$. As usual, we postulate that $\overline{\overline{a}} = a$ and $a \neq \overline{a}$ for all $a \in \textsc{Act}$. Then, we let $\textsc{Act}_\tau = \textsc{Act} \cup \overline{\textsc{Act}} \cup \{\tau\}$, where $\tau \notin \textsc{Act} \cup \overline{\textsc{Act}}$. Henceforth, we let $\alpha, \beta, \ldots$ range over actions in $\textsc{Act}_\tau$, $\mu, \nu, \ldots$ range over actions in $\textsc{Act} \cup \overline{\textsc{Act}}$, and $a, b, \ldots$ range over actions in $\textsc{Act}$.

Following [152], the action symbol τ will result from the synchronized occurrence of the complementary actions a and $\overline{a}$, as described by the inference rules in Table 6.7.

The semantics of the full operator $\mid$, which is included in both languages we study in this section are given in Table 6.6.

6.5.1 The Case of Restriction

We denote by $\mathrm{CCS}^{\mathtt{r}}$ the recursion and relabeling free fragment of CCS with the full merge operator (denoted by $\mid$) generated by the following grammar:

$$t ::= \ \mathtt{nil} \ \mid \ x \ \mid \ \alpha.t \ \mid \ t+t \ \mid \ t \mid t \ \mid \ t \backslash R \qquad (\mathrm{CCS}^{\mathtt{r}})$$

$$\text{(r1)}\ \frac{t \xrightarrow{\alpha} t'}{t \mid u \xrightarrow{\alpha} t' \mid u} \qquad \text{(r2)}\ \frac{u \xrightarrow{\alpha} u'}{t \mid u \xrightarrow{\alpha} t \mid u'} \qquad \text{(r3)}\ \frac{t \xrightarrow{\mu} t' \quad u \xrightarrow{\overline{\mu}} u'}{t \mid u \xrightarrow{\tau} t' \mid u'}$$

Table 6.6: The SOS rules for $\mid$ operator ($\alpha \in \textsc{Act}_\tau$, $\mu \in \textsc{Act} \cup \overline{\textsc{Act}}$).

$$\text{(r4)}\ \frac{t \xrightarrow{\mu} t'}{t\backslash L \xrightarrow{\mu} t'\backslash R}\ \mu, \overline{\mu} \notin R \qquad \text{(r5)}\ \frac{t \xrightarrow{\tau} t'}{t\backslash L \xrightarrow{\tau} t'\backslash R}$$

Table 6.7: The SOS rules for the $\backslash$ operator ($\alpha \in \textsc{Act}_\tau$, $\mu \in \textsc{Act} \cup \overline{\textsc{Act}}$).

where $x \in \mathcal{V}$, $\alpha \in \textsc{Act}_\tau$ and $R \subseteq \textsc{Act} \cup \overline{\textsc{Act}}$.

We recall that the *restriction operator* $t\backslash R$ prevents t (and its derivatives) from performing any α-transition, for all $\alpha \in R$.

The operational semantics of $\text{CCS}^{\mathtt{r}}$ is obtained by adding the inference rules for the restriction operator given in Table 6.7 to the rules for BCCSP operators given in Table 6.1 and the full merge operator given in Table 6.6. In the technical results that follow, we will need to distinguish between transitions over $\text{CCS}^{\mathtt{r}}$ processes, and transitions over CCS_a processes. Hence, to avoid possible confusion, we adopt the same strategy we used in Section 6.4, and use special symbols to distinguish them: we denote the transition relation induced by the rules in Tables 6.1 and 6.7 by $\rightarrow_{\mathtt{r}}$, and bisimilarity over $\mathcal{P}(\text{CCS}^{\mathtt{r}})$ by $\sim_{\mathtt{r}}$.

Definition 6.7 (Bisimulation over $\text{CCS}^{\mathtt{r}}$). *Bisimulation relations over* $\text{CCS}^{\mathtt{r}}$ *processes are defined by applying Definition 2.2 to the LTS* $(\mathcal{P}(\text{CCS}^{\mathtt{r}}), \textsc{Act}_\tau, \rightarrow_{\mathtt{r}})$ *induced by the SOS rules in Table 6.7. We use the symbol* $\sim_{\mathtt{r}}$ *to denote bisimilarity over* $\text{CCS}^{\mathtt{r}}$ *processes.*

Our main goal here is to prove the following theorem:

Theorem 6.8. *Bisimilarity has no finite, ground-complete equational axiomatization over* $\text{CCS}^{\mathtt{r}}$*.*

We begin by providing the mapping. Choose an action a from the action set $\textsc{Act}$. Then we define a mapping $\mathtt{r}_a \colon \mathbb{T}(\text{CCS}^{\mathtt{r}}) \rightarrow \mathbb{T}(\text{CCS}_a)$ allowing us to rewrite any $\text{CCS}^{\mathtt{r}}$ term into a CCS_a term.

Definition 6.8 (The mapping $\mathtt{r}_a$). *The mapping* $\mathtt{r}_a \colon \mathbb{T}(\text{CCS}^{\mathtt{r}}) \rightarrow \mathbb{T}(\text{CCS}_a)$ *is defined inductively as follows:*

$$\mathtt{r}_a(\mathtt{nil}) = \mathtt{nil} \qquad \mathtt{r}_a(t+u) = \mathtt{r}_a(t) + \mathtt{r}_a(u)$$

$$\mathtt{r}_a(x) = x \qquad \mathtt{r}_a(t \mid u) = \mathtt{r}_a(t) \parallel \mathtt{r}_a(u)$$

$$\mathtt{r}_a(\alpha.t) = \begin{cases} a.\mathtt{r}_a(t) & \textit{if } \alpha = a \\ \mathtt{nil} & \textit{otherwise} \end{cases} \qquad \mathtt{r}_a(t\backslash R) = \begin{cases} \mathtt{r}_a(t) & \textit{if } a, \overline{a} \notin R \\ \mathtt{nil} & \textit{otherwise.} \end{cases}$$

Notice that a is the only action that may possibly occur in $\mathtt{r}_a(t)$, for each $t \in \mathbb{T}(\mathrm{CCS}^{\mathrm{r}})$.

We now proceed to show that the mapping $\mathtt{r}_a$ is a well-defined reduction, according to Definition 6.1. As a first step, we notice that $\mathtt{r}_a$ is structural by definition.

Lemma 6.8. *The mapping $\mathtt{r}_a$ is structural.*

We now proceed to prove two technical lemmas, that will be useful to prove that $\mathtt{r}_a$ is a reduction.

Lemma 6.9. *For all $p \in \mathcal{P}(\mathrm{CCS}^{\mathrm{r}})$, and $q \in \mathcal{P}(\mathrm{CCS}_a)$, if $\mathtt{r}_a(p) \xrightarrow{a} q$, then there exists some $p' \in \mathcal{P}(\mathrm{CCS}^{\mathrm{r}})$ such that $p \xrightarrow{a}_{\mathtt{r}} p'$ and $\mathtt{r}_a(p') = q$.*

Proof. The proof proceeds by structural induction over the $\mathcal{P}(\mathrm{CCS}^{\mathrm{r}})$ process p. As for prefixing, nondeterministic choice, and parallel composition the proof is analogous to that of the corresponding steps in Lemma 6.2, we limit ourselves to present only the inductive step related to the restriction operator.

Let $p = p_1 \backslash R$. We can distinguish two cases, according to whether $a \in R$ or $\overline{a} \in R$, or not (see Definition 6.8):

- Assume that $a \in R$ or $\overline{a} \in R$. Then $\mathtt{r}_a(p) = \mathtt{nil}$, and this case becomes vacuous as $\mathtt{r}_a(p) \overset{a}{\not\rightarrow}$.

- Assume now that $a, \overline{a} \notin R$. Then $\mathtt{r}_a(p) = \mathtt{r}_a(p_1)$ and $\mathtt{r}_a(p_1) \xrightarrow{a} q$. By induction over p_1, there is some $p_1' \in \mathcal{P}(\mathrm{CCS}^{\mathrm{r}})$ such that $p_1 \xrightarrow{a}_{\mathtt{r}} p_1'$ and $\mathtt{r}_a(p_1') = q$. Since $a, \overline{a} \notin R$, by an application of rule (r4) from Table 6.7 we obtain that $p \xrightarrow{a}_{\mathtt{r}} p_1' \backslash R$. Finally, by Definition 6.8, since $a, \overline{a} \notin R$ it follows that $\mathtt{r}_a(p_1' \backslash R) = \mathtt{r}_a(p_1') = q$ as required. □

Lemma 6.10. *For all $p, p' \in \mathcal{P}(\mathrm{CCS}^{\mathrm{r}})$, if $p \xrightarrow{a}_{\mathtt{r}} p'$, then $\mathtt{r}_a(p) \xrightarrow{a} \mathtt{r}_a(p')$.*

Proof. The proof proceeds by induction over the size of the proof for the transition $p \xrightarrow{a}_{\mathtt{r}} p'$. Also in this case, given the similarities with the proofs of the corresponding cases in Lemma 6.3, we limit ourselves to analyse only the case in which the last inference rule from Table 6.7 that is applied in the proof for $p \xrightarrow{a}_{\mathtt{r}} p'$ is rule (r4), i.e., the rule for restriction. (In particular, we remark that since $a \neq \tau$, rules (r3) and (r5) cannot be applied as the last rules in the proof for $p \xrightarrow{a}_{\mathtt{r}} p'$.)

Let (r4) be the last rule applied in the proof. In this case, $p = p_1 \backslash R$, $p_1 \xrightarrow{a}_{\mathtt{r}} p_1'$, and $p' = p_1' \backslash R$. In particular, the application of rule (r4) guarantees that $a, \overline{a} \notin R$, so that $\mathtt{r}_a(p) = \mathtt{r}_a(p_1)$, by Definition 6.8. By induction we obtain that $\mathtt{r}_a(p_1) \xrightarrow{a} \mathtt{r}_a(p_1')$. Clearly, this directly gives $\mathtt{r}_a(p) \xrightarrow{a} \mathtt{r}_a(p_1')$. Since, moreover, $a, \overline{a} \notin R$, by Definition 6.8 we also get that $\mathtt{r}_a(p') = \mathtt{r}_a(p_1' \backslash R) = \mathtt{r}_a(p_1')$. We can then conclude that $\mathtt{r}_a(p) \xrightarrow{a} \mathtt{r}_a(p')$. □

We now have all the ingredients necessary to prove that the mapping $\mathtt{r}_a$ is a well-defined ground Φ-reflecting reduction.

Proposition 6.5. *The mapping $\mathtt{r}_a$ satisfies the following properties:*

1. *For each $t, u \in \mathbb{T}(\mathrm{CCS}^{\mathrm{r}})$, $t \sim_{\mathtt{r}} u$ implies $\mathtt{r}_a(t) \sim \mathtt{r}_a(u)$.*

2. The mapping $\mathtt{r}_a$ is ground Φ-reflecting.

Proof. We prove the two statements separately.

1. First of all, for each $t \in \mathbb{T}(\mathrm{CCS}_a)$ we define $t^{\mathtt{r}}_a \in \mathbb{T}(\mathrm{CCS}^{\mathtt{r}})$ as follows:

$$\begin{array}{ccc} \mathtt{nil}^{\mathtt{r}}_a = \mathtt{nil} & x^{\mathtt{r}}_a = a & (a.t)^{\mathtt{r}}_a = a.t^{\mathtt{r}}_a \\ (t+u)^{\mathtt{r}}_a = t^{\mathtt{r}}_a + u^{\mathtt{r}}_a & & (t \parallel u)^{\mathtt{r}}_a = t^{\mathtt{r}}_a \mid u^{\mathtt{r}}_a. \end{array}$$

   It is then immediate to check that for each $t \in \mathbb{T}(\mathrm{CCS}_a)$ we have that $\mathtt{r}_a(t^{\mathtt{r}}_a) = t$. Then, given any CCS_a substitution σ, we define $\sigma^{\mathtt{r}}_a$ as the $\mathrm{CCS}^{\mathtt{r}}$ substitution such that $\sigma^{\mathtt{r}}_a(x) = (\sigma(x))^{\mathtt{r}}_a$. The claim then follows by applying the same reasoning used in the proof of Proposition 6.2.

2. Consider the family of equations $\Phi_{\mathtt{r}}$ defined as follows:

$$\varphi^n_{\mathtt{r}} : a \mid \sum_{i=1}^{n} a^i \approx a.\sum_{i=1}^{n} a^i + \sum_{j=1}^{n} a^{j+1} \qquad (n \geq 0)$$

$$\Phi_{\mathtt{r}} = \{\varphi^n_{\mathtt{r}} \mid n \geq 0\}.$$

   It is straightforward to prove that $\mathtt{r}_a(\varphi^n_{\mathtt{r}}) = \varphi_n$ for each $n \in \mathbb{N}$, and thus that $\mathtt{r}_a$ is ground Φ-reflecting. □

Theorem 6.8 is then a immediate consequence of Lemma 6.8, Proposition 6.5, Theorem 6.3, and Theorem 6.2.

6.5.2 The Case of Relabeling

We denote by CCS^{ℓ} the recursion and restriction free fragment of CCS with the full merge operator $\mid$ generated by the following grammar:

$$t ::= \quad \mathtt{nil} \quad \mid \quad x \quad \mid \quad \alpha.t \quad \mid \quad t+t \quad \mid \quad t \mid t \quad \mid \quad t[f] \qquad (\mathrm{CCS}^{\ell})$$

where $x \in \mathcal{V}$, $\alpha \in \mathrm{Act}_{\tau}$ and is a relabeling function $f : \mathrm{Act}_{\tau} \to \mathrm{Act}_{\tau}$ such that $f(\tau) = \tau$, and $f(\overline{a}) = \overline{f(a)}$ for each action $a \in \mathrm{Act}$.

Intuitively, $t[f]$ behaves like t, but each α transition that t (or any of its derivatives) can perform is transformed into an $f(\alpha)$ transition.

The operational semantics of CCS^{ℓ} is obtained by adding the inference rules for the relabeling operator given in Table 6.8 to the rules for BCCSP operators given in Table 6.1 and those for the full merge operator from Table 6.6.

In the technical results that follow, we will need to distinguish between transitions over CCS^{ℓ} processes, and transitions over CCS_a processes. As done in the previous sections, we use special symbols to distinguish them: we denote the transition relation induced by the rules in Tables 6.1 6.6, and 6.8 by $\to_{\ell}$, and bisimilarity over $\mathcal{P}(\mathrm{CCS}^{\ell})$ by $\sim_{\ell}$.

Definition 6.9 (Bisimulation over CCS^{ℓ}). *Bisimulation relations over* CCS^{ℓ} *processes are defined by applying Definition 2.2 to the LTS* $(\mathcal{P}(\mathrm{CCS}^{\ell}), \mathrm{Act}_{\tau}, \to_{\ell})$ *induced by the SOS rules of* CCS^{ℓ}*. We use the symbol* $\sim_{\ell}$ *to denote bisimilarity over* CCS^{ℓ} *processes.*

$$\text{(r6)}\ \frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]}$$

Table 6.8: The SOS rules for the $[f]$ operator (with f a symmetric relation in $\textsc{Act} \times \textsc{Act} \cup \{\tau, \tau\}$).

Our main goal here is to prove the following theorem:

Theorem 6.9. *Bisimilarity has no finite, ground-complete equational axiomatization over* CCS^{ℓ}.

Again, we will prove the above result using the reduction technique. We begin by providing the reduction mapping. Choose an action a from the action set $\textsc{Act}$. Then we define a mapping $\ell_a \colon \mathbb{T}(\mathrm{CCS}^{\ell}) \to \mathbb{T}(\mathrm{CCS}_a)$ allowing us to rewrite any CCS^{ℓ} term into a CCS_a term.

Definition 6.10 (The mapping ℓ_a)**.** *The mapping* $\ell_a \colon \mathbb{T}(\mathrm{CCS}^{\ell}) \to \mathbb{T}(\mathrm{CCS}_a)$ *is defined inductively as follows:*

$$\begin{array}{ll} \ell_a(\mathtt{nil}) = \mathtt{nil} & \qquad \ell_a(t+u) = \ell_a(t) + \ell_a(u) \\ \ell_a(x) = x & \qquad \ell_a(t \mid u) = \ell_a(t) \parallel \ell_a(u) \\ \ell_a(\alpha.t) = a.\ell_a(t) & \qquad \ell_a(t[f]) = \ell_a(t) \end{array}$$

Notice that a is the only action that may possibly occur in $\ell_a(t)$, for each $t \in \mathbb{T}(\mathrm{CCS}^{\ell})$.

We now proceed to show that the mapping ℓ_a is a well-defined reduction, according to Definition 6.1. As a first step, we notice that ℓ_a is structural by definition.

Lemma 6.11. *The mapping* ℓ_a *is structural.*

We now proceed to prove two technical lemmas that will be useful to prove that ℓ_a is a reduction.

Lemma 6.12. *For all* $p \in \mathcal{P}(\mathrm{CCS}^{\ell})$*, and* $q \in \mathcal{P}(\mathrm{CCS}_a)$*, if* $\ell_a(p) \xrightarrow{a} q$*, then there exists some* $p' \in \mathcal{P}(\mathrm{CCS}^{\ell})$ *and* $b \in \textsc{Act}_{\tau}$ *such that* $p \xrightarrow{b}_{\ell} p'$ *and* $\ell_a(p') = q$*.*

Proof. The proof proceeds by structural induction over the $\mathcal{P}(\mathrm{CCS}^{\ell})$ process p. As for nondeterministic choice and parallel composition the proof is analogous to that of the corresponding steps in Lemma 6.2, we limit ourselves to present only the cases for the action prefixing and relabeling operators.

Let $p = \alpha.p'$. By Definition 6.10 we have that $\ell_a(p) = a.\ell_a(p')$, and by assumption, $\ell_a(p) \xrightarrow{a} q$.

It follows that $\ell_a(p') = q$. Since $p \xrightarrow{\alpha}_{\ell} p'$, we are done.

Let $p = p_1[f]$. By Definition 6.10, we have that $\ell_a(p[f]) = \ell_a(p_1)$, and by assumption $\ell_a(p_1[f]) = \ell_a(p_1) \xrightarrow{a} q$. By induction over p_1, there are some $p_1' \in$

$\mathcal{P}(\mathrm{CCS}^\ell)$ and $b \in \textsc{Act}_\tau$ such that $p_1 \xrightarrow{b}_\ell p_1'$ and $\ell_a(p_1') = q$. By an application of rule (r6) from Table 6.8 we obtain that $p \xrightarrow{f(b)}_\ell p_1'[f]$, and $\ell_a(p_1'[f]) = q$, and the claim follows. □

Lemma 6.13. *For all $p, p' \in \mathcal{P}(\mathrm{CCS}^\ell)$, if $p \xrightarrow{\alpha}_\ell p'$, then $\ell_a(p) \xrightarrow{a} \ell_a(p')$.*

Proof. The proof proceeds by induction over the size of the proof for the transition $p \xrightarrow{a}_\ell p'$. Also in this case, given the similarities with the proofs of the corresponding cases in Lemma 6.3, we limit ourselves to analyse only the case in which the last inference rule from Table 6.8 that is applied in the proof for $p \xrightarrow{\alpha}_\ell p'$ is rule (r6), i.e., the rule for relabeling. Let (r6) be the last rule applied in the proof. In this case, $p = p_1[f]$, and there exists α' such that $f(\alpha') = \alpha$, and $p_1 \xrightarrow{\alpha'}_\ell p_1'$, and $p' = p_1'[f]$. By induction we obtain that $\ell_a(p_1) \xrightarrow{a} \ell_a(p_1')$. Clearly, by Definition 6.10 this directly gives $\ell_a(p) \xrightarrow{a} \ell_a(p')$, and we are done. □

We now have all the ingredients necessary to prove that the mapping ℓ_a is a well-defined ground Φ-reflecting reduction.

Proposition 6.6. *The mapping ℓ_a satisfies the following properties:*

1. *For each $t, u \in \mathbb{T}(\mathrm{CCS}^\ell)$, $t \sim_\ell u$ implies $\ell_a(t) \sim \ell_a(u)$.*
2. *The mapping ℓ_a is ground Φ-reflecting.*

Proof. We prove the two statements separately.

1. We omit the proof of the first statement since it follows the lines of those for previous statements (6.2), using Lemmas 6.12 and 6.13.
2. Consider the family of equations Φ_ℓ defined as follows:

$$\begin{aligned}
&\varphi_n^\ell : a \mid \sum_{i=1}^{n} a^i \approx a.\sum_{i=1}^{n} a^i + \sum_{j=1}^{n} a^{j+1} && (n \geq 0)\\
&\Phi_\ell = \{\varphi_n^\ell \mid n \geq 0\}.
\end{aligned}$$

   Notice that the equations in Φ_ℓ are sound modulo bisimilarity. It is straightforward to prove that $\ell_a(\varphi_\ell^n) = \varphi_n$ for each $n \in \mathbb{N}$, and thus that ℓ_a is ground Φ-reflecting. □

Theorem 6.9 is then a immediate consequence of Lemma 6.11, Proposition 6.6, Theorem 6.3, and Theorem 6.2.

6.6 Concluding Remarks

In this chapter, we have exploited the reduction technique from [12], for the lifting of negative results across process algebras, to prove the non-finite axiomatizability of various extensions of BCCSP modulo bisimilarity. In detail, we have proved that bisimilarity does not admit a finite, ground-complete axiomatization

- over $\mathrm{BCCSP}^{\mathtt{p}}_{A}$, i.e., BCCSP enriched with a CSP-like parallel operator $|_A$, with $A \subset \textsc{Act}$,
- over $\mathrm{BCCSP}^{\mathtt{p}}$, i.e., BCCSP enriched with CSP-like parallel operators with any possible synchronization set,
- over $\mathrm{CCS}^{\mathtt{r}}$, i.e., the recursion and relabeling free fragment of CCS and
- over CCS^{ℓ}, i.e., the recursion and restriction free fragment of CCS.

Interestingly, among all these negative results, we found a positive one: if we consider only the CSP-like parallel operator $|_{\textsc{Act}}$, forcing all the actions in the parallel components to be synchronized, then a finite, ground-complete axiomatization of bisimilarity over $\mathrm{BCCSP}^{\mathtt{p}}_{\textsc{Act}}$ exists.

As a natural step for future work, we plan to investigate how far the lifting technique of [12] can be pushed. In particular, we are interested in studying whether (some variations of) it can be used to lift known results for strong behavioral equivalences to their weak counterparts or to potentially extend results over weak behavioral congruences (such as Theorem 10 presented in [22]) to new settings.

Another possible direction for future work, would be to focus on full recursion free CCS. As a first step, we conjecture that a specific combination of our results over CCS restriction, and CCS relabeling, can be stated in order to acquire the relative result for recursion-free CCS with restriction, and relabeling.

Specifically, we have that since we now have the non finite axiomatizability result over CCS with restriction, we could use that as a target language. Then (to the best of our current knowledge), the same mapping defined for CCS with relabeling extended over CCS with restriction and relabeling, where the treatment of the restriction operator is just prepositional to the mapping, i.e.:

$$\widehat{t\backslash R} = \widehat{t}\,\backslash R\ ,$$

would yield a structural mapping from the source to the target language, wich would preserve the soundness of the family of equations 3.1, and thus extending the negative result to CCS with restriction and relabeling. We note here that the combination of the two mappings would not work if not applied in this order (i.e. from the complete language to the one with relabeling) under the current mappings. This is because the mapping defined for the case of restriction is in a way too strict, and forces a lot of terms to collapse to `nil`, something that in the presence of relabeling would unfortunately not work, as seemingly non necessary actions could have been defined as mapped to necessary via an appropriate f relabeling function.

Furthermore, Aceto, Ingólfsdóttir, Luttik and van Tilburg gave an equational axiomatization of bisimilarity over recursion-free CCS with interleaving parallel composition and the left-merge operator in [10]. That result crucially depends on the fact that restriction and relabeling distribute over interleaving parallel composition. On the other hand, neither restriction nor relabeling distribute over parallel composition in the presence of synchronization. Obtaining a complete axiomatization of full recursion free CCS modulo bisimilarity, with restriction, relabeling and parallel composition that allows for synchronization is a natural, and very challenging, avenue for future research.

Chapter 7

Complexity Through Translations For Modal Logic with Recursion

7.1 Introduction

We introduce a family of multi-modal logics with fixed-point operators that are interpreted on restricted classes of Kripke models. One can consider these logics as extensions of the usual multi-agent logics of knowledge and belief [94] by adding recursion to their syntax or of the μ-calculus [139] by interpreting formulas on different classes of frames and thus giving an epistemic interpretation to the modalities. We define *translations* between these logics, and we demonstrate how one can rely on these translations to prove finite-model theorems, complexity bounds, and tableau termination for each logic in the family.

Modal logic comes in several variations [60]. Some of these, such as multi-modal logics of knowledge and belief [94], are of particular interest to Epistemology and other application areas. Semantically, the classical modal logics used in epistemic (but also other) contexts result from imposing certain restrictions on their models. On the other hand, the modal μ-calculus [139] can be seen as an extension of the smallest normal modal logic **K** with greatest and least fixed-point operators, νX and μX respectively. We explore the situation where one allows both recursion (*i.e.* fixed-point) operators in a multi-modal language and imposes restrictions on the semantic models.

We are interested in the complexity of satisfiability for the resulting logics. Satisfiability for the μ-calculus is known to be EXP-complete [139], while for the modal logics between **K** and **S5** the problem is PSPACE-complete or NP-complete, depending on whether they have Negative Introspection [143, 117]. In the multi-modal case, satisfiability for those modal logics becomes PSPACE-complete, and is EXP-complete with the addition of a common knowledge operator [116].

There is plenty of relevant work on the μ-calculus on restricted frames, mainly in its single-agent form. Alberucci and Facchini examine the alternation hierarchy of the μ-calculus over reflexive, symmetric, and transitive frames in [32]. D'Agostino and Lenzi have studied the μ-calculus on different classes of frames in great detail. In [91], they reduce the μ-calculus over finite transitive frames to first-order logic. In [81], they prove that $\mathbf{S5}^\mu$-satisfiability is NP-complete, and that the two-agent version of $\mathbf{S5}^\mu$ does not have the finite model property. In [82], they consider finite symmetric frames, and they prove that $\mathbf{B}^\mu$-satisfiability is in 2EXP, and EXP-hard. They also examine planar frames in [83], where they show that the alternation hierarchy of the μ-calculus over planar frames is infinite.

Our primary method of proving complexity results is through translations to and from the multi-modal μ-calculus. We show that we can use surprisingly simple translations from modal logics without recursion to the base modal logic $\mathbf{K}_n$, reproving the PSPACE upper bound for these logics (Theorem 7.6 and Corollary 7.6.1). These translations and our constructions to prove their correctness do not generally transfer to the corresponding logics with recursion. We present translations from specific logics to the μ-calculus and back, and we discuss the remaining open cases. We discover, through the properties of our translations, that several behaviors induced on the transitions do not affect the complexity of the satisfiability problem. As a result, we prove that all logics with axioms among D, T, and 4, and the least-fixed-point fragments of logics that also have B, have their satisfiability in EXP, and a matching lower bound for the logics with axioms from D, T, B (Corollaries 7.11.1 and 7.11.2).

Finally, we present tableaux for the discussed logics, based on the ones by Kozen for the μ-calculus [139], and by Fitting and Massacci for modal logic [99, 150]. We give tableau-termination conditions for every logic with a finite model property (Theorem 7.13).

The addition of recursive operators to M_L increases expressiveness. An important example is that of *common knowledge* or *common belief*, which can be expressed with a greatest fixed-point thus: $\nu X.(\varphi \wedge \bigwedge_\alpha [\alpha]X)$. But the combination of epistemic logics and fixed-points can potentially express more interesting epistemic concepts. For instance, the formula $\mu X. \bigvee_\alpha([\alpha]\varphi \vee [\alpha]X)$, in the context of a belief interpretation, can be thought to claim that there is a rumour of φ. It would be interesting to see what other meaningful sentences of epistemic interest one can express using recursion. Furthermore, the family of logics we consider allows each agent to behave according to a different logic. This flexibility allows one to mix different interpretations of modalities, such as a temporal interpretation for one agent and an epistemic interpretation for another. Such logics can even resemble hyper-logics [75] if a set of agents represents different streams, and combinations of epistemic and temporal or hyper-logics have recently been used to express safety and privacy properties of systems [66].

The chapter is organized as follows. Section 7.2 gives the necessary background and an overview of current results. Section 7.3 defines a class of translations that provide us with several upper and lower bounds, and identifies conditions under which they can be composed. In Section 7.4 we finally give tableaux for our multimodal logics with recursion. We conclude in Section 7.5 with a set of open questions and directions.

7.2 Definitions and Background

This section introduces the logics that we study and the necessary background on the complexity of M_L and the μ-calculus.

7.2.1 The Multi-Modal Logics with Recursion

We start by defining the syntax of the logics.

Definition 7.1. *We consider formulas constructed from the following grammar:*

$$\begin{array}{llllllll} \varphi, \psi \in L ::= p & \mid \neg p & \mid \mathtt{tt} & \mid \mathtt{ff} & \mid X & \mid \varphi \wedge \psi & \mid \varphi \vee \psi \\ \quad \mid \langle \alpha \rangle \varphi & \mid [\alpha]\varphi & \mid \mu X.\varphi & \mid \nu X.\varphi, \end{array}$$

where X comes from a countable set of logical (or fixed-point) variables, LVar*, α from a finite set of agents,* Act*, and p from a finite set of propositional variables,* PVar*. When* Act $= \{\alpha\}$*, $\Box\varphi$ stands for $[\alpha]\varphi$, and $\Diamond\varphi$ for $\langle \alpha \rangle \varphi$. We also write $[A]\varphi$ to mean $\bigwedge_{\alpha \in A} [\alpha]\varphi$ and $\langle A \rangle \varphi$ for $\bigvee_{\alpha \in A} \langle \alpha \rangle \varphi$.*

A formula is closed when every occurrence of a variable X is in the scope of recursive operator νX or μX. Henceforth we consider only closed formulas, unless we specify otherwise.

Moreover, for recursion-free closed formulas we associate the notion of *modal depth*, which is the nesting depth of the modal operators[1]. The modal depth of φ is defined inductively as:

- $md(p) = md(\neg p) = md(\mathtt{tt}) = md(\mathtt{ff}) = 0$, where $p \in$ PVar,
- $md(\varphi \vee \psi) = md(\varphi \wedge \psi) = max(md(\varphi), md(\psi))$, and
- $md([a]\varphi) = md(\langle a \rangle \varphi) = 1 + md(\varphi)$, where $a \in$ Act.

We assume that in formulas, each recursion variable X appears in a unique fixed-point formula $\mathsf{fx}(X)$, which is either of the form $\mu X.\varphi$ or $\nu X.\varphi$. If $\mathsf{fx}(X)$ is a least-fixed-point (*resp.* greatest-fixed-point) formula, then X is called a least-fixed-point (*resp.* greatest-fixed-point) variable. We can define a partial order on fixed-point variables, such that $X \leq Y$ iff $\mathsf{fx}(X)$ is a subformula of $\mathsf{fx}(Y)$, and $X < Y$ when $X \leq Y$ and $X \neq Y$. If X is $\leq$-minimal among the free variables of φ, then we define the *closure* of φ to be $cl(\varphi) = cl(\varphi[\mathsf{fx}(X)/X])$, where $\varphi[\psi/X]$ is the usual substitution operation, and if φ is closed, then $cl(\varphi) = \varphi$.

We define $\mathsf{sub}(\varphi)$ as the set of subformulas of φ, and $|\varphi| = |\mathsf{sub}(\varphi)|$ is bounded by the length of φ as a string of symbols. Negation, $\neg\varphi$, and implication, $\varphi \to \psi$, can be defined in the usual way. Then, we define $\overline{\mathsf{sub}}(\varphi) = \mathsf{sub}(\varphi) \cup \{\neg\psi \in L \mid \psi \in \mathsf{sub}(\varphi)\}$.

[1] The modal depth of recursive formulas can be either zero, or infinite. However, this is not relevant for the spectrum of this work.

$$\begin{array}{llll}
\llbracket \mathtt{tt}, \rho \rrbracket = \mathbf{P}, & \llbracket \mathtt{ff}, \rho \rrbracket = \emptyset, & \llbracket p, \rho \rrbracket = \{s \mid p \in V(s)\}, & \llbracket \neg p, \rho \rrbracket = \mathbf{P} \backslash \llbracket p, \rho \rrbracket,
\end{array}$$

$$\begin{array}{ll}
\llbracket [\alpha]\varphi, \rho \rrbracket = \left\{ s \mid \forall t.\ sR_\alpha t \text{ implies } t \in \llbracket \varphi, \rho \rrbracket \right\}, & \llbracket \varphi_1 \wedge \varphi_2, \rho \rrbracket = \llbracket \varphi_1, \rho \rrbracket \cap \llbracket \varphi_2, \rho \rrbracket, \\
\llbracket \langle\alpha\rangle\varphi, \rho \rrbracket = \left\{ s \mid \exists t.\ sR_\alpha t \text{ and } t \in \llbracket \varphi, \rho \rrbracket \right\}, & \llbracket \varphi_1 \vee \varphi_2, \rho \rrbracket = \llbracket \varphi_1, \rho \rrbracket \cup \llbracket \varphi_2, \rho \rrbracket, \\
\llbracket \mu X.\varphi, \rho \rrbracket = \bigcap \left\{ S \mid S \supseteq \llbracket \varphi, \rho[X \mapsto S] \rrbracket \right\}, & \llbracket X, \rho \rrbracket = \rho(X), \\
\llbracket \nu X.\varphi, \rho \rrbracket = \bigcup \left\{ S \mid S \subseteq \llbracket \varphi, \rho[X \mapsto S] \rrbracket \right\}. &
\end{array}$$

Table 7.1: Semantics of modal formulas on a model $\mathsf{M} = (W, R, V)$. We omit M from the notation.

Semantics We interpret formulas on the states of a *Kripke model.* A Kripke model, or simply model, is a quadruple $M = (\mathbf{P}, R, V)$ where $\mathbf{P}$ is a nonempty set of states, $R \subseteq \mathbf{P} \times \text{ACT} \times \mathbf{P}$ is a transition relation, and $V : \mathbf{P} \to 2^{\text{PVAR}}$ determines on which states a propositional variable is true. $(\mathbf{P}, R)$ is called a *frame.* We usually write $(u, v) \in \xrightarrow{\alpha}$ or $u \xrightarrow{\alpha} v$ instead of $(u, \alpha, v) \in R$, or uRv, when ACT is a singleton $\{\alpha\}$.

Formulas are evaluated in the context of an *environment* $\rho : \text{LVAR} \to 2^{\mathbf{P}}$, which gives values to the logical variables. For an environment ρ, variable X, and set $S \subseteq \mathbf{P}$, we write $\rho[X \mapsto S]$ for the environment that maps X to S and all $Y \neq X$ to $\rho(Y)$. The semantics for our formulas is given through a function $\llbracket - \rrbracket_{\mathsf{M}}$, defined in Table 7.1. The semantics of $\neg\varphi$ are constructed as usual, where $\llbracket \neg X, \rho \rrbracket_{\mathsf{M}} = \mathbf{P} \backslash \rho(X)$.

We sometimes use $\mathsf{M}, s \models_\rho \varphi$ for $s \in \llbracket \varphi, \rho \rrbracket_{\mathsf{M}}$, and as the environment has no effect on the semantics of a closed formula φ, we often drop it from the notation and write $\mathsf{M}, s \models \varphi$ or $s \in \llbracket \varphi \rrbracket_{\mathsf{M}}$. If $\mathsf{M}, s \models \varphi$, we say that φ is true, or satisfied, in s. When the particular model does not matter, or is clear from the context, we may omit it.

Depending on how we further restrict our syntax and the model, we can describe several logics. Without further restrictions, the resulting logic is the μ-calculus [139]. The max-fragment (resp. min-fragment) of the μ-calculus is the fragment that only allows the νX (resp. the μX) recursive operator. If $|\text{ACT}| = k$ and we allow no recursive operators (or recursion variables), then we have the basic modal logic $\mathbf{K}_k$ (or $\mathbf{K}$, if $k = 1$), and further restrictions on the frames can result in a wide variety of modal logics (see [59]). We give names to the following frame conditions, or frame constraints, for the case where $\text{ACT} = \{\alpha\}$. These conditions correspond to the usual axioms for normal modal logics — see [60, 59, 94], which we will revisit in Section 7.3.

D**:** R is serial: $\forall s. \exists t. sRt$;

T**:** R is reflexive: $\forall s. sRs$;

B**:** R is symmetric: $\forall s, t. (sRt \Rightarrow tRs)$;

4: R is transitive: $\forall s, t, r$, if sRt and tRr then sRr;

5: R is euclidean: $\forall s, t, r$. if sRt and sRr, then tRr.

We consider modal logics that are interpreted over models that satisfy a combination of these constraints for each agent. D, which we call Consistency, is a special case of T, called Factivity. Constraint 4 is Positive Introspection and 5 is called Negative Introspection.[2] Given a logic $\mathbf{L}$ and constraint c, $\mathbf{L}+c$ is the logic that is interpreted over all models with frames that satisfy all the constraints of $\mathbf{L}$ and c. The name of a single-agent logic is a combination of the constraints that apply to its frames, including K, if the constraints are among 4 and 5. Therefore, logic $\mathbf{D}$ is $\mathbf{K}+D$, $\mathbf{T}$ is $\mathbf{K}+T$, $\mathbf{B}$ is $\mathbf{K}+B$, $\mathbf{K4}=\mathbf{K}+4$, $\mathbf{D4}=\mathbf{K}+D+4=\mathbf{D}+4$, and so on. We use the special names **S4** for **T4** and **S5** for **T45**. We define a (multi-agent) logic $\mathbf{L}$ on $\textsc{Act}$ as a map from agents to single-agent logics. $\mathbf{L}$ is interpreted on Kripke models of the form $(\mathbf{P}, R, V)$, where for every $\alpha \in \textsc{Act}$, $(\mathbf{P}, \xrightarrow{\alpha})$ is a frame for $\mathbf{L}(\alpha)$.

For a logic $\mathbf{L}$, $\mathbf{L}^\mu$ is the logic that results from $\mathbf{L}$ after we allow recursive operators in the syntax — in case they were not allowed in $\mathbf{L}$. Furthermore, if for every $\alpha \in \textsc{Act}$, $\mathbf{L}(\alpha)$ is the same single-agent logic $\mathbf{L}$, we write $\mathbf{L}$ as $\mathbf{L}_k$, where $|\textsc{Act}| = k$. Therefore, the μ-calculus is $\mathbf{K}_k^\mu$.

From now on, unless we explicitly say otherwise, by a logic, we mean one of the logics we have defined above. We call a formula satisfiable for a logic $\mathbf{L}$, if it is satisfied in some state of a model for $\mathbf{L}$.

***Example* 13.** *For a formula φ, we define $Inv(\varphi) = \nu X.(\varphi \wedge [\textsc{Act}]X)$. $Inv(\varphi)$ asserts that φ is true in* all *reachable states, or, alternatively, it can be read as an assertion that φ is common knowledge. We dually define $Eve(\varphi) = \mu X.(\varphi \vee \langle\textsc{Act}\rangle X)$, which asserts that φ is true in* some *reachable state.*

7.2.2 Known Results

For logic $\mathbf{L}$, the satisfiability problem for $\mathbf{L}$, or $\mathbf{L}$-satisfiability is the problem that asks, given a formula φ, if φ is satisfiable. Similarly, the model checking problem for $\mathbf{L}$ asks if φ is true at a given state of a given finite model.

Ladner [143] established the classical result of PSPACE-completeness for the satisfiability of **K**, **T**, **D**, **K4**, **D4**, and **S4** and NP-completeness for the satisfiability of **S5**. Halpern and Rêgo later characterized the NP–PSPACE gap for one-action logics by the presence or absence of Negative Introspection [117], resulting in Theorem 7.1. Later, Rybakov and Shkatov [180] proved the PSPACE-completeness of **B** and **TB**. For formulas with fixed-point operators, D'Agostino and Lenzi in [81] show that satisfiability for single-agent logics with constraint 5 is also NP-complete.

Theorem 7.1 ([143, 117, 180])**.** *If $\mathbf{L} \in \{\mathbf{K}, \mathbf{T}, \mathbf{D}, \mathbf{B}, \mathbf{TB}, \mathbf{K4}, \mathbf{D4}, \mathbf{S4}\}$, then $\mathbf{L}$-satisfiability is* PSPACE*-complete; and $\mathbf{L}+5$-satisfiability and $(\mathbf{L}+5)^\mu$-satisfiability is* NP*-complete.*

Theorem 7.2 ([116])**.** *If $k > 1$ and $\mathbf{L}$ has a combination of constraints from $D, T, 4, 5$ and no recursive operators, then $\mathbf{L}_k$-satisfiability is* PSPACE*-complete.*

[2] These are names for properties or axioms of a logic. When we refer to these conditions as conditions of a frame or model, we may refer to them with the name of the corresponding relation condition: seriality, reflexivity, symmetry, transitivity, and euclidicity.

Remark 12. *We note that Halpern and Moses in [116] only prove these bounds for the cases of* $\mathbf{K}_k, \mathbf{T}_k, \mathbf{S4}_k, \mathbf{KD45}_k$*, and* $\mathbf{S5}_k$*; and D'Agostino and Lenzi in [81] only prove the* NP*-completeness of satisfiability for* $\mathbf{S5}^\mu$*. However, it is not hard to see that their respective methods also work for the rest of the logics of Theorems 7.1 and 7.2.* ■

Theorem 7.3 ([139])**.** *The satisfiability problem for the* μ*-calculus is* EXP*-complete.*

Theorem 7.4 ([93])**.** *The model checking problem for the* μ*-calculus is in* NP $\cap$ coNP*.*[3]

Finally we have the following initial known results about the complexity of satisfiability, when we have recursive operators. Theorems 7.5 and 7.1 have already been observed in [18].

Theorem 7.5. *The satisfiability problem for the min- and max-fragments of the* μ*-calculus is* EXP*-complete, even when* $|\text{ACT}| = 1$*.*

Proof sketch. It is known that satisfiability for the min- and max-fragments of the μ-calculus (on one or more action) is EXP-complete. It is in EXP due to Theorem 7.3, and these fragments suffice [171] to describe the PDL formula that is constructed by the reduction used in [98] to prove EXP-hardness for PDL. Therefore, that reduction can be adjusted to prove that satisfiability for the min- and max-fragments of the μ-calculus is EXP-complete. □

It is not hard to express in logics with both frame constraints and recursion operators that formula φ is common knowledge, with formula $\nu X.\varphi \wedge [\text{ACT}]X$. Since validity for $\mathbf{L}_k$ with common knowledge (and without recursive operators) and $k > 1$ is EXP-complete [116][4], $\mathbf{L}_k^\mu$ is EXP-hard.

Proposition 7.1. *Satisfiability for* $\mathbf{L}_k^\mu$*, where* $k > 1$*, is* EXP*-hard.*

7.3 Complexity Through Translations

In this section, we examine **L**-satisfiability. We use formula translations to reduce the satisfiability of one logic to the satisfiability of another. We investigate the properties of these translations and how they compose with each other, and we achieve complexity bounds for several logics.

In the context of this chapter, a formula translation from logic $\mathbf{L}_1$ to logic $\mathbf{L}_2$ is a mapping f on formulas such that each formula φ is $\mathbf{L}_1$ -satisfiable if and only if $f(\varphi)$ is $\mathbf{L}_2$ -satisfiable. We only consider translations that can be computed in polynomial time, and therefore, our translations are polynomial-time reductions, transfering complexity bounds between logics.

According to Theorem 7.3, $\mathbf{K}_k^\mu$-satisfiability is EXP-complete, and therefore for each logic **L**, we aim to connect $\mathbf{K}_k^\mu$ and **L** via a sequence of translations in either direction, to prove complexity bounds for **L**-satisfiability.

[3] In fact, the problem is known to be in UP $\cap$ coUP [133].

[4] Similarly to Remark 12, [116] does not explicitly cover all these cases, but the techniques can be adjusted.

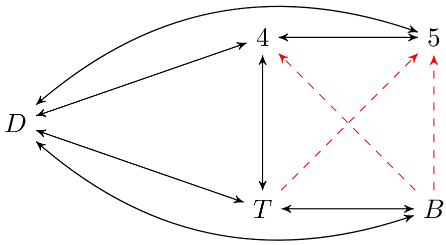

Figure 7.1: The frame property hierarchy

7.3.1 Translating Towards $\mathbf{K}_k$

We begin by presenting translations from logics with more to logics with fewer frame conditions. To this end, we study how taking the closure of a frame under one condition affects any other frame conditions.

Composing Frame Conditions

We now discuss how the conditions for frames affect each other. For example, to construct a transitive frame, one can take the transitive closure of a possibly non-transitive frame. The resulting frame will satisfy condition 4. As we see, taking the closure of a frame under condition x may affect whether that frame maintains condition y, depending on x and y. In the following we observe that one can apply the frame closures in certain orders that preserve the properties one aquires with each application.

Let $F = (W, R)$ be a frame, $\alpha \in A \subseteq \textsc{Act}$, and x a frame restriction among $T, B, 4, 5$. Then, $\overline{R_\alpha}^x$ is the closure of R_α under x, $\overline{R}^{x,A}$ is defined by $\overline{R_\beta}^{x,A} = \overline{R_\beta}^x$, if $\beta \in A$, and $\overline{R_\beta}^{x,A} = R_\beta$, otherwise. Then, $\overline{F}^{x,A} = (W, \overline{R}^{x,A})$. We make the following observation.

Lemma 7.1. *Let x be a frame restriction among $D, T, B, 4, 5$, and y a frame restriction among $T, B, 4, 5$, such that $(x, y) \neq (4, B), (5, T), (5, B)$. Then, for every frame F that satisfies x, $\overline{F}^y$ also satisfies x.*

According to Lemma 7.1, frame conditions are preserved as seen in Figure 7.1. In Figure 7.1, an arrow from x to y indicates that property x is preserved though the closure of a frame under y. Dotted red arrows indicate one-way arrows. For convenience, we define $\overline{F}^D = (W, \overline{R}^D)$, where $\overline{R}^D = R \cup \{(a, a) \in W^2 \mid \nexists\, (a, b) \in R\}$.

Remark 13. *We note that, in general, not all frame conditions are preserved through all closures under another condition. For example, the accessibility relation $\{(a, b), (b, b)\}$ is euclidean, but its reflexive closure $\{(a, b), (b, b), (a, a)\}$ is not.*

There is at least one linear ordering of the frame conditions $D, T, B, 4, 5$, such that all preceding conditions are preserved by closures under the following conditions. We call such an order a closure-preserving order. We use the linear order $D, T, B, 4, 5$ in the rest of the chapter.

Modal Logics

We start with translations that map logics without recursive operators to logics with fewer constraints. As mentioned in Subsection 7.2.2, all of the logics $\mathbf{L} \in \{\mathbf{K}, \mathbf{T}, \mathbf{D}, \mathbf{K4}, \mathbf{D4}, \mathbf{S4}\}$ and $\mathbf{L} + 5$ with one agent have known completeness results, and the complexity of modal logic is well-studied for multi-agent modal logics as well. The missing cases are very few and concern the combination of frame conditions (other than 5) as well as the multi-agent case. However we take this opportunity to present an intuitive introduction to our general translation method. In fact, the translations that we use for logics without recursion are surprisingly straightforward. Each frame condition that we introduced in Section 7.2 is associated with an axiom for modal logic, such that whenever a model has the condition, every substitution instance of the axiom is satisfied in all worlds of the model (see [60, 59, 94]). We give for each frame condition x and agent α, the axiom ax^x_α:

ax^D_α: $\langle\alpha\rangle\mathtt{tt}$ $\qquad$ ax^B_α: $\langle\alpha\rangle[\alpha]p \to p$ $\qquad$ ax^5_α: $\langle\alpha\rangle[\alpha]p \to [\alpha]p$

ax^T_α: $[\alpha]p \to p$ $\qquad$ ax^4_α: $[\alpha]p \to [\alpha][\alpha]p$

For each formula φ and $d \geq 0$, let $Inv_d(\varphi) = \bigwedge_{i \leq d}[\textsc{Act}]^i\varphi$. Our first translations are straightforwardly defined from the above axioms.

Translation 1 (One-step Translation)**.** Let $A \subseteq \textsc{Act}$ and let x be one of the frame conditions. For every formula φ, let $d = md(\varphi)$ if $x \neq 4$, and $d = md(\varphi)|\varphi|$, if $x = 4$. We define:

$$\mathsf{F}^x_A(\varphi) = \varphi \wedge Inv_d\left(\bigwedge_{\substack{\psi\in\overline{\mathsf{sub}}(\varphi)\\ \alpha\in A}} \mathsf{ax}^x_\alpha[\psi/p]\right).$$

Theorem 7.6. *Let $A \subseteq \textsc{Act}$, x be one of the frame conditions, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics without recursion operators, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + x$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ only includes frame conditions that precede x in the fixed order of frame conditions. Then, φ is $\mathbf{L}_1$-satisfiable if and only if $\mathsf{F}^x_A(\varphi)$ is $\mathbf{L}_2$-satisfiable.*

Proof. Assume first that $\mathsf{M}, w \models \varphi$, where M is an $\mathbf{L}_1$-model. For every subformula ψ of φ and $\alpha \in A$, $\mathsf{ax}^x_\alpha[cl(\psi)/p]$ is an instantiation of the axiom ax^x_α, and therefore it holds at all states of M that ar reachable from w. Thus, $\mathsf{M}, w \models Inv_d\left(\bigwedge_{\substack{\psi\in\mathsf{sub}(\varphi)\\ \alpha\in A}} \mathsf{ax}^x_\alpha[\psi/p]\right)$, which yields that $\mathsf{M}, w \models \mathsf{F}^x_A(\varphi)$.

For the other direction, assume that $\mathsf{M}, w \models \mathsf{F}^x_A(\varphi)$ for some $\mathbf{L}_2$-model $\mathsf{M} = (W, R, V)$ and state w. We assume that for every $\alpha, \beta \in \textsc{Act}$, if $\alpha \neq \beta$, then $R_\alpha \cap R_\beta = \emptyset$. For each $k \geq 0$, we define $W_k \subseteq W^* \times \mathbb{N}$ in the following way:

$W_0 := \{(w, 0)\}$; and $W_{k+1} = W_k \cup \{(pba, k+1) \mid \exists (pb, k) \in W_k, \alpha \in \textsc{Act}.\ bR_\alpha a\}$. Let $W_\infty = \bigcup_{k=0}^{\infty} W_k$. Let for each $\alpha \in \textsc{Act}$, $R^u_\alpha = \{((p, k), (pv, k+1)) \in W_\infty \times W_\infty \mid k \geq 0\}$ and $V^u(pv, k) = V(v)$ for all $k \geq 0$. Then, $\mathsf{M}^u = (W_\infty, R^u, V^u)$ is a bisimilar unfolding of M, and therefore for every formula ψ and $(pv, k) \in W_k$, $\mathsf{M}^u, v \models \psi$ if and only if $\mathsf{M}, v \models \psi$. Then, it is not hard to see that for every formula ψ and $(v, k) \in W_k$, $\mathsf{M}^u, (v, k) \models \psi$ if and only if $\mathsf{M}^c, (v, k) \models \psi$, where $\mathsf{M}^c = (W_\infty, R^c, V^u)$, where R^c is the closure of R^u under the conditions of $\mathbf{L}_2$.

Let $d = md(\varphi)$ if $x \neq 4$, and $d = md(\varphi)|\varphi|$, if $x = 4$.

We first handle the case for $x \in \{D, T\}$. Let $\mathsf{M}' = (W', R', V')$, where $W' = W_d$, R' is the (*resp.* D-closure of) the restriction of R^c on W' (*resp.* if $\mathbf{L}_2$ has constraint D), and V' is the restriction of V^u on W'. We observe that M' remains a $\mathbf{L}_2$-model: removing states only affects condition D, and the D-closure does not affect the other conditions. Furthermore, one can see that, by induction on ψ, for every formula ψ with $md(\psi) \leq d$, and every $v \in W_{d-md(\psi)}$, $\mathsf{M}, v \models \psi$ if and only if $\mathsf{M}', v \models \psi$: propositional cases are straightforward, and modal cases ensure that $md(\psi) > 0$, and therefore $v \in W_{d-1}$, thus the accessible states from v remain the same in M and in M'. Specifically, $\mathsf{M}', w \models \varphi$.

Let $\mathsf{M}^x = (W', \overline{R'}^x, V')$. It remains to prove that for every subformula ψ of φ, and every $v \in W_{d-md(\psi)}$, $\mathsf{M}', v \models \psi$ if and only if $\mathsf{M}^x, v \models \psi$. We continue by induction on the structure of ψ. The propositional and diamond cases are straightforward, since they are preserved by the introduction of pairs in the accessibility relation. We now consider the case of $\psi = [\alpha]\psi'$. We observe that $md(\psi) > 0$, and therefore $v \in W_{d-1}$. Specifically, if $md(\psi) = e$, then $v \in W_{d-e}$. If $\alpha \notin A$, then the accessible states from v remain the same in M' and in M^x, and we are done. Therefore, we assume that $\alpha \in A$. If there is some $v\overline{R'_\alpha}^x u$, but not $vR'_\alpha u$, then we take cases for x:

$x = D$ We observe that $md(\langle\alpha\rangle\mathtt{tt}) = 1$, and therefore $\mathsf{M}', v \models \langle\alpha\rangle\mathtt{tt}$, since we observe above that $v \in W_{d-1}$. This contradicts our assumtion that $v\overline{R'_\alpha}^x u$ but not $vR'_\alpha u$, due to definition of $\overline{R'_\alpha}^D$.

$x = T$ We observe that $md([\alpha]\psi' \to \psi') = md([\alpha]\psi') = e$, and therefore $\mathsf{M}', v \models [\alpha]\psi' \to \psi'$, yielding that $\mathsf{M}', v \models \psi'$, and, by the inductive hypothesis, $\mathsf{M}^x, v \models \psi'$. By the definition of the T-closure, $v = u$, and therefore $\mathsf{M}^x, u \models \psi'$.

We now consider the case for $x = B$. We construct model M^x similarly, and we then prove that for every subformula ψ of φ, and every $(v, d - md(\psi)) \in W_{d-md(\psi)}$, $\mathsf{M}', (v, d - md(\psi)) \models \psi$ if and only if $\mathsf{M}^x, v \models \psi$.

We examine the case for $\psi = [\alpha]\psi'$, where $a \in A$. If $(v, d-e)\overline{R'_\alpha}^x(u, k)$, but not $vR'_\alpha u$, then we see that $k = d - e - 1$, and $uR'_\alpha v$. Then, $\mathsf{M}', u \models \langle\alpha\rangle[\alpha]\psi' \to \psi'$, and therefore $\mathsf{M}', u \models \langle\alpha\rangle[\alpha]\psi'$ and we are done by the inductive hypothesis.

We now consider the case for $x = 4$. For each $(v, k) \in W_\infty$ and $\alpha \in \textsc{Act}$, let $b_\alpha(v, k) = \{[\alpha]\psi \in \mathsf{sub}(\varphi) \mid \mathsf{M}^u, (v, k) \models [\alpha]\psi\}$, and $d_\alpha(v, k) = \{\langle\alpha\rangle\psi \in \mathsf{sub}(\varphi) \mid$

$\mathsf{M}^u, (v,k) \models \langle\alpha\rangle\psi\}$. Observe that for all $(v,k)R_\alpha(v',k+1)$, where $\alpha \in A$ and $k \leq d$, $b_\alpha(v,k) \subseteq b_\alpha(v',k+1)$ and $d_\alpha(v',k+1) \subseteq d_\alpha(v,k)$. We call a state $(pv, k+1)$ *α-stable*, when $(p,k)R^u_\alpha(pv,k+1)$, and $b_\alpha(p,k) = b_\alpha(pv,k+1)$ and $d_\alpha(p,k) = d_\alpha(pv,k+1)$ for some $\alpha \in A$, and write $(p,k) \rhd_\alpha (pv,k+1)$. Observe that, by the Pigeonhole Principle, for $\alpha \in A$, in any sequence $(p,k)R_\alpha(pv_1,k+1)R_\alpha \cdots R_\alpha(pv_1\cdots v_l, k+l)$, where $l \geq |\varphi|$, there must be an α-stable state.

Let for each $\alpha \notin A$, $R^4_\alpha = R^u_\alpha$, and for each $\alpha \in A$,

$$R^4_\alpha = \{(a,b) \in R^u \mid a \text{ is not } \alpha\text{-stable}\} \cup \{(a,b) \mid \exists c \rhd_\alpha a.\ cR^u_\alpha b\}.$$

Observe that if $a \rhd_\alpha bR^u_\alpha c$, then c is not reachable from $(w,0)$ by R^4.

Let $\mathsf{M}_4 = (W', R^4, V')$ and $\mathsf{M}' = (W', R', V')$, where

$$W' = \{v \in W_d \mid v \text{ is reachable from } (w,0) \text{ by } R^4\},$$

R' is the closure of the restriction of R^4 on W' under the $\mathbf{L}_2$ constraints, and V' is the restriction of V^u on W'. M' is now a $\mathbf{L}_2$-model. We prove, by induction on ψ, that for every formula $\psi \in \mathsf{sub}(\varphi)$ and every $v = (p,k) \in W'$, where $k \leq d - md(\psi)|\varphi|$, $\mathsf{M}^u, v \models \psi$ if and only if $\mathsf{M}^4, v \models \psi$ if and only if $\mathsf{M}', v \models \psi$. The propositional cases are straightforward for both biimplications.

The modal cases $\psi = \langle\alpha\rangle\psi'$ or $\psi = [\alpha]\psi'$ ensure that $md(\psi) > 0$, and therefore $v \in W_{d-1}$, and there is some $(v,u) \in R^4_\alpha$. We first prove that $\mathsf{M}^u, v \models \psi$ if and only if $\mathsf{M}^4, v \models \psi$. If $\alpha \notin A$ or v is not α-stable, then this case follows from the observation that the accessible states from v in M^u and in M^4 are the same. If $\alpha \in A$ and $v' \lhd v$, then v' is not α-stable, because otherwise $v \notin W'$, as it would not be reachable from $(w,0)$ by R^4 in M^u. From $v' \lhd v$, we get that $\mathsf{M}^u, v' \models \psi$; by the inductive hypothesis, for every state $uR^4_\alpha v'$, $\mathsf{M}^u, u \models \psi'$ implies $\mathsf{M}^4, u \models \psi'$, and since v' has the same accessible states in M^u and in M^4, $\mathsf{M}^u, v' \models \psi$. This completes the induction, and we conclude that $\mathsf{M}^u, v \models \psi$ if and only if $\mathsf{M}^4, v \models \psi$.

To prove the second biimplication, note that we have that for every formula $\psi \in \mathsf{sub}(\varphi)$ and every $v = (pu,k) \in W'$, where $k \leq d - md(\psi)|\varphi|$, $((pu,k),(puu',k')) \in R^4_\alpha$. Furthermore, for every $((pu,k),(puu',k')) \in R^4_\alpha$, we have that $(u,u') \in R_\alpha$. We then see that, since M is an $\mathbf{L}_2$-model, for every $((pu,k),(puu',k')) \in R^4_\alpha$, it must be the case that $(u,u') \in R_\alpha$. We can then conclude that $\mathsf{M}^4, v \models \psi$ if and only if $\mathsf{M}', v \models \psi$. Specifically, $\mathsf{M}', w \models \varphi$.

Let $\mathsf{M}^x = (W', \overline{R'}^{x,A}, V')$. It is straightforward now to prove that for every subformula ψ of φ, and every $v \in W_{d-md(\psi)}$, $\mathsf{M}', v \models \psi$ if and only if $\mathsf{M}^x, v \models \psi$, and the proof is complete.

We now consider the case for $x = 5$. We construct model M^x similarly, and we then prove that for every subformula ψ of φ, and every $(v, d - md(\psi)) \in W_{d-md(\psi)}$, $\mathsf{M}', (v, d - md(\psi)) \models \psi$ if and only if $\mathsf{M}^x, v \models \psi$.

We examine the case for $\psi = [\alpha]\psi'$, where $\alpha \in A$. Let $v\overline{R'_\alpha}^x u$, but not $vR'_\alpha u$. It suffices to prove that $\mathsf{M}^u, u \models \psi'$. From our assumption that $v\overline{R'_\alpha}^x u$, but not $vR'_\alpha u$, the euclidean closure condition, and the tree-structure of M^u, we can see that there are some $a, b, c \in W'$, such that $aR'_\alpha b, c$, and v is R'_α-reachable from b, and u is R'_α-reachable from c. Observe that for every $\psi \in \mathsf{sub}(\varphi)$, $\alpha \in A$ and

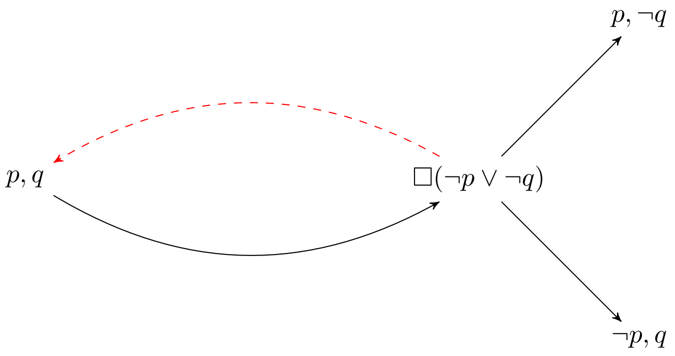

Figure 7.2: An example where the use of the alternative axiom for symetricity would wield a non-valid translation.

$v_1, v_2 \in W_d$, $v_1 R^c_\alpha v_2$, $\mathsf{M}^c, v_1 \models \langle\alpha\rangle\psi$ implies $\mathsf{M}^c, v_2 \models \langle\alpha\rangle\psi$, and $\mathsf{M}^c, v_2 \models [\alpha]\psi$ implies $\mathsf{M}^c, v_1 \models [\alpha]\psi$.

Assume that $\mathsf{M}^u, u \not\models \psi'$, to reach a contradiction. We can see that there is some u' that is R'_α-reachable from a and $u' R^u u$, such that $\mathsf{M}^u, u' \models \langle\alpha\rangle\neg\psi'$; in other words, $\mathsf{M}^u, u' \not\models [\alpha]\psi'$, and by the definition of ax^5_α, it must be the case that $\mathsf{M}^u, u' \not\models \langle\alpha\rangle[\alpha]\psi'$. Therefore, $\mathsf{M}^u, u' \models [\alpha]\neg\psi$, and by the definition of ax^5_α, it must be the case that $\mathsf{M}^u, a \models [\alpha]\neg\psi$, or, equivallently, $\mathsf{M}^u, a \models [\alpha]\langle\alpha\rangle\neg\psi'$. In turn, this yields that $\mathsf{M}^u, b \models \langle\alpha\rangle\neg\psi'$, and therefore $\mathsf{M}^u, v \models \neg\psi$, which is a contradiction. □

Remark 14. *As one can see we we used the contra-positive version of the axioms for symmetric and euclidean conditions on a frame. It turns out that the original axioms $p \to \Box\Diamond p$ for symmetricity ($\Diamond p \to \Box\Diamond p$ for euclidean) and would work with the translation we defined. To see this consider the case of a translation defined based on this version of the axiom. A non-symetric frame satisfying the translation could satisfy some formula of the form $\Box\psi'$ in a world s, but not satisfy ψ' in all the worlds predecessors (because at this point s's predecessors are not necessarily connected to it). The traslation would still be valid though, because $\Box\Diamond\psi'$ could be valid in s though it's successors. Thus there would not be a way to add symmetric edges to this model without losing the validity of $\Box\psi'$ in s. See figure 7.2 for such an example*

Corollary 7.6.1. *The satisfiability problem for every logic without fixed-point operators is in* PSPACE.

Modal Logics with Recursion

In the remainder of this section we will modify our translations and proof technique, in order to lift our results to logics with fixed-point operators. It is not clear whether the translations of Subsection 7.3.1 can be extended straightforwardly in the case of logics with recursion, by using unbounded invariance Inv, instead of the bounded Inv_d.

***Example* 14.** *Let $\varphi_f = \mu X.\Box X$, which requires all paths in the model to be finite, and thus it is not satisfiable in reflexive frames. In Subsection 7.3.1, to translate formulas from reflexive models, we did not need to add the negations of subformulas as conjuncts. In this case, such a translation would give*

$$\varphi_t := \varphi_f \wedge Inv((\Box\varphi_f \to \varphi_f) \wedge (\Box\Box\varphi_f \to \Box\varphi_f)).$$

*Indeed, on reflexive frames, the formulas $\Box\varphi_f \to \varphi_f$ and $\Box\Box\varphi_f \to \Box\varphi_f$ are valid, and therefore φ_t is equivalent to φ_f, which is **K**-satisfiable. This was not an issue in Subsection 7.3.1, as the finiteness of the paths in a model cannot be expressed without recursion.*

One would then naturally wonder whether conjoining over $\overline{\mathsf{sub}}(\varphi_f)$ in the translation would make a difference. The answer is affirmative, as the tranlation

$$\varphi_f \wedge Inv\left(\bigwedge_{\psi \in \overline{\mathsf{sub}}(\varphi_f)} \Box\psi \to \psi \right)$$

would then yield a formula that is not satisfiable. However, our constructions would not work to prove that such a translation preserves satisfiability. For example, consider $\mu X.\Box(p \to (r \wedge (q \to X)))$, whose translation is satisfied on a pointed model that satisfies at the same time p and q. We invite the reader to verify the details.

The only case where the approach that we used for the logics without recursion can be applied is for the case of seriality (condition D), as $Inv(\langle\alpha\rangle\mathtt{tt})$ directly ensures the seriality of a model.

Translation 2.

$$\mathsf{F}_A^{D^\mu}(\varphi) = \varphi \wedge Inv(\bigwedge_{\alpha\in A} \langle\alpha\rangle\mathtt{tt}).$$

Theorem 7.7. *Let $A \subseteq$ Act and $|\text{Act}| = k$, and let **L** be a logic, such that $\mathbf{L}(\alpha) = \mathbf{D}$ when $\alpha \in A$, and **K** otherwise. Then, φ is **L**-satisfiable if and only if $\mathsf{F}_A^{D^\mu}(\varphi)$ is $\mathbf{K}_k^\mu$-satisfiable.*

For the cases of reflexivity and transitivity, our simple translations substitute the modal subformulas of a formula to implicitly enforce the corresponding condition.

Translation 3. The operation $\mathsf{F}_A^{T^\mu}(-)$ is defined to be such that

- $\mathsf{F}_A^{T^\mu}([\alpha]\varphi) = [\alpha]\mathsf{F}_A^{T^\mu}(\varphi) \wedge \mathsf{F}_A^{T^\mu}(\varphi)$;
- $\mathsf{F}_A^{T^\mu}(\langle\alpha\rangle\varphi) = \langle\alpha\rangle\mathsf{F}_A^{T^\mu}(\varphi) \vee \mathsf{F}_A^{T^\mu}(\varphi)$;
- and it commutes with all other operations.

Theorem 7.8. *Let $\emptyset \neq A \subseteq \textsc{Act}$, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + T$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ at most includes frame condition D. Then, φ is $\mathbf{L}_1$-satisfiable if and only if $\mathsf{F}^{T^\mu}_A(\varphi)$ is $\mathbf{L}_2$-satisfiable.*

Proof. First we assume that $\mathsf{M}, w \models \varphi$, where $\mathsf{M} = (W, R, V)$ is an $\mathbf{L}_1$-model and $w \in W$. We can easily verify, by induction on every subformula ψ of φ on every environment ρ and state s of W, that $\mathsf{M}, s \models_\rho \psi$ if and only if $\mathsf{M}, s \models_\rho \mathsf{F}^{T^\mu}_A(\psi)$. We then conclude that $\mathsf{M}, w \models \mathsf{F}^{T^\mu}_A(\varphi)$, and thus the translated formula remains satisfiable.

For the converse direction, we assume an $\mathbf{L}_2$-model $\mathsf{M} = (W, R, V)$ and $w \in W$, such that $\mathsf{M}, w \models \mathsf{F}^{T^\mu}_A(\varphi)$. We construct an $\mathbf{L}_1$-model that satisfies φ: let $\mathsf{M}_p = (W, R^p, V^p)$, where for each $\alpha \in A$, $R^p_\alpha = R_\alpha \cup \{(u,u) \mid u \in W\}$, the reflexive closure of R_α, and for each $\alpha \notin A$, $R^p_\alpha = R_\alpha$. It now suffices to prove that for every $v \in W$, environment ρ, and $\psi \in \mathsf{sub}(\varphi)$, $\mathsf{M}, v \models_\rho \mathsf{F}^{T^\mu}_A(\psi)$ if and only if $\mathsf{M}_p, v \models_\rho \psi$. We proceed by induction on ψ. The cases of propositional and recursion variables and boolean connectives are straightforward. The cases of fixed-points are also not hard by using the inductive hypothesis. Finally, the modal cases use the form of the translation to show that boxes are preserved in the reflexive closure, and that no diamonds are introduced. □

Translation 4. The operation $\mathsf{F}^{4^\mu}_A(-)$ is defined to be such that

- $\mathsf{F}^{4^\mu}_A([\alpha]\psi) = \mathit{Inv}([\alpha](\mathsf{F}^{4^\mu}_A(\psi))$,
- $\mathsf{F}^{4^\mu}_A(\langle\alpha\rangle\psi) = \mathit{Eve}(\langle\alpha\rangle(\mathsf{F}^{4^\mu}_A(\psi))$,
- $\mathsf{F}^{4^\mu}_A(-)$ commutes with all other operations.

Theorem 7.9. *Let $\emptyset \neq A \subseteq \textsc{Act}$, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + 4$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ at most includes frame conditions D, T, B. Then, φ is $\mathbf{L}_1$-satisfiable if and only if $\mathsf{F}^{4^\mu}_A(\varphi)$ is $\mathbf{L}_2$-satisfiable.*

Proof. If $\mathsf{F}^{4^\mu}_A(\varphi)$ is satisfied in a model $M = (W, R, V)$, let $M' = (W, R^+, V)$, where R^+_α is the transitive closure of R_α, if $\alpha \in A$, and $R^+_\alpha = R_\alpha$, otherwise. It is now not hard to use induction on ψ to show that for every (possibly open) subformula ψ of φ, for every environment ρ, $\llbracket \mathsf{F}^{4^\mu}_A(\psi), \rho \rrbracket_{\mathsf{M}} = \llbracket \psi, \rho \rrbracket_{\mathsf{M}'}$. The other direction is more straightforward. □

In order to produce a similar translation for symmetric frames, we needed to use a more intricate type of construction. Moreover, we only prove the correctness of the following translation for formulas without least-fixed-point operators.

Translation 5. The operation $\mathsf{F}^{\mathbf{B}^\mu}_A(-)$ is defined as

$$\mathsf{F}^{\mathbf{B}^\mu}_A(\varphi) = \varphi \wedge \mathit{Inv}\big([\alpha]\langle\alpha\rangle p \wedge \bigwedge_{\psi \in \overline{\mathsf{sub}}(\varphi)} (\psi \to [\alpha][\alpha](p \to \psi))\big),$$

where p is a new propositional variable, not occurring in φ.

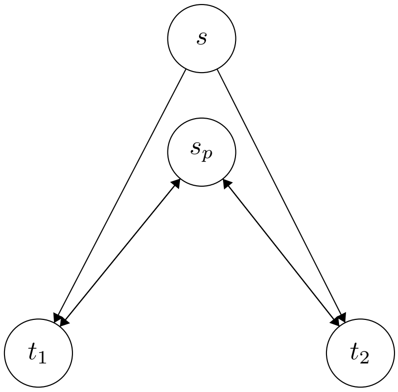

Figure 7.3: The new model, based on a world s in $\mathcal{M}$, with two neighbors t_1, t_2.

Theorem 7.10. *Let $\emptyset \neq A \subseteq$ ACT, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + \mathbf{B}$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ at most includes frame conditions D, T. Then, a formula φ that has no μX operators is $\mathbf{L}_1$-satisfiable if and only if $\mathsf{F}_A^{\mathbf{B}^\mu}(\varphi)$ is $\mathbf{L}_2$-satisfiable.*

Proof. First we assume that the μ-free formula φ is satisfied in M, w, where $\mathsf{M} = (W, R, V)$ is an $\mathbf{L}_1$-model and $w \in W$. We assume that $p \notin V(u)$ for every $u \in W$. We construct an $\mathbf{L}_2$-model that satisfies $\mathsf{F}_A^{B^\mu}(\varphi)$. Let $\mathsf{M}_p = (W_u \cup W_p, R^p, V^p)$ and $\mathsf{M}_u = (W_u \cup W_u, R^u, V^u)$, where:

- $\mathcal{M}_u$ is the unfolding of $\mathcal{M}$.
- $W_p = \{u^p \mid u \in W_u\}$ is a set of distinct copies of states from W_u;
- For each $u \in W_u$, $V^p(u) = V(u)$ and $V^p(u_p) = V(u) \cup \{p\}$.
- $R^p_\alpha = R^u_\alpha \cup \{(s, t_p), (t_p, s) \mid (s, t) \in R^u_\alpha\}$, if $\alpha \in A$, and $R^p_\alpha = R^u_\alpha$ otherwise.

This construction is demonstrated Figure 7.3.

It is easy to see that we have $Inv(\Box\Diamond p)$ in all worlds s of $\mathcal{M}_p$. Namely, for each original world in the unfolding, we have that if there existed any arrow from it, now there is a new world s_p at distance 2 from s, where p holds. Moreover, since $\mathcal{M}_u$ is an unfolding, all other worlds t_p in W_p are at distance strictly larger or smaller than 2 from s since they have been only connected to either s itself or to worlds that are not neighbors of s. Since the world s_p has identical neighbors and propositional variables as s we have that they are bissimilar and thus they satisfy the same formulae. For all worlds $w \in W_p$, we have that they can reach themselves in 2 steps, and for all other worlds they can reach in 2 steps (remember that M_u is an unfolding and thus each world has exactly one incoming edge), p does not hold. Thus we can also see that the whole translation of φ is satisfied in $\mathcal{M}'$. Furthermore, it is not hard to see that if we take the reflexive closure of R^p_α for each α such that $\mathbf{L}_2(\alpha)$ has condition T, then each state in the resulting model would be bisimilar to itself in M^p; and that the respective serial closure would not affect the model.

For the converse, we assume an $\mathbf{L}_2(\alpha)$ model $\mathsf{M} = (W, R, V)$ and $w \in W$, such that $\mathsf{M}, w \models \mathsf{F}_A^{B^\mu}(\varphi)$, and we construct an $\mathbf{L}_1$-model M' that satisfies φ. Let $\mathsf{M}' = (W, R', V)$, where R'_α is the symmetric closure of R_α, if $\alpha \in A$, and $R'_\alpha = R_\alpha$ otherwise. Let ρ be an environment such that for every Y, $\rho(Y) = [\![cl(Y)]\!]$. We prove that for every state $v \in W$ and $\psi \in \mathsf{sub}(\varphi)$, $\mathsf{M}, v \models_\rho \mathsf{F}_A^{B^\mu}(\psi)$ implies that $\mathsf{M}', v \models_\rho \psi$. First, notice that for every $\alpha \in A$ and $vR_\alpha v_1 R_\alpha v_2$, $\mathsf{M}, v \models_\rho \psi$ iff $\mathsf{M}, v_2 \models_\rho \mathsf{F}_A^{B^\mu}(\psi)$. We now proceed by induction on ψ.

- Propositional cases, the case of logical variables, and the case of $\psi = [\alpha]\psi'$, where $\alpha \notin A$, are straightforward.

- For the case of $\psi = \langle\alpha\rangle\psi'$, notice that introducing pairs in the accessibility relation preserves diamonds.

- For the case of $\psi = [\alpha]\psi'$, where $\alpha \in A$, let $(v, v') \in R'_\alpha$. It suffices to prove that $\mathsf{M}', v' \models \psi'$. If $(v, v') \in R_\alpha$, then we are done. If not, then $v'R_\alpha v$, and therefore there is some $(v, v_p) \in R_\alpha$, such that $\mathsf{M}, v_p \models p$. Therefore, $\mathsf{M}, v_p \models_\rho \psi'$, yielding $\mathsf{M}, v' \models_\rho \psi'$.

- For the case of $\psi = \nu Y.\psi'$, notice that $[\![\psi', \rho[Y \mapsto [\![cl(Y)]\!]_{\mathsf{M}}]]\!]_{\mathsf{M}'} = [\![\psi', \rho]\!]_{\mathsf{M}'} = [\![\psi', \rho]\!]_{\mathsf{M}}$, by the inductive hypothesis. Therefore, $[\![\psi', \rho]\!]_{\mathsf{M}} \subseteq [\![\psi', \rho]\!]_{\mathsf{M}'}$, which is what we wanted to prove. □

Remark 15. *A translation for euclidean frames and for the full syntax on symmetric frames would need different approaches. D'Agostino and Lenzi show in [81] that $\mathbf{S5}_2^\mu$ does not have a finite model property, and their result can be easily extended to any logic $\mathbf{L}$ with fixed-point operators, where there are at least two distinct agents α, β, such that $\mathbf{L}(\alpha)$ and $\mathbf{L}(\beta)$ have constraint B or 5. Therefore, as our constructions for the translations to $\mathbf{K}_k^\mu$ guarantee the finite model property to the corresponding logics, they do not apply to multimodal logics with B or 5.*

7.3.2 Embedding $\mathbf{K}_n^\mu$

In this subsection, we present translations from logics with fewer frame conditions to ones with more conditions. This will allow us to prove EXP-completeness in the following subsection. Let p, q be distinguished propositional variables that do not appear in our formulas. We let $\vec{p}$ range over p, $\neg p$, $p \wedge q$, $p \wedge \neg q$, and $\neg p \wedge q$.

Definition 7.2 (function *next*)**.** *$next(p \wedge q) = p \wedge \neg q$, $next(p \wedge \neg q) = \neg p \wedge q$, and $next(\neg p \wedge q) = p \wedge q$; and $next(p) = \neg p$ and $next(\neg p) = p$.*

We use a uniform translation from $\mathbf{K}_k^\mu$ to any logic with a combination of conditions D, T, B.

Translation 6. The operation $\mathsf{F}_A^{\mathbf{K}^\mu}(-)$ on formulas is defined such that:

- $\mathsf{F}_A^{\mathbf{K}^\mu}(\langle\alpha\rangle\psi) = \bigwedge_{\vec{p}}(\vec{p} \to \langle\alpha\rangle(next(\vec{p}) \wedge \mathsf{F}_A^{\mathbf{K}^\mu}(\psi)))$, if $\alpha \in A$;

- $\mathsf{F}_A^{\mathbf{K}^\mu}([\alpha]\psi) = \bigwedge_{\vec{p}}(\vec{p} \to [\alpha](next(\vec{p}) \to \mathsf{F}_A^{\mathbf{K}^\mu}(\psi)))$, if $\alpha \in A$;
- $\mathsf{F}_A^{\mathbf{K}^\mu}(-)$ commutes with all other operations.

We note that there are simpler translations for the cases of logics with only D ot T as a constraint, but the $\mathsf{F}_A^{\mathbf{K}^\mu}(-)$ is uniform for all the logics that we consider in this subsection.

Theorem 7.11. *Let $\emptyset \neq A \subseteq$ Act, $|\text{Act}| = k$, and let $\mathbf{L}$ be such that $\mathbf{L}(\alpha)$ includes only frame conditions from $\mathbf{D}, \mathbf{T}, \mathbf{B}$ when $\alpha \in A$, and $\mathbf{L}(\alpha) = \mathbf{K}$ otherwise. Then, φ is $\mathbf{K}^\mu_k$-satisfiable if and only if $\mathsf{F}_A^{\mathbf{K}^\mu}(\varphi)$ is $\mathbf{L}$-satisfiable.*

Proof. For the "only if" direction, let $\mathsf{M} = (W, R, V)$ be an unfolded model and $w \in W$ is its root, such that $\mathsf{M}, w \models \varphi$. Variables p and q do not appear in φ, so we can assume that $\mathsf{M}, w \models p$ and q, and that for each $vR_\alpha v'$, if $\alpha \in A$, then for each $\vec{p}$, $\mathsf{M}, v \models \vec{p}$ implies that $\mathsf{M}, v' \models next(\vec{p})$. Let $\mathsf{M}' = (W, R', V)$, where R is the appropriate closure of R under the conditions of L. We observe that for every $\alpha \in A$, $R_\alpha = R'_\alpha \cap \bigcup_{\vec{p}} [\![\vec{p}]\!] \times [\![next(\vec{p})]\!]$. We prove that for any environment ρ, any $\psi \in \mathsf{sub}(\varphi)$, and any $v \in W$, $\mathsf{M}, v \models_\rho \psi$ iff $\mathsf{M}', v \models_\rho \mathsf{F}_A^{\mathbf{K}^\mu}(\psi)$. The proof proceeds by induction on ψ. We fix a $v \in W$ and a $\vec{p}$, such that $\mathsf{M}, v \models \vec{p}$.

- The propositional cases and the case of $\psi = X$ are immediate.
- So are the cases of $\psi = \langle\alpha\rangle\psi'$ and $\psi = [\alpha]\psi'$, where $\alpha \notin A$.
- For the case of $\psi = \nu X.\psi'$ or $\psi = \mu X.\psi'$, note that due to the inductive hypothesis, for any $S \subseteq W$, $[\![\psi', \rho[X \mapsto S]]\!]_{\mathsf{M}} = \left[\!\left[\mathsf{F}_A^{\mathbf{K}^\mu}(\psi'), \rho[X \mapsto S]\right]\!\right]_{\mathsf{M}'}$, and therefore $[\![\psi, \rho]\!]_{\mathsf{M}} = \left[\!\left[\mathsf{F}_A^{\mathbf{K}^\mu}(\psi), \rho\right]\!\right]_{\mathsf{M}'}$.
- For the case of $\psi = \langle\alpha\rangle\psi'$, where $\alpha \in A$, $\mathsf{M}, v \models_\rho \psi$ iff $\mathsf{M}, v \models_\rho \langle\alpha\rangle\psi'$ iff there is some $(v, v') \in R_\alpha$, $\mathsf{M}, v' \models_\rho \psi'$ iff there is some $(v, v') \in R'_\alpha$, where $\mathsf{M}, v' \models next(\vec{p})$, $\mathsf{M}, v' \models_\rho \psi'$, iff there is some $(v, v') \in R'_\alpha$, $\mathsf{M}, v' \models next(\vec{p}) \wedge \psi'$ iff
$$\mathsf{M}, v' \models \mathsf{F}_A^{\mathbf{K}^\mu}(\langle\alpha\rangle\psi') = \bigwedge_{\vec{p}}(\vec{p} \to \langle\alpha\rangle(next(\vec{p}) \wedge \mathsf{F}_A^{\mathbf{K}^\mu}(\psi'))).$$
- For the case of $\psi = [\alpha]\psi'$, where $\alpha \in A$, $\mathsf{M}, v \models_\rho \psi$ iff $\mathsf{M}, v \models_\rho [\alpha]\psi'$ iff for every $(v, v') \in R_\alpha$, $\mathsf{M}, v' \models_\rho \psi'$ iff for every $(v, v') \in R'_\alpha$, where $\mathsf{M}, v' \models next(\vec{p})$, $\mathsf{M}, v' \models_\rho \psi'$, iff for every $(v, v') \in R'_\alpha$, $\mathsf{M}, v' \models next(\vec{p}) \to \psi'$ iff
$$\mathsf{M}, v' \models \mathsf{F}_A^{\mathbf{K}^\mu}([\alpha]\psi') = \bigwedge_{\vec{p}}(\vec{p} \to [\alpha](next(\vec{p}) \to \mathsf{F}_A^{\mathbf{K}^\mu}(\psi'))),$$
which completes the proof by induction.

For the "if" direction, let $\mathsf{M} = (W, R, V)$ be an L-model and $w \in W$, such that $\mathsf{M}, w \models \mathsf{F}_A^{\mathbf{K}^\mu}(\varphi)$. Let $\mathsf{M}' = (W, R', V)$, where for every $\alpha \in A$, $R'_\alpha = R_\alpha \cap \bigcup_{\vec{p}} [\![\vec{p}]\!] \times [\![next(\vec{p})]\!]$, and for every $\alpha \notin A$, $R'_\alpha = R_\alpha$. We can prove that for any environment ρ, any $\psi \in \mathsf{sub}(\varphi)$, and any $v \in W$, $\mathsf{M}, v \models_\rho \mathsf{F}_A^{\mathbf{K}^\mu}(\psi)$ iff $\mathsf{M}', v \models_\rho \psi$. The proof proceeds by induction on ψ and is very similar to the "only if" direction. □

7.3.3 Complexity Results

We observe that our translations all result in formulas of size at most linear with respect to the original. The exceptions are Translations 1 and 5, which have a quadratic cost.

Corollary 7.11.1. *If* $\mathbf{L}$ *only has frame conditions* D, T*, then its satisfiability problem is* EXP*-complete; if* $\mathbf{L}$ *only has frame conditions* $D, T, 4$*, then its satisfiability problem is in* EXP*.*

Proof. Immediately from Theorems 7.7, 7.8, 7.9, and 7.11. □

Corollary 7.11.2. *If* $\mathbf{L}$ *only has frame conditions* D, T, B*, then*

1. $\mathbf{L}$*-satisfiability is* EXP*-hard; and*

2. *the restriction of* $\mathbf{L}$*-satisfiability on formulas without* μX *operators is* EXP*-complete.*

Proof. Immediately from Theorems 7.7, 7.8, 7.10, and 7.11. □

7.4 Tableaux for $\mathbf{L}_k^\mu$

We give a sound and complete tableau system for logic $\mathbf{L}$. Furthermore, if $\mathbf{L}$ has a finite model property, then we give terminating conditions for its tableau. The system that we give in this section is based on Kozen's tableaux for the μ-calculus [139] and the tableaux of Fitting [99] and Massacci [150] for M_L. We can use Kozen's finite model theorem [139] to help us ensure the termination of the tableau for some of these logics.

Theorem 7.12 ([139])**.** *There is a computable* $\kappa : \mathbb{N} \to \mathbb{N}$*, such that every* $\mathbf{K}_k^\mu$*-satisfiable formula* φ *is satisfied in a model with at most* $\kappa(|\varphi|)$ *states.*[5]

Corollary 7.12.1. *If* $\mathbf{L}$ *only has frame conditions* $D, T, 4$*, then there is a computable* $\kappa : \mathbb{N} \to \mathbb{N}$*, such that every* $\mathbf{L}$*-satisfiable formula* φ *is satisfied in a model with at most* $\kappa(|\varphi|)$ *states.*

Proof. Immediately, from Theorems 7.12, 7.7, 7.8, and 7.9, and Lemma 7.1. □

Remark 16. *We note that not all modal logics with recursion have a finite model property — see Remark 15.*

Intuitively, a tableau attempts to build a model that satisfies the given formula. When it needs to consider two possible cases, it branches, and thus it may generate several branches. Each branch that satisfies certain consistency conditions, which we define below, represents a corresponding model.

[5] The tableau in [139] yields an upper bound of $2^{2^{O(n^3)}}$ for $\kappa_0(n)$, but that bound is not useful to obtain a "good" decision procedure. The purpose of this section is not to establish any good upper bound for satisfiability testing, which is done in Section 7.3.

$$\frac{\sigma\ \pi X.\varphi}{\sigma\ \varphi}\,(\mathsf{fix}) \qquad \frac{\sigma\ X}{\sigma\ \mathsf{fx}(X)}\,(\mathsf{X}) \qquad \frac{\sigma\ \varphi\vee\psi}{\sigma\ \varphi \mid \sigma\ \psi}\,(\mathsf{or}) \qquad \frac{\sigma\ \varphi\wedge\psi}{\begin{array}{c}\sigma\ \varphi\\ \sigma\ \psi\end{array}}\,(\mathsf{and})$$

$$\frac{\sigma\ [\alpha]\varphi}{\sigma.\alpha\langle\psi\rangle\ \varphi}\,(\mathsf{B}) \qquad \frac{\sigma\ \langle\alpha\rangle\varphi}{\sigma.\alpha\langle\varphi\rangle\ \varphi}\,(\mathsf{D}) \qquad \frac{\sigma\ [\alpha]\varphi}{\sigma.\alpha\langle\varphi\rangle\ \varphi}\,(\mathsf{d}) \qquad \frac{\sigma\ [\alpha]\varphi}{\sigma.\alpha\langle\psi\rangle\ [\alpha]\varphi}\,(\mathsf{4})$$

where, for rules (B) and (4), $\sigma.\alpha\langle\psi\rangle$ has already appeared in the branch; and for (D), σ is not α-flat.

$$\frac{\sigma.\alpha\langle\psi\rangle\ [\alpha]\varphi}{\sigma\ [\alpha]\varphi}\,(\mathsf{B5}) \qquad \frac{\sigma.\alpha\langle\psi\rangle\ \langle\alpha\rangle\varphi}{\sigma.\alpha\langle\psi\rangle.\alpha\langle\varphi\rangle\ \varphi}\,(\mathsf{D5}) \qquad \frac{\sigma.\alpha\langle\psi\rangle\ [\alpha]\varphi}{\sigma\ \varphi}\,(\mathsf{b}) \qquad \frac{\sigma\ [\alpha]\varphi}{\sigma\ \varphi}\,(\mathsf{t})$$

$$\frac{\sigma.\alpha\langle\psi\rangle\ [\alpha]\varphi}{\sigma.\alpha\langle\psi'\rangle\ [\alpha]\varphi}\,(\mathsf{B55}) \qquad \frac{\sigma.\alpha\langle\psi\rangle.\alpha\langle\psi'\rangle\ \langle\alpha\rangle\varphi}{\sigma.\alpha\langle\psi\rangle.\alpha\langle\varphi\rangle\ \varphi}\,(\mathsf{D55}) \qquad \frac{\sigma.\alpha\langle\psi\rangle\ [\alpha]\varphi}{\sigma\ [\alpha]\varphi}\,(\mathsf{b4})$$

where, for rule (B55), $\sigma.\alpha\langle\psi'\rangle$ has already appeared in the branch; for rule (D5), σ is not α-flat, and $\sigma\ \langle\alpha\rangle\varphi$ does not appear in the branch; for rule (D55), $\sigma\ \langle\alpha\rangle\varphi$ does not appear in the branch.

Table 7.2: The tableau rules for $\mathbf{L} = \mathbf{L}_n^{\mu}$.

Our tableaux use *prefixed formulas*, that is, formulas of the form $\sigma\ \varphi$, where $\sigma \in (\textsc{Act} \times L)^*$ and $\varphi \in L$; σ is the prefix of φ in that case, and we say that φ is prefixed by σ. We note that we separate the elements of σ with a dot. We say that the prefix σ is α-flat when α has axiom 5 and $\sigma = \sigma'.\alpha\langle\psi\rangle$ for some ψ. Each prefix possibly represents a state in a corresponding model, and a prefixed formula $\sigma\ \varphi$ declares that φ is satisfied in the state represented by σ. As we will see below, the prefixes from $(\textsc{Act} \times L)^*$ allow us to keep track of the diamond formula that generates a prefix through the tableau rules. For agents with condition 5, this allows us to restrict the generation of new prefixes and avoid certain redundancies, due to the similarity of euclidean binary relations to equivalence relations [161, 117].

The tableau rules that we use appear in Table 7.2. These include fixed-point and propositional rules, as well as rules that deal with modalities. Depending on the logic that each agent α is based on, a different set of rules applies for α: for rule (d), $\mathbf{L}(\alpha)$ must have condition D; for rule (t), $\mathbf{L}(\alpha)$ must have condition T; for rule (4), $\mathbf{L}(\alpha)$ must have condition 4; for rule (B5), (D5), and (D55), $\mathbf{L}(\alpha)$ must have condition 5; for (b) $\mathbf{L}(\alpha)$ must have condition B; and for (b4) $\mathbf{L}(\alpha)$ must have both B and 4. Rule (or) is the only rule that splits the current tableau branch into two. A tableau branch is propositionally closed when σ `ff` or both $\sigma\ p$ and $\sigma\ \neg p$

appear in the branch for some prefix σ. For each prefix σ that appears in a fixed tableau branch, let $\Phi(\sigma)$ be the set of formulas prefixed by σ in that branch. We use the notation $\sigma \prec \sigma'$ to mean that $\sigma' = \sigma.\sigma''$ for some σ'', in which case σ is an ancestor of σ'.

We define the relation $\xrightarrow{X}$ on prefixed formulas in a tableau branch as $\chi_1 \xrightarrow{X} \chi_2$, if $\frac{\chi_1}{\chi_2}$ is a tableau rule and χ_1 is not of the form $\sigma\ Y$, where $X < Y$; then, $\xrightarrow{X}^{+}$ is the transitive closure of $\xrightarrow{X}$ and $\xrightarrow{X}^{*}$ is its reflexive and transitive closure. We can also extend this relation to prefixes, so that $\sigma \xrightarrow{X} \sigma'$, if and only if $\sigma\ \psi \xrightarrow{X} \sigma'\ \psi'$, for some $\psi \in \Phi(\sigma)$ and $\psi' \in \Phi(\sigma')$. If in a branch there is a $\xrightarrow{X}$-sequence where X is a least fixed-point and appears infinitely often, then the branch is called fixed-point-closed. A branch is closed when it is either fixed-point-closed or propositionally closed; if it is not closed, then it is called open.

Now, assume that there is a $\kappa : \mathbb{N} \to \mathbb{N}$, such that every $\mathbf{L}$-satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states. An open tableau branch is called (*resp. locally*) *maximal* when all tableau rules (*resp. the tableau rules that do not produce new prefixes*) have been applied. A branch is called *sufficient* for φ when it is locally maximal and for every $\sigma\ \psi$ in the branch, for which a rule can be applied and has not been applied to $\sigma\ \psi$, $|\sigma| > |\textsc{Act}| \cdot \kappa(|\varphi|)^{|\varphi|^2} \cdot 2^{2|\varphi|+1}$. A tableau is called maximal when all of its open branches are maximal, and closed when all of its branches are closed. It is called sufficiently closed for φ if it is propositionally closed, or for some least fixed-point variable X, it has a $\xrightarrow{X}$-path, where X appears at least $\kappa(|\varphi|) + 1$ times. A sufficient branch for φ that is not sufficiently closed is called sufficiently open for φ.

A tableau for φ starts from $\varepsilon\ \varphi$ and is built using the tableau rules of Table 7.2. A tableau proof for φ is a closed tableau for the negation of φ.

Theorem 7.13 (Soundness, Completeness, and Termination of $\mathbf{L}^{\mu}_{k}$-Tableaux)**.** *From the following, the first two are equivalent for any formula $\varphi \in L$ and any logic $\mathbf{L}$. Furthermore, if there is a $\kappa : \mathbb{N} \to \mathbb{N}$, such that every $\mathbf{L}$-satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states, then all the following are equivallent.*

1. *φ has a maximal $\mathbf{L}$-tableau with an open branch;*
2. *φ is $\mathbf{L}$-satisfiable; and*
3. *φ has an $\mathbf{L}$-tableau with a sufficiently open branch for φ.*

Here we state and prove certain definitions and lemmata that will be used in the main proof of this Theorem. For an agent $\alpha \in \textsc{Act}$ and a state s in a model (W, R, V), let $Reach_\alpha(s)$ be the states that are reachable in W by R_α. We also use $R|_S$ for the restriction of a relation R on a set S.

For a logic $\mathbf{L}$, we call a state s in a model $\mathsf{M} = (W, R, V)$ for $\mathbf{L}$ *flat* when for every $\alpha \in \textsc{Act}$ for which $\mathbf{L}(\alpha)$ has constraint 5, there is a set of states W_0, such that:

- $Reach_\alpha(s) = \{s\} \cup W_0$;

- $R_\alpha|_{Reach_\alpha(s)} = E_0 \cup E_1$, where
  $E_0 \subseteq \{s\} \times W_0$ and
  $E_1 = W_0^2$; and
- if $\mathbf{L}(\alpha)$ has constraint T, or $E_0 \neq \emptyset$ and $\mathbf{L}(\alpha)$ has constraint B, then $s \in W_0$.

Lemma 7.2 ([161, 117])**.** *Every pointed* $\mathbf{L}$*-model is bisimilar to* $\mathbf{L}$*-model whose states are all flat.*

We are now ready to proceed with our proof of Theorem 7.13.

Proof. **To prove that 1 implies 2,** let b be a maximal open branch in the tableau for φ. We construct a $\mathbf{K}_k^\mu$-model $\mathsf{M} = (W, R, V)$ for φ in the following way. Let W be the set of prefixes that appear in the branch, and let, for each $\alpha \in \textsc{Act}$,

$$R_\alpha^0 = \{\sigma, \sigma.\alpha\langle\psi\rangle \in W^2\} \cup \left\{ \sigma, \sigma \in W^2 \mid \mathbf{L}(\alpha) \text{ has } \begin{array}{c} \text{reflexive frames, or} \\ \text{serial frames and} \\ \forall\psi.\sigma.\alpha\langle\psi\rangle \notin W^2 \end{array} \right\};$$

R_α^1 is the symmetric closure of R_α^0, if $\mathbf{L}(\alpha)$ has symmetric frames, and it is R_α^0 otherwise; R_α^2 is the euclidean closure of R_α^1, if $\mathbf{L}(\alpha)$ has euclidean frames, and it is R_α^1 otherwise; and finally, R_α is the transitive closure of R_α^2, if $\mathbf{L}(\alpha)$ has transitive frames, and it is R_α^2 otherwise. By Lemma 7.1, R_α satisfies all the necessary closure conditions. We also set $V(p) = \{\sigma \in W \mid \sigma\ p \text{ appears in the branch } \}$.

It is now possible to prove, by straightforward induction, that for every subformula ψ of φ, if $\sigma\ \psi$ appears in the branch, then for any environment ρ, such that $\{\sigma' \in W \mid \sigma\ \psi \xrightarrow{X}{}^{*} \sigma'\ X\} \subseteq \rho(X)$, $\sigma \in [\![\psi, \rho_{\sigma,\psi}]\!]$. The only interesting cases are fixed-point formulas, so let $\psi = \nu X.\psi'$. Let S_X be the set of prefixes of X in the branch. We can immediately see that if $\sigma\ X$ appears in the branch, then so does $\sigma\ \psi'$, and therefore, by the inductive hypothesis, $S_X \subseteq [\![\psi, \rho[X \mapsto S_X]]\!]$. From the semantics in Table 7.1, $\sigma \in [\![\psi, \rho]\!]$.

On the other hand, if $\psi = \mu X.\psi'$, then we prove that if $\sigma \notin S \subseteq W$, then $S \not\supseteq [\![\psi', \rho[X \mapsto S]]\!]$. Let $\Psi = \{\sigma'\ \chi S \mid \sigma\ \psi' \xrightarrow{X}{}^{*} \sigma'\ \chi \text{ and } \sigma' \notin S\}$. We know that $\sigma\ \psi' \in \Psi$, so $\Psi \neq \emptyset$. There are no infinite $\xrightarrow{X}$-paths in the branch, so there is some $\sigma'\ \psi' \in \Psi$, such that $\sigma'\ \psi' \not\xrightarrow{X}{}^{+} \sigma''\ \psi'$ for any σ''. Then, we see that $\{\sigma'' \in W \mid \sigma'\ \psi \xrightarrow{X}{}^{*} \sigma''\ X\} \subseteq S$, because if $\sigma'\ \psi \xrightarrow{X}{}^{*} \sigma''\ X$, then $\sigma'\ \psi' \xrightarrow{X}{}^{*} \sigma''\ \psi'$. But then, by the inductive hypothesis, $\sigma' \in [\![\psi', \rho[X \mapsto S]]\!]$, and therefore $S \not\supseteq [\![\psi', \rho[X \mapsto S]]\!]$, which was what we wanted to prove.

To prove that 2 implies 3, let $\mathsf{M} = (W, R, V)$ be a $\mathbf{L}$-model and $w \in W$, such that $\mathsf{M}, w \models \varphi$, and W has at most $\kappa(|\varphi|)$ states. Let the environment ρ be such that $\rho(X) = [\![\mathsf{fx}(X), \rho]\!]_{\mathsf{M}}$ for every variable X. We say that $\rightarrow \subseteq (W \times L)^2$ is a dependency relation on M when it satisfies the following conditions:

- if $u \models_\rho \psi_1 \wedge \psi_2$, then $(u, \psi_1 \wedge \psi_2) \rightarrow (u, \psi_1)$ and $(u, \psi_1 \wedge \psi_2) \rightarrow (u, \psi_2)$;

- if $u \models_\rho \psi_1 \vee \psi_2$, then $(u, \psi_1 \vee \psi_2) \to (u, \psi_1)$ and $u \models_\rho \psi_1$ or $(u, \psi_1 \wedge \psi_2) \to (u, \psi_2)$ and $u \models_\rho \psi_2$;
- if $u \models_\rho [\alpha]\psi$, then $(u, [\alpha]\psi) \to (v, \psi)$ for all $v \in W$, such that $uR_\alpha v$;
- if $u \models_\rho \langle\alpha\rangle\psi$, then $(u, \langle\alpha\rangle\psi) \to (v, \psi)$ for some $v \in W$, such that $uR_\alpha v \models \psi$;
- if $u \models_\rho \mu X.\psi$ or $u \models_\rho \nu X.\psi$, then $(u, \mathsf{fx}(X)) \to (u, \psi)$; and
- if $u \models_\rho X$, then $(u, X) \to (u, \mathsf{fx}(X))$.

For each variable X and dependency relation $\to$, we also define $\xrightarrow{X}$, such that $(w_1, \psi_1) \xrightarrow{X} (w_2, \psi_2)$ whenever $(w_1, \psi_1) \to (w_2, \psi_2)$ and $\psi_1 \neq Y$ for all variables Y where $\mathsf{fx}(Y)$ is not a subformula of $\mathsf{fx}(X)$. We call a dependency relation $\to$ lfp-finite, if for every least-fixed-point variable X, X appears finitely many times on every $\xrightarrow{X}$-sequence.

Claim: There is a lfp-finite dependency relation. The claim amounts to the memoryless determinacy of parity games and it can be proven similarly, as in [203]. Thus, we fix such a lfp-finite dependency relation.

The tableau starts with $\varepsilon\ \varphi$ and we can keep expanding this branch to a sufficient one using the tableau rules, such that every prefix is mapped to a state in W, whenever σ is mapped to u, $\mathsf{M}, u \models_\rho \psi$ for every $\psi \in \Phi(\sigma)$, and if $\sigma.\alpha\langle\psi\rangle$ to v, then $uR_\alpha v$; furthermore, this can be done by following the lfp-finite dependency relation. This is by straightforward induction on the application of the tableau rules. A special case are the agents with euclidean accessibility relations, for which we can use Lemma 7.2. It is not hard to see that in this way we generate a set of branches that are not propositionally closed. Furthermore, since the tableau rule applications follow a lfp-finite dependency relation and W has at most $\kappa(|\varphi|)$ states, it is not hard to see that for every least-fixed-point variable X, on every $\xrightarrow{X}$-path, X appears at most $\kappa(|\varphi|)$ times.

To prove that 3 implies 1, we assume that there is a sufficient open branch b in a tableau for φ and we demonstrate that φ has a maximal tableau with an open branch — specifically, we construct such an open branch from b.

We call a prefix σ a leaf when there is no $\sigma' \neq \sigma$ in the branch, such that $\sigma \prec \sigma'$; we call σ productive when a tableau rule on a formula $\sigma\ \psi$ in the branch can produce a new prefix. We call σ ready if it is of the form $\sigma'.\alpha\langle\psi_1\rangle.\alpha\langle\psi_2\rangle$, or if σ is not α-flat for any $\alpha \in \textsc{Act}$.

For each $\sigma\ \psi$ in b and each least-fixed-point variable X, let

$$c(\sigma\ \psi, X) = \max\{n \mid \text{there is a } \xrightarrow{X} \begin{array}{l}\text{-path that ends in } \sigma\ \psi,\\ \text{where } X \text{ appears } n \text{ times}\end{array}\}.$$

Since b is not sufficiently closed, always $c(\sigma\ \psi, X) \leq \kappa(|\varphi|)$. We use the notation $\sigma \sim \sigma'$ to mean that $\sigma = \sigma_1.\alpha\langle\psi_1\rangle$, $\sigma = \sigma_2.\alpha\langle\psi_2\rangle$ for some $\sigma_1, \sigma_2, \alpha$, and $\Phi(\sigma) = \Phi(\sigma')$; and the notation $\sigma \equiv \sigma'$ to mean that $\sigma \sim \sigma'$ for every least-fixed-point variable X and $\psi \in \Phi(\sigma)$, $c(\sigma\ \psi, X) = c(\sigma'\ \psi, X)$ and either both are ready, or both σ and σ' are not ready and $\sigma + 1 \sim \sigma_2$, where $\sigma = \sigma_1.\alpha\langle\psi\rangle$ and $\sigma' = \sigma_2.\alpha\langle\psi'\rangle$. We then say that σ and σ' are equivalent.

Every productive leaf σ in the branch has an ancestor $e(\sigma)$ that has more that has a distinct, ready, and $\equiv$-equivalent ancestor; let $s(\sigma)$ be such an ancestor of $e(\sigma)$. We further assume that $e(\sigma)$ is the $\prec$-minimal ancestor of σ with these properties. We note that for any two productive leaves σ_1 and σ_2, if $e(\sigma_1) \prec \sigma_2$, then $e(\sigma_1) = e(\sigma_2)$.

Let b_0 be the branch that results by removing from b all prefixed formulas of the form $\sigma.\alpha\langle\psi_1\rangle\ \psi_2$, where $e(\sigma') \prec \sigma$ for some productive leaf σ'. Observe that b_0 is such that each of its productive leafs has an equivalent ancestor that is not a leaf. To complete the proof, it suffices to show how to extend any branch b_i, where each of its productive leafs σ has an equivalent proper ancestor $s(\sigma)$, to a branch b_{i+1} that preserves this property, and has an increased minimum length of its productive leafs. To form b_{i+1}, simply add to b_i all formulas of the form $\sigma.\sigma'\ \psi$, where $s(\sigma).\sigma'\ \psi$ appears in b_i.

Finally, observe that the finite model property of $\mathbf{L}$ was only used to prove equivalence with the third statement of the theorem. □

Corollary 7.13.1. $\mathbf{L}$*-tableaux are sound and complete for* $\mathbf{L}$*.*

***Example* 15.** *Let* $\text{ACT} = \{a, b\}$ *and* $\mathbf{L}$ *be a logic, such that* $\mathbf{L}(a) = \mathbf{K}^\mu$ *and* $\mathbf{L}(b) = \mathbf{K5}^\mu$*. Let*

$$\varphi_1 = (p \wedge \langle a\rangle p) \wedge \mu X.(\neg p \vee [a]X) \qquad \textit{and} \qquad \varphi_2 = \langle b\rangle p \wedge \mu X.([b]\neg p \vee [b])X.$$

As we see in Figure 7.4, the tableau for φ_1 *produces an open branch, while the one for* φ_2 *has all of its branches closed, the leftmost one due to an infinite* $\xrightarrow{X}$*-sequence.*

7.5 Conclusions

We studied multi-modal logics with recursion. These logics mix the frame conditions from epistemic modal logic, and the recursion of the μ-calculus. We gave simple translations among these logics that connect their satisfiability problems. This allowed us to offer complexity bounds for satisfiability and to prove certain finite model results. We also presented a sound and complete tableau that has termination guarantees, conditional on a logic's finite model property.

Conjectures and Future Work We currently do not posses full translations for the cases of symmetric and euclidean frames. What is interesting is that we also do not have a counterexample to prove that the translations that we already have, as well as other attempts, are *not* correct. In the case of symmetric frames, we have managed to prove that our construction works for formulas without least-fixed-point operators. A translation for euclidean frames and for the full syntax on symmetric frames is left as future work. We know that we cannot use the same

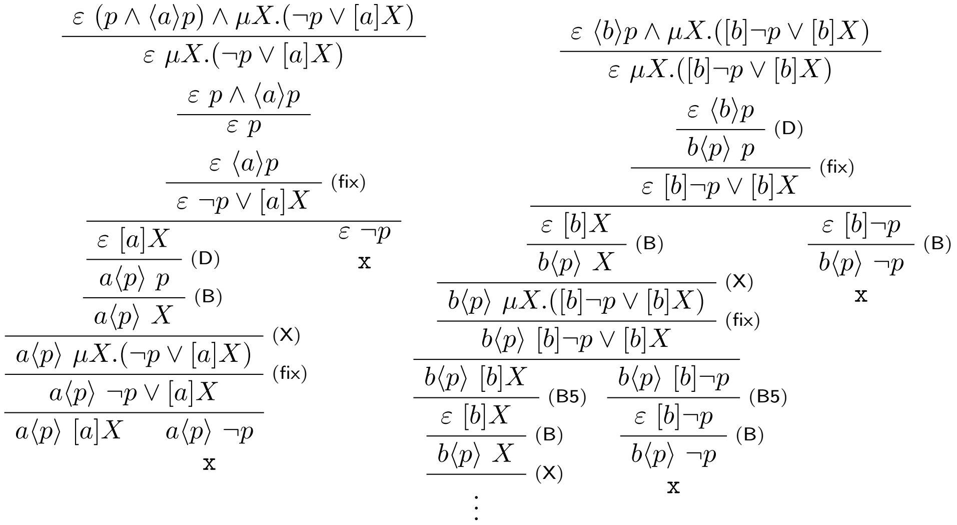
$\varepsilon\ (p \wedge \langle a\rangle p) \wedge \mu X.(\neg p \vee [a]X)$
$\varepsilon\ \mu X.(\neg p \vee [a]X)$
$\varepsilon\ p \wedge \langle a\rangle p$
$\varepsilon\ p$
$\varepsilon\ \langle a\rangle p$
(fix)
$\varepsilon\ \neg p \vee [a]X$
$\varepsilon\ [a]X$
(D)
$a\langle p\rangle\ p$
(B)
$a\langle p\rangle\ X$
(X)
$a\langle p\rangle\ \mu X.(\neg p \vee [a]X)$
(fix)
$a\langle p\rangle\ \neg p \vee [a]X$
$a\langle p\rangle\ [a]X$
$a\langle p\rangle\ \neg p$
x
$\varepsilon\ \neg p$
x
$\varepsilon\ \langle b\rangle p \wedge \mu X.([b]\neg p \vee [b]X)$
$\varepsilon\ \mu X.([b]\neg p \vee [b]X)$
$\varepsilon\ \langle b\rangle p$
(D)
$b\langle p\rangle\ p$
(fix)
$\varepsilon\ [b]\neg p \vee [b]X$
$\varepsilon\ [b]X$
(B)
$b\langle p\rangle\ X$
(X)
$b\langle p\rangle\ \mu X.([b]\neg p \vee [b]X)$
(fix)
$b\langle p\rangle\ [b]\neg p \vee [b]X$
$b\langle p\rangle\ [b]X$
(B5)
$\varepsilon\ [b]X$
(B)
$b\langle p\rangle\ X$
(X)
$\vdots$
$b\langle p\rangle\ [b]\neg p$
(B5)
$\varepsilon\ [b]\neg p$
(B)
$b\langle p\rangle\ \neg p$
x
$\varepsilon\ [b]\neg p$
(B)
$b\langle p\rangle\ \neg p$
x

Figure 7.4: Tableaux for φ_1 and φ_2. The dots represent that the tableau keeps repeating as from the identical node above. The x mark represents a propositionally closed branch.

model constructions that preserve the finiteness of the model as in Subection 7.3.1 (see Remark 15).

We do not prove the finite model property on all logics. We note that although it is known that logics with recursion with at least two agents with either B or 5 do not have this property (see 15, [81]), the situation is unclear if there is only one such agent.

We further conjecture that it is not possible to prove EXP-completenes for all the single-agent cases. Specifically, we expect $\mathbf{K4}^{\mu}$-satisfiability to be in PSPACE, similarly to how $\mathbf{K5}^{\mu}$-satisfiability is in NP[81]. As such, we do not expect Translation 6 to be correct for these cases.

The model checking problem for the μ-calculus is an important open problem. The problem does not depend on the frame restrictions of the particular logic, though one may wonder whether additional frame restrictions would help solve the problem more efficiently. We are not aware of a way to use our translations to solve model checking more efficiently.

As, to the best of our knowledge, most of the logics described in this chapter have not been explicitly defined before, with notable exceptions such as [91, 81, 32], they also lack any axiomatizations and completeness theorems. We do expect the classical methods from [139, 143, 116] and others to work out in these cases as well. However it would be interesting to see if there are any unexpected situations that arise.

Given the importance of common knowledge for epistemic logic and the fact that it has been known that common knowledge can be thought of as a (greatest) fixed-point already from [118, 49], we consider the logics that we presented to be natural extensions of M_L. Besides the examples given in Section 7.2, we are interested in exploring what other natural concepts can be defined with this enlarged language.

Part III

Epilogue

Chapter 8

Closing Remarks

8.1 Summary of the Contributions

This thesis, focuses on obtaining negative results in the theory of algebraic process description languages and modal logics. In particular, we offered several non-finite-axiomatizability results for process algebras modulo bisimulation- and trace-based notions of behavioral equivalence (Chapters 3-6), and proved lower and upper bounds on the complexity for the satisfiability problem for various modal logics and their extensions with fixed-point operators (Chapter 7). All the results we presented in the dissertation were proved using techniques rooted in the syntactic nature of those formalisms for describing reactive systems and their specifications. Specifically, we discovered the following results:

- BPA enriched with a priority operator does not afford a finite axiomatization modulo order insensitive bisimilarity (Chapter 3).
- Closed recursion-free regular monitors over a finite set of actions afford finite axiomatizations, both modulo verdict and ω-verdict equivalence. However, the landscape of the axiomatizability results is more varied in the setting of open terms — that is, terms that may include variables. In the presence of more than one, but finitely many, actions, we prove that no finite axiomatization exists, and we provide an infinite one that we prove complete. We then perform an exhaustive study for the remaining cases for terms over one action or an infinite set of actions, modulo both equivalences, and show that over a singleton action set, we can acquire completeness with finitely many axioms. In the case of infinitely many actions, our axiomatic basis for closed terms extended by only one axiom is also complete for open terms (Chapter 4).
- We extend Moller's negative result over open terms in CCS modulo bisimilarity to the weak setting modulo rooted weak, and rooted branching bisimilarity (Chapter 5).

- We further study Moller's negative result and prove it applies to several new settings over languages obtained as extensions of BCCSP through different operators. The technique we employed here is based on creating mappings from different process algebraic languages onto Moller's CCS. Our main contribution was identifying the core similarities between behaviors of the used operators and, by exploiting those, defining the mappings as mentioned above. Our first lifting of Moller's result was BCCSP extended with the merge operator $|_A$, for $A \subseteq \textsc{Act}$, from CSP. We considered the cases where the resulting process algebras contain exactly one such operator for a fixed A or when they contain all such operators, one for each A. In the first case, we show that BCCSP extended with those operators does not afford a finite ground-complete axiomatization when $A \subset \textsc{Act}$. When $A = \textsc{Act}$, instead, we provide a finite ground-axiomatization for bisimilarity. Moreover, in the second case, we showed that without the presence of τ actions, our proof technique through mappings could not be applied to prove the non-existence of a finite ground-complete axiomatization. Finally, we created two new mappings and used them to prove two negative results, one for the restriction- and recursion-free, and one for the relabeling- and recursion-free fragments of CCS (Chapter 6).

- Finally, in Chapter 7, we defined the multi-agent modal μ-calculus over frames satisfying modal axioms and studied the complexity of the satisfiability problem. Our research produced tight lower and upper bounds for several sublogics employing many such axioms and at least some hardness results for other ones. We also provided sound and complete tableaux for these logics and used them to prove the decidability of the satisfiability problem in cases that were not addressed in the literature. However, perhaps the most significant contribution of this final study was our novel methodology, which advocates for a more uniform tackling of complexity questions over modal logics.

Overall, we think that our work demonstrates the benefits of developing and applying general and "abstract" methodologies that can be used to prove negative results. It advocates a transition from a case-by-case analysis specific to a given setting to using proof techniques that can be instantiated to show results in various settings. This is akin to the transition from showing specific complexity-theoretic results using ingenuity and problem-specific information to developing so-called "algorithmic meta-theorems"—see, for instance, the survey article [142].

Such transitions from the individual study of specific problems and their associated results to the development of general theories covering classes of those take place as a scientific field matures and scientists discover commonalities amongst disparate problems. In the personal journey through the work presented in this dissertation, the step from problem-specific to general techniques was initially motivated mainly by intuition at the time of each individual work, but eventually, it became a more conscious choice. The reason is that the use of more general methods seemed more "productive", in the sense that at least the length of the resulting arguments was decreased, and the answers they provided were easier to generalize and reuse.

However, one downside of replacing well-established and streamlined methods with these generalizations is that we have to count more on intuition and be at more risk for unexpected mistakes. Namely, we have to be more creative (and lucky) in our search for similarities between mathematical theories, and it is possible to not succeed in that, even though similarities exist. Moreover, as the solutions were based on less established or novel methods, we did not have experience using them or access to possible earlier examples to help us. This meant that some proof steps were harder to complete, and often we would not know how to tackle or could overlook some cases of the proof.

Luckily though, since proofs became shorter, potential difficult stages occurred fewer times once we had managed to identify the necessary patterns. Moreover, even though a lengthier proof could have more manageable individual steps, this does not necessarily decrease the chances of mistakes. That was because at least for the author, longer proofs tended to increase the chances of mistakes occurring, regardless of the familiarity with the techniques and the simplicity of the steps. Another outcome for the author was the empirical realization of this fact, which led to the subsequent preference for “more general” approaches.

8.2 Future Work

The work presented above is by no means the last stone to be laid for any of the studied fields. We have already provided specific avenues for future work based on our contributions included in this thesis, which can be found in the relevant parts of Chapters 3-7. However, there is one last question we would like to present to conclude the research presented in this dissertation, which we will describe shortly.

Aside from this final question, we plan to build on this thesis by developing general techniques that are more applicable than those that are currently at our disposal. For example, the reduction technique used in Chapter 6 could not be applied in the setting of BCCSP with all the $|_A$ operators from CSP and no τ actions. Moreover, to date, it has never been successfully employed to prove negative results for “weak congruences”, whereas Moller’s proof technique has.

We reserve the remainder of this section to present a final avenue of research as an epilogue to the work presented already. The reason for selecting this question for this purpose is twofold. First, its theme is very relevant to this thesis. However, it is not tied enough to any of the specific contributions to be considered part of their future work. Secondly, our current attempts to answer this question were heavily affected by the new knowledge and expertise that the included contributions provided to the author.

8.2.1 Towards an Infinite Axiomatic Basis of CCS Modulo Bisimilarity

We begin by reminding the reader of the role played by an infinite and complete axiomatic basis over Mon_F modulo verdict equivalence, in Section 4.5, towards acquiring a negative result. There, having this axiomatization enabled the use of

(A0)	$x + \mathtt{nil} \approx x$	(P0)	$x \parallel \mathtt{nil} \approx x$
(A1)	$x + y \approx y + x$	(P1)	$x \parallel y \approx y \parallel x$
(A2)	$(x + y) + z \approx x + (y + z)$		
(A3)	$x + x \approx x$		

Table 8.1: Basic axioms for CCS. $\mathsf{E}_0 = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{A3}\}$ and $\mathsf{E}_1 = \mathsf{E}_0 \cup \{\mathrm{P0}, \mathrm{P1}\}$.

a proof technique based on compactness ([148, 105]), which simplified and streamlined the proof of the non-finite axiomatizability result in Theorem 4.10. This is a classic proof technique for showing non-finite axiomatizability results ([181]), but it does require the existence of an infinite equational basis. However, this simplification motivated us to see whether similar infinite axiomatizations could be useful elsewhere. Here we will discuss the possibility for a similar axiomatic basis of CCS modulo bisimilarity, and explore its possible uses. We note that no such equational axiomatization is known for CCS without the use of auxiliary operators.

In our work, Moller's negative axiomatizability result for bisimilarity over CCS has been heavily used and extended. In Chapter 5 we managed to use some equations provided by Moller [160] to extend his result to rooted weak bisimilarity and rooted branching bisimilarity over open CCS terms. Our conjecture is that having an infinite complete axiomatization over CCS will enable us to extend more results over CCS in the weak equivalence setting. Here we present our initial steps towards producing a similar result for open terms in the recursion, restriction and relabeling-free fragment of CCS modulo bisimilarity.

Our search began as the attempt to identify all possible variations of Moller's M_n equations, used in Chapter 5. Indeed, we managed to produce several more. In the setting of closed terms, an infinite complete axiomatization has already been produced. However, since that axiomatization is over closed equations, the argument of Chapter 4 cannot be applied. This is because in order to apply that argument one needs a complete axiomatization over open terms.

The first study of the equational characterization of parallel composition was carried out by Hennessy and Milner, in their seminal paper [125] (preliminary version of [126]). There, they provided a ground-complete axiomatization of CCS modulo bisimilarity. This axiomatization consisted of $\mathsf{E}_0 = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{A3}\}$ given in Table 8.1, which is a ground-complete axiomatization of BCCSP modulo bisimilarity (a proof can be found, e.g., in [193]), enriched with the axiom schema EL in Table 8.2, known as *the expansion law*.

The expansion law was used to deal with parallel composition: it states that whenever the initial behavior of the two parallel components is known, then the initial behavior of their composition can be described explicitly by the term on the right-hand side of equation EL. Informally, as parallel composition does not distribute over choice in either of its arguments, modulo bisimilarity, the only way to describe equationally the initial behavior of a closed term of the form $p \parallel q$ is first to express p as $\sum_{i \in I} \mu_i p_i$ and q as $\sum_{j \in J} \nu_j q_j$. One can then apply the expansion

$$\begin{aligned}&\sum_{i\in I}\mu_i x_i \parallel \sum_{j\in J}\nu_j y_j \approx \\ &\sum_{i\in I}\mu_i(x_i \parallel \sum_{j\in J}\nu_j y_j) + \sum_{j\in J}\nu_j(\sum_{i\in I}\mu_i x_i \parallel y_j) + \sum_{\substack{i\in I, j\in J\\ \mu_i=\overline{\nu_j}}}\tau(x_i \parallel y_j)\end{aligned} \tag{EL}$$

Table 8.2: The expansion law.

law, from left to right, to eliminate all occurrences of parallel composition from CCS processes, reducing them to BCCSP processes, and then use the ground-completeness of E_0 over BCCSP to conclude that the axiom system extending E_0 with all the instances of EL is ground-complete over CCS modulo bisimilarity.

Theorem 8.1 (Hennessy and Milner [125, 126]). *The axiom system E_0 extended by all instances of the EL schema, is a ground-complete axiomatization of* CCS *modulo bisimilarity.*

However, the axiomatization proposed by Hennessy and Milner was *not finite.* In fact, the axiom schema EL generates *infinitely many axioms*, even if the set of actions over which CCS terms are built is finite. Our aim was to produce a new, also infinite, family of equations, that are *complete over open terms.* So far, we seem to have succeeded on the first part, namely defining infinitely many equations which we do not know how to prove from the already available ones. However, in order to describe *all* the equations we defined in some concise way we needed to introduce heavy notation. Moreover, describing this whole family we discovered would only be of use in the case we actually aimed to also prove the completeness of some relevant axiom set, something we have not achieved yet. Thus we decided not to present the whole family of equations, as this would force this discussion to become much harder to read, and in our opinion would disorient the reader. We will only give one such equation as an example of this new family, and we will discuss our general methodology for acquiring the remaining ones.

We first remind the reader of Moller's equations that we used in Chapter 5, referred to as $\{M_n\}_{n\geq 1}$:

$$\begin{aligned}&\big((x+y) \parallel (\sum_{i=1}^{n} z_i)\big) + \sum_{i=1}^{n}\Big((x \parallel z_i) + (y \parallel z_i)\Big) \approx \\ &(x \parallel \sum_{i=1}^{n} z_i) + (y \parallel \sum_{i=1}^{n} z_i) + \sum_{i=1}^{n}\Big((x+y) \parallel z_i\Big)\ .\end{aligned} \tag{M_n}$$

We recall these equations, because their thorough study is what led us to identify the other sound equations we will present in what follows. For now, we focus on one of the smallest instances of Moller's family:

$$\begin{aligned}&\big((x+y) \parallel (z+w)\big) + (x \parallel z) + (y \parallel z) + (x \parallel w) + (y \parallel w) \approx \\ &\big(x \parallel (z+w)\big) + \big(y \parallel (z+w)\big) + \big((x+y) \parallel z\big) + \big((x+y) \parallel w\big)\ .\end{aligned} \tag{M_2}$$

We now try to replicate this structure, but over somewhat more complicated terms, that is, the parallel composition of three sub-terms, with each one being the alternative composition of two variables. This gave rise to the following equation, which can be easily verified to be sound:

$$\begin{aligned}
&\big((x+x') \parallel (y+y') \parallel (z+z')\big)\\
&+ \big(x \parallel y \parallel (z+z')\big) + \big(x \parallel y' \parallel (z+z')\big) + \big(x' \parallel y \parallel (z+z')\big) + \big(x' \parallel y' \parallel (z+z')\big)\\
&+ \big(x \parallel (y+y') \parallel z\big) + \big(x \parallel (y+y') \parallel z'\big) + \big(x' \parallel (y+y') \parallel z\big) + \big(x' \parallel (y+y') \parallel z'\big)\\
&+ \big((x+x') \parallel y \parallel z\big) + \big((x+x') \parallel y \parallel z'\big) + \big((x+x') \parallel y' \parallel z\big) + \big((x+x') \parallel y' \parallel z'\big)\\
\approx&\big(x \parallel (y+y') \parallel (z+z')\big) + \big(x' \parallel (y+y') \parallel (z+z')\big)\\
&+ \big((x+x') \parallel y \parallel (z+z')\big) + \big((x+x') \parallel y' \parallel (z+z')\big)\\
&+ \big((x+x') \parallel (y+y)' \parallel z\big) + \big((x+x') \parallel (y+y)' \parallel z'\big)\\
&+ \big(x \parallel y \parallel z\big) + \big(x \parallel y \parallel z'\big) + \big(x \parallel y' \parallel z\big) + \big(x \parallel y' \parallel z'\big)\\
&+ \big(x' \parallel y \parallel z\big) + \big(x' \parallel y \parallel z'\big) + \big(x' \parallel y' \parallel z\big) + \big(x' \parallel y' \parallel z'\big) \qquad (E_{tr})
\end{aligned}$$

We emphasize that E_{tr} is essentially the *shortest* of the equations we acquired, something that should hopefully justify our choice not to present all of them. Moreover, the complexity of this equation alone seems to highlight the difficulties one would encounter in the attempt to define an infinite complete axiomatization for CCS without auxiliary operators. However, we hope that, if such a task is finally managed, it will enable the definition of more general lifting techniques and help extend negative results from the strong to the weak setting of equivalences.

A possible roadmap for this would require the acquired equational base over CCS to be extended to a complete one modulo some new weak equivalence. This subsequent infinite basis would let us use compactness for proving the non-finite axiomatizability of such a weak congruence over closed terms in CCS (similarly to Section 4.5). The only existing such result so far (over closed terms) is that of [30], which explores the axiomatizability or rooted branching bisimilarity though Moller's technique.

Bibliography

[1] Luca Aceto, Bard Bloom, and Frits W. Vaandrager. Turning SOS rules into equations. *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 113–124, 1992. doi:10.1109/LICS.1992.185526. URL `https://doi.org/10.1109/LICS.1992.185526`. 153

[2] Luca Aceto, Wan Fokkink, and Anna Ingólfsdóttir. A menagerie of non finitely based process semantics over BPA* - from ready simulation to completed traces. *Math. Struct. Comput. Sci.*, 8(3):193–230, 1998. URL `http://journals.cambridge.org/action/displayAbstract?aid=44743`. 154

[3] Luca Aceto, Wan J. Fokkink, and Anna Ingólfsdóttir. On a question of A. Salomaa: The equational theory of regular expressions over a singleton alphabet is not finitely based. *Theoretical Computer Science*, 209(1–2):163–178, 1998. URL `https://doi.org/10.1016/S0304-3975(97)00104-7`. 87, 143

[4] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50021-7. 20, 22, 44

[5] Luca Aceto, Wan Fokkink, Rob J. van Glabbeek, and Anna Ingólfsdóttir. Nested semantics over finite trees are equationally hard. *Inf. Comput.*, 191 (2):203–232, 2004. doi:10.1016/j.ic.2004.02.001. 154

[6] Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. CCS with Hennessy's merge has no finite-equational axiomatization. *Theor. Comput. Sci.*, 330(3):377–405, 2005. doi:10.1016/j.tcs.2004.10.003. 41, 154

[7] Luca Aceto, Wan J. Fokkink, Anna Ingólfsdóttir, and Bas Luttik. Finite equational bases in process algebra: Results and open questions. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 338–367. Springer, 2005. URL `https://doi.org/10.1007/11601548_18`. 14, 159

[8] Luca Aceto, Taolue Chen, Wan Fokkink, and Anna Ingólfsdóttir. On the axiomatizability of priority. In *Proceedings of ICALP 2006, (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 480–491, 2006. doi:10.1007/11787006_41. 40

[9] Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Sumit Nain. Bisimilarity is not finitely based over BPA with interrupt. *Theor. Comput. Sci.*, 366(1-2): 60–81, 2006. doi:10.1016/j.tcs.2006.07.003. 43, 53, 154

[10] Luca Aceto, Anna Ingólfsdóttir, Bas Luttik, and Paul van Tilburg. Finite equational bases for fragments of CCS with restriction and relabelling. In *Proceedings of IFIP TCS 2008*, volume 273 of *IFIP*, pages 317–332, 2008. doi:10.1007/978-0-387-09680-3_22. 186

[11] Luca Aceto, Wan J. Fokkink, Anna Ingólfsdóttir, and Bas Luttik. A finite equational base for CCS with left merge and communication merge. *ACM Trans. Comput. Log.*, 10(1):1–26, 2009. doi:10.1145/1459010.1459016. URL `https://doi.org/10.1145/1459010.1459016`. 11, 153

[12] Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Mohammad Reza Mousavi. Lifting non-finite axiomatizability results to extensions of process algebras. *Acta Inf.*, 47(3):147–177, 2010. doi:10.1007/s00236-010-0114-7. 6, 155, 156, 160, 161, 178, 185

[13] Luca Aceto, Taolue Chen, Anna Ingólfsdóttir, Bas Luttik, and Jaco van de Pol. On the axiomatizability of priority II. *Theor. Comput. Sci.*, 412(28): 3035–3044, 2011. doi:10.1016/j.tcs.2011.02.033. 39, 40, 43, 46, 47, 50, 59, 71, 72, 74, 83

[14] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang.*, 3(POPL):52:1–52:29, 2019. doi:10.1145/3290365. URL `https://doi.org/10.1145/3290365`. 15, 85, 86, 88, 91, 92, 142, 143

[15] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *Lecture Notes in Computer Science*, pages 433–453. Springer, 2019. doi:10.1007/978-3-030-30446-1_23. URL `https://doi.org/10.1007/978-3-030-30446-1_23`. 85, 86, 88, 142

[16] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfsdóttir, and Mathias R. Pedersen. On the axiomatizability of priority III: the return of sequential composition. In *Proceedings of ICTCS 2019*, volume 2504 of *CEUR Workshop Proceedings*, pages 145–157. CEUR-WS.org, 2019. 42

[17] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfsdóttir, and Mathias Ruggaard Pedersen. On the axiomatizability of priority III: the return of sequential composition. In Alessandra Cherubini, Nicoletta Sabadini, and Simone Tini, editors, *Proceedings of the 20th Italian Conference on Theoretical Computer Science, ICTCS 2019, Como, Italy, September 9-11, 2019*, volume 2504 of *CEUR Workshop Proceedings*, pages 145–157. CEUR-WS.org, 2019. URL `http://ceur-ws.org/Vol-2504/paper18.pdf`. 8

[18] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. The complexity of identifying characteristic formulae. *J. Log. Algebraic Methods Program.*, 112:100529, 2020. URL `https://doi.org/10.1016/j.jlamp.2020.100529`. 193

[19] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfsdóttir, Bas Luttik, and Mathias Ruggaard Pedersen. On the axiomatisability of priority III: priority strikes again. *Theor. Comput. Sci.*, 837:223–246, 2020. doi:10.1016/j.tcs.2020.07.044. URL `https://doi.org/10.1016/j.tcs.2020.07.044`. 8, 154

[20] Luca Aceto, Valentina Castiglioni, Anna Ingólfsdóttir, Bas Luttik, and Mathias Ruggaard Pedersen. On the axiomatisability of parallel composition: A journey in the spectrum. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020*, volume 171 of *LIPIcs*, pages 18:1–18:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.18. URL `https://doi.org/10.4230/LIPIcs.CONCUR.2020.18`. 14, 154

[21] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Anna Ingolfsdottir, Karoliina Lehtinen, and Mathias Ruggaard Pedersen. On probabilistic monitorability. http://icetcs.ru.is/theofomon/ProbMon2020.pdf, 2021. 9

[22] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfsdóttir, and Bas Luttik. In search of lost time: Axiomatising parallel composition in process algebras. In *Proceedings of LICS 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470526. 154, 185

[23] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfsdóttir, and Bas Luttik. In search of lost time: Axiomatising parallel composition in process algebras. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470526. URL `https://doi.org/10.1109/LICS52264.2021.9470526`. 8

[24] Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. On benchmarking for concurrent runtime verification. In Esther Guerra and Mariëlle Stoelinga, editors, *Fundamental Approaches to Software Engineering - 24th International Conference, FASE 2021, Held as*

Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, volume 12649 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021. doi:10.1007/978-3-030-71500-7_1. URL `https://doi.org/10.1007/978-3-030-71500-7_1`. 86

[25] Luca Aceto, Valentina Castiglioni, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. Are two binary operators necessary to finitely axiomatise parallel composition? In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021*, volume 183 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL `https://doi.org/10.4230/LIPIcs.CSL.2021.8`. 14, 154

[26] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Adrian Francalanza. Monitoring hyperproperties with circuits. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 42nd IFIP WG 6.1 International Conference, FORTE 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13273 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2022. doi:10.1007/978-3-031-08679-3_1. URL `https://doi.org/10.1007/978-3-031-08679-3_1`. 9

[27] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, and Anna Ingolfsdottir. Complexity through translations for modal logic with recursion. Games, Automata, Logics, and Formal Verification, to appear, https://cgi.cse.unsw.edu.au/ eptcs/paper.cgi?GandALF2022.3.pdf, Sept. 2022. 9

[28] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Anna Ingólfsdóttir. Axiomatizing recursion-free, regular monitors. *J. Log. Algebraic Methods Program.*, 127:100778, 2022. doi:10.1016/j.jlamp.2022.100778. URL `https://doi.org/10.1016/j.jlamp.2022.100778`. 8

[29] Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, and Anna Ingólfsdóttir. Non-finite axiomatisability results via reductions: CSP parallel composition and CCS restriction. In Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos, editors, *A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, volume 13560 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2022. doi:10.1007/978-3-031-15629-8_1. URL `https://doi.org/10.1007/978-3-031-15629-8_1`. 9

[30] Luca Aceto, Valentina Castiglioni, Anna Ingólfsdóttir, and Bas Luttik. On the axiomatisation of branching bisimulation congruence over CCS. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl -

Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CONCUR.2022.6. URL `https://doi.org/10.4230/LIPIcs.CONCUR.2022.6`. 151, 226

[31] Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 239–252, 2016. doi:10.1109/CSF.2016.24. 15

[32] Luca Alberucci and Alessandro Facchini. The modal μ-calculus hierarchy over restricted classes of transition systems. *The Journal of Symbolic Logic*, 74(4):1367–1400, 2009. doi:10.2178/jsl/1254748696. 188, 214

[33] Lennart Åqvist. *Deontic Logic*, pages 605–714. Springer Netherlands, Dordrecht, 1984. ISBN 978-94-009-6259-0. doi:10.1007/978-94-009-6259-0_11. URL `https://doi.org/10.1007/978-94-009-6259-0_11`. 31

[34] E. A. Ashcroft. Proving assertions about parallel programs. *J. Comput. Syst. Sci.*, 10(1):110–135, February 1975. ISSN 0022-0000. doi:10.1016/S0022-0000(75)80018-3. URL `https://doi.org/10.1016/S0022-0000(75)80018-3`. 12

[35] Didier Austry and Gérard Boudol. Algèbre de processus et synchronisation. *Theor. Comput. Sci.*, 30:91–131, 1984. doi:10.1016/0304-3975(84)90067-7. URL `https://doi.org/10.1016/0304-3975(84)90067-7`. 13

[36] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*, volume 50 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, United Kingdom, 2009. doi:10.1017/CBO9781139195003. 14, 153

[37] Jos C. M. Baeten and Jan A. Bergstra. Process algebra with a zero object. In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 1990. doi:10.1007/BFb0039053. URL `https://doi.org/10.1007/BFb0039053`. 14

[38] Jos C. M. Baeten and Frits W. Vaandrager. An algebra for process creation. *Acta Informatica*, 29(4):303–334, 1992. doi:10.1007/BF01178776. 153

[39] Jos C. M. Baeten, Bas Luttik, and Paul van Tilburg. Reactive turing machines. In Olaf Owe, Martin Steffen, and Jan Arne Telle, editors, *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings*, volume 6914 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2011. doi:10.1007/978-3-642-22953-4_30. URL `https://doi.org/10.1007/978-3-642-22953-4_30`. 23

[40] Jos C.M. Baeten, Jan A. Bergstra, and Jan W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986. 39, 40, 43, 78, 79

[41] José L. Balcázar, Joaquim Gabarró, and Miklos Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992. 80, 81

[42] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 789–819. Springer, 2018. doi:10.1007/978-3-319-96884-1_26. URL `https://doi.org/10.1007/978-3-319-96884-1_26`. 11

[43] Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic temporal logic for information flow security. In Aslan Askarov and Joshua D. Guttman, editors, *Proceedings of the 2011 Workshop on Programming Languages and Analysis for Security, PLAS 2011, San Jose, CA, USA, 5 June, 2011*, page 6. ACM, 2011. doi:10.1145/2166956.2166962. URL `https://doi.org/10.1145/2166956.2166962`. 35

[44] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2004. doi:10.1007/978-3-540-24622-0_5. URL `https://doi.org/10.1007/978-3-540-24622-0_5`. 87, 107

[45] Howard Barringer, David E. Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: From Eagle to RuleR. *Journal of Logic and Computation*, 20(3):675–706, 2010. doi:10.1093/logcom/exn076. URL `https://doi.org/10.1093/logcom/exn076`. 85, 89

[46] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods - 18th International Symposium*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2012. doi:10.1007/978-3-642-32759-9_9. URL `https://doi.org/10.1007/978-3-642-32759-9_9`. 89

[47] Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018. ISBN 978-3-319-75631-8. doi:10.1007/978-3-319-75632-5. URL `https://doi.org/10.1007/978-3-319-75632-5`. 4, 85, 88

[48] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer,

2018. doi:10.1007/978-3-319-75632-5_1. URL `https://doi.org/10.1007/978-3-319-75632-5_1`. 15

[49] Jon Barwise. Three views of common knowledge. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge*, TARK '88, pages 365–379, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc. ISBN 0-934613-66-9. URL `http://dl.acm.org/citation.cfm?id=1029718.1029753`. 215

[50] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *J. Log. Comput.*, 20(3):651–674, 2010. doi:10.1093/logcom/exn075. URL `https://doi.org/10.1093/logcom/exn075`. 85

[51] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4): 14:1–14:64, 2011. doi:10.1145/2000799.2000800. URL `https://doi.org/10.1145/2000799.2000800`. 31

[52] Andreas Bauer, Jan-Christoph Küster, and Gil Vegliach. The ins and outs of first-order runtime verification. *Formal Methods in System Design*, 46(3): 286–316, 2015. doi:10.1007/s10703-015-0227-2. URL `https://doi.org/10.1007/s10703-015-0227-2`. 89

[53] J. A. Bergstra and J. W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In Jan Paredaens, editor, *Automata, Languages and Programming*, pages 82–94, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg. ISBN 978-3-540-38886-9. 13

[54] Jan A. Bergstra. Put and get, primitives for synchronous unreliable message passing. Logic Group Preprint Series Nr. 3, CIF, State University of Utrecht, 1985. 40

[55] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137, 1984. URL `https://doi.org/10.1016/S0019-9958(84)80025-X`. 11, 13, 14, 24, 39, 43, 44, 153

[56] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985. doi:10.1016/0304-3975(85)90088-X. 146, 153

[57] Jan A. Bergstra and Jan Willem Klop. Act$_{\mathrm{tau}}$: A universal axiom system for process specification. In Martin Wirsing and Jan A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications [papers from a workshop in Passau, Germany, June 9-11, 1987]*, volume 394 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 1987. doi:10.1007/BFb0015048. URL `https://doi.org/10.1007/BFb0015048`. 13

[58] Garrett Birkhoff. On the structure of abstract algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 31(4):433–454, 1935. doi:10.1017/S0305004100013463. 19

[59] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. ISBN 978-0521527149. doi:10.1017/cbo9781107050884. 4, 34, 191, 196

[60] Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier Science, 2006. ISBN 9780444516909. 32, 187, 191, 196

[61] Stefan Blom, Wan Fokkink, and Sumit Nain. On the axiomatizability of ready traces, ready simulation, and failure traces. In *Proceedings of ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 109–118, 2003. doi:10.1007/3-540-45061-0_10. 14

[62] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995. doi:10.1145/200836.200876. 47, 153

[63] Roland N. Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996. doi:10.1145/234752.234756. URL `http://doi.acm.org/10.1145/234752.234756`. 44

[64] Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 162–174. IEEE Computer Society, 2018. doi:10.1109/CSF.2018.00019. URL `https://doi.org/10.1109/CSF.2018.00019`. 15

[65] Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David A. Rosenblueth, and Corentin Travers. Decentralized asynchronous crash-resilient runtime verification. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016*, volume 59 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.16. URL `https://doi.org/10.4230/LIPIcs.CONCUR.2016.16`. 89

[66] Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11. URL `https://doi.org/10.1007/978-3-662-46678-0_11`. 189

[67] S. D. Brookes, A. W. Roscoe, and D. J. Walker. An operational semantics for CSP. Report, University of Oxford, 1986. 173

[68] Stephen D. Brookes. A semantics and proof system for communicating processes. In Edmund M. Clarke and Dexter Kozen, editors, *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 1983. doi:10.1007/3-540-12896-4_356. URL `https://doi.org/10.1007/3-540-12896-4_356`. 14

[69] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6. URL `https://doi.org/10.1007/978-3-642-11269-0_6`. 10

[70] Taolue Chen and Wan J. Fokkink. On the axiomatizability of impossible futures: Preorder versus equivalence. In *Proceedings of LICS 2008*, pages 156–165. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.13. 14

[71] Taolue Chen, Wan J. Fokkink, Bas Luttik, and Sumit Nain. On finite alphabets and infinite bases. *Information and Computation*, 206(5):492–519, 2008. doi:10.1016/j.ic.2007.09.003. URL `https://doi.org/10.1016/j.ic.2007.09.003`. 143

[72] Taolue Chen, Wan Fokkink, and Rob J. van Glabbeek. On the axiomatizability of impossible futures. *Log. Methods Comput. Sci.*, 11(3), 2015. doi:10.2168/LMCS-11(3:17)2015. 154

[73] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981. 15

[74] Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. Introduction to model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1–26. Springer, 2018. doi:10.1007/978-3-319-10575-8_1. URL `https://doi.org/10.1007/978-3-319-10575-8_1`. 4

[75] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393. URL `https://doi.org/10.3233/JCS-2009-0393`. 189

[76] Manuel Clavel, Francisco Durán, Steven Eker, José Meseguer, and Mark-Oliver Stehr. Maude as a formal meta-tool. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume II*, volume 1709 of *Lecture Notes in Computer Science*, pages 1684–1703. Springer,

1999. doi:10.1007/3-540-48118-4_39. URL https://doi.org/10.1007/3-540-48118-4_39. 14

[77] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority in process algebra. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 711 – 765. Elsevier Science, Amsterdam, 2001. ISBN 978-0-444-82830-9. URL https://doi.org/10.1016/B978-044482830-9/50030-8. 39

[78] John Horton Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, London, 1971. 14, 87, 143

[79] Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571. URL https://doi.org/10.1051/ita/1992260302571. 6

[80] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2013. doi:10.1007/978-3-642-36742-7_15. URL https://doi.org/10.1007/978-3-642-36742-7_15. 14

[81] Giovanna D'Agostino and Giacomo Lenzi. On modal μ-calculus in s5 and applications. *Fundamenta Informaticae*, 124(4):465–482, 2013. 188, 192, 193, 204, 214

[82] Giovanna D'Agostino and Giacomo Lenzi. On modal μ-calculus over reflexive symmetric graphs. *J. Log. Comput.*, 23(3):445–455, 2013. doi:10.1093/logcom/exs028. URL https://doi.org/10.1093/logcom/exs028. 188

[83] Giovanna D'Agostino and Giacomo Lenzi. The μ-calculus alternation depth hierarchy is infinite over finite planar graphs. *Theoretical Computer Science*, 737:40–61, 2018. ISSN 0304-3975. doi:https://doi.org/10.1016/j.tcs.2018.04.009. URL https://www.sciencedirect.com/science/article/pii/S0304397518302317. 188

[84] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 71–78. ACM, 2006. doi:10.1145/1132516.1132527. URL https://doi.org/10.1145/1132516.1132527. 11

[85] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *TCS*, 34(1-2):83–133, 1984. doi:10.1016/0304-3975(84)90113-0. URL `http://www.sciencedirect.com/science/article/pii/0304397584901130`. 14

[86] Robert de Simone. Higher-level synchronising devices in Meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985. doi:10.1016/0304-3975(85)90093-3. 158

[87] Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008. 91

[88] Edsger W. Dijkstra. A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8:174–186, 1968. 4

[89] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer, 1990. ISBN 978-3-540-96957-0. doi:10.1007/978-1-4612-3228-5. URL `https://doi.org/10.1007/978-1-4612-3228-5`. 14

[90] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. ISBN 978-1-4471-5558-4. doi:10.1007/978-1-4471-5559-1. URL `https://doi.org/10.1007/978-1-4471-5559-1`. 6

[91] Giovanna D'Agostino and Giacomo Lenzi. On the μ-calculus over transitive and finite transitive frames. *Theoretical Computer Science*, 411(50):4273–4290, 2010. ISSN 0304-3975. doi:https://doi.org/10.1016/j.tcs.2010.09.002. URL `https://www.sciencedirect.com/science/article/pii/S030439751000469X`. 188, 214

[92] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980. doi:10.1007/3-540-10003-2_69. URL `https://doi.org/10.1007/3-540-10003-2_69`. 13, 15

[93] E Allen Emerson, Charanjit S Jutla, and A Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1-2): 491–522, 2001. doi:10.1016/S0304-3975(00)00034-7. 15, 193

[94] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* The MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001. 31, 187, 191, 196

[95] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for*

Technology Transfer, 14(3):349–382, 2012. doi:10.1007/s10009-011-0196-8. URL `https://doi.org/10.1007/s10009-011-0196-8`. 89

[96] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. In Manfred Broy, Doron A. Peled, and Georg Kalus, editors, *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 141–175. IOS Press, 2013. doi:10.3233/978-1-61499-207-3-141. URL `https://doi.org/10.3233/978-1-61499-207-3-141`. 85

[97] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019. doi:10.1007/s10703-019-00334-z. URL `https://doi.org/10.1007/s10703-019-00334-z`. 15

[98] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1. 193

[99] Melvin Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2):237–247, 1972. 188, 207

[100] Wan Fokkink and Bas Luttik. An *omega*-complete equational specification of interleaving. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 729–743, 2000. doi:10.1007/3-540-45022-X_61. 153

[101] Wan J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000. ISBN 978-3-540-66579-3. doi:10.1007/978-3-662-04293-9. 5

[102] Lance Fortnow. The status of the P versus NP problem. *Commun. ACM*, 52 (9):78–86, 2009. doi:10.1145/1562164.1562186. URL `https://doi.org/10.1145/1562164.1562186`. 10

[103] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu Lahiri and Giles Reger, editors, *Runtime verification. RV*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017. doi:10.1007/978-3-319-67531-2_2. URL `https://doi.org/10.1007/978-3-319-67531-2_2`. 15, 27, 31, 88

[104] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods Syst. Des.*, 51(1): 87–116, 2017. doi:10.1007/s10703-017-0273-z. URL `https://doi.org/10.1007/s10703-017-0273-z`. 30, 85, 86, 88, 91, 142, 143

[105] Haim Gaifman and Moshe Y. Vardi. A simple proof that connectivity of finite graphs is not first-order definable. *Bull. EATCS*, 26:43–44, 1985. 222

[106] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985. doi:10.1109/TIT.1985.1057074. URL `https://doi.org/10.1109/TIT.1985.1057074`. 10

[107] Dina Q. Goldin, Scott A. Smolka, Paul C. Attie, and Elaine L. Sonderegger. Turing machines, transition systems, and interaction. *Inf. Comput.*, 194(2): 101–128, 2004. doi:10.1016/j.ic.2004.07.002. URL `https://doi.org/10.1016/j.ic.2004.07.002`. 23

[108] Clemens Grabmayer and Wan Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 465–478. ACM, 2020. doi:10.1145/3373718.3394744. URL `https://doi.org/10.1145/3373718.3394744`. 14

[109] David Gries. Development of correct programs. In Krzysztof R. Apt and Tony Hoare, editors, *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 141–168. ACM / Morgan & Claypool, 2022. doi:10.1145/3544585.3544594. URL `https://doi.org/10.1145/3544585.3544594`. 4

[110] Jan Friso Groote. Transition system specifications with negative premises. *Theoret. Comput. Sci.*, 118(2):263–299, 1993. 44

[111] Jan Friso Groote and Erik P. de Vink. An axiomatization of strong distribution bisimulation for a language with a parallel operator and probabilistic choice. In *From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*, volume 11865 of *Lecture Notes in Computer Science*, pages 449–463, 2019. doi:10.1007/978-3-030-30985-5_26. 153

[112] Jan Friso Groote and Michel A. Reniers. Algebraic process verification. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 1151–1208. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50035-7. URL `https://doi.org/10.1016/b978-044482830-9/50035-7`. 14

[113] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2): 202–260, 1992. doi:10.1016/0890-5401(92)90013-6. 153

[114] Jan Friso Groote, Jeroen J. A. Keiren, Bas Luttik, Erik P. de Vink, and Tim A. C. Willemse. Modelling and analysing software in mcrl2. In Farhad Arbab and Sung-Shik Jongmans, editors, *Formal Aspects of Component Software - 16th International Conference, FACS 2019, Amsterdam, The Netherlands, October 23-25, 2019, Proceedings*, volume 12018 of *Lecture Notes in Computer Science*, pages 25–48. Springer, 2019. doi:10.1007/978-3-030-40914-2_2. URL `https://doi.org/10.1007/978-3-030-40914-2_2`. 14

[115] Brent Hailpern. Verifying concurrent processes using temporal logic. In *Lecture Notes in Computer Science*, 1982. 13

[116] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54 (3):319–379, 1992. ISSN 0004-3702. doi:10.1016/0004-3702(92)90049-4. 188, 193, 214

[117] Joseph Y Halpern and Leandro Chaves Rêgo. Characterizing the NP-PSPACE gap in the satisfiability problem for modal logic. *Journal of Logic and Computation*, 17(4):795–806, 2007. doi:10.1093/logcom/exm029. 188, 192, 193, 207, 210

[118] Gilbert Harman. Review of linguistic behavior by Jonathan Bennett. *Language*, 53:417–424, 1977. 215

[119] Per Frederik Vilhelm Hasle and Peter Øhrstrøm. *The Significance of the Contributions of A.N. Prior and Jerzy* Loś *in the Early History of Modern Temporal Logic*, volume II of *Logic and Philosophy of Time*, pages 31–40. Aalborg Universitetsforlag, 2019. 12

[120] Klaus Havelund and Allen Goldberg. Verify your runs. In Bertrand Meyer and Jim Woodcock, editors, *Verified Software: Theories, Tools, Experiments, First IFIP TC, 2/WG 2.3 Conference, VSTTE 2005*, volume 4171 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2005. doi:10.1007/978-3-540-69149-5_40. URL https://doi.org/10.1007/978-3-540-69149-5_40. 85

[121] Klaus Havelund and Grigore Rosu. Monitoring Java programs with Java PathExplorer. *Electron. Notes Theor. Comput. Sci.*, 55(2):200–217, 2001. doi:10.1016/S1571-0661(04)00253-1. URL https://doi.org/10.1016/S1571-0661(04)00253-1. 85

[122] Klaus Havelund and Grigore Rosu. An overview of the runtime verification tool java pathexplorer. *Formal Methods Syst. Des.*, 24(2):189–215, 2004. doi:10.1023/B:FORM.0000017721.39909.4b. URL https://doi.org/10.1023/B:FORM.0000017721.39909.4b. 15

[123] Jan Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theor. Comput. Sci.*, 43:149–167, 1986. URL https://doi.org/10.1016/0304-3975(86)90173-8. 14

[124] Matthew Hennessy. A term model for synchronous processes. *Information and Control*, 51(1):58–75, 1981. doi:10.1016/S0019-9958(81)90082-6. URL https://doi.org/10.1016/S0019-9958(81)90082-6. 14

[125] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes*

in Computer Science, pages 299–309. Springer, 1980. doi:10.1007/3-540-10003-2_79. URL https://doi.org/10.1007/3-540-10003-2_79. 34, 223, 224

[126] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. URL https://doi.org/10.1145/2455.2460. 13, 14, 154, 157, 158, 176, 177, 223, 224

[127] Jaakko Hintikka. Modality and quantification. *Theoria*, 27(3):119–128, 1961. doi:https://doi.org/10.1111/j.1755-2567.1961.tb00020.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1755-2567.1961.tb00020.x. 12

[128] C. A. R Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969. ISSN 0001-0782. doi:10.1145/363235.363259. URL https://doi.org/10.1145/363235.363259. 11

[129] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21 (8):666–677, 1978. doi:10.1145/359576.359585. URL https://doi.org/10.1145/359576.359585. 13, 24, 40, 154, 157

[130] C. A. R. Hoare, Ian J. Hayes, Jifeng He, Carroll Morgan, A. W. Roscoe, Jeff W. Sanders, Ib Holm Sørensen, J. Michael Spivey, and Bernard Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987. doi:10.1145/27651.27653. URL https://doi.org/10.1145/27651.27653. 14, 87

[131] Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, pages 237–240. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766282. URL https://doi.org/10.1109/CCC.1999.766282. 10

[132] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229. ACM, 1997. doi:10.1145/258533.258590. URL https://doi.org/10.1145/258533.258590. 11

[133] Marcin Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998. 193

[134] Hans Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, UCLA, 1968. 91

[135] Johan Anthony Wilem Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles, 1968. 13

[136] Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent kleene algebra with observations: From hypotheses to completeness. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020*, volume 12077 of *Lecture Notes in Computer Science*, pages 381–400. Springer, 2020. doi:10.1007/978-3-030-45231-5_20. URL `https://doi.org/10.1007/978-3-030-45231-5_20`. 14

[137] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251. 26

[138] D. J. Kleitman and B. L. Rothschild. Asymptotic enumeration of partial orders on a finite set. *Transaction of American Mathematical Society*, 205: 205–220, 1975. ISSN 0002-9947. doi:10.2307/1997200. 42, 80

[139] Dexter Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6. 187, 188, 191, 193, 207, 214

[140] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037. URL `https://doi.org/10.1006/inco.1994.1037`. 14, 143

[141] Dexter Kozen and Alexandra Silva. Left-handed completeness. *Theoretical Computer Science*, 807:220–233, 2020. doi:10.1016/j.tcs.2019.10.040. URL `https://doi.org/10.1016/j.tcs.2019.10.040`. 14

[142] Stephan Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*, pages 177–270. Cambridge University Press, 2011. 221

[143] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977. doi:10.1137/0206033. 188, 192, 193, 214

[144] Simon S. Lam and A. Udaya Shankar. Protocol verification via projections. *IEEE Trans. Software Eng.*, 10(4):325–342, 1984. doi:10.1109/TSE.1984.5010246. URL `https://doi.org/10.1109/TSE.1984.5010246`. 13

[145] Leslie Lamport. Verification and specifications of concurrent programs. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 347–374. Springer, 1993. doi:10.1007/3-540-58043-3_23. URL `https://doi.org/10.1007/3-540-58043-3_23`. 3

[146] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In Horst Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium, FCT '95, Dresden, Germany, August 22-25, 1995, Proceedings*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995. doi:10.1007/3-540-60249-6_41. URL `https://doi.org/10.1007/3-540-60249-6_41`. 14

[147] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logical and Algebraic Methods in Programming*, 78(5): 293–303, 2009. doi:10.1016/j.jlap.2008.08.004. URL `https://doi.org/10.1016/j.jlap.2008.08.004`. 85

[148] Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN 3-540-21202-7. doi:10.1007/978-3-662-07003-1. URL `http://www.cs.toronto.edu/%7Elibkin/fmt`. 222

[149] Huimin Lin. PAM: A process algebra manipulator. *Formal Methods in System Design*, 7(3):243–259, 1995. URL `https://doi.org/10.1007/BF01384078`. 14

[150] Fabio Massacci. Strongly analytic tableaux for normal modal logics. In Alan Bundy, editor, *Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994, Proceedings*, volume 814 of *Lecture Notes in Computer Science*, pages 723–737. Springer, 1994. doi:10.1007/3-540-58156-1_52. URL `https://doi.org/10.1007/3-540-58156-1_52`. 188, 207

[151] Cornelis A. Middelburg. Probabilistic process algebra and strategic interleaving. *Sci. Ann. Comput. Sci.*, 30(2):205–243, 2020. doi:10.7561/SACS.2020.2.205. 153

[152] R. Milner. *Communication and Concurrency.* PHI Series in computer science. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-115007-3. doi:10.5555/534666. 6, 14, 21, 22, 23, 40, 87, 88, 107, 153, 154, 157, 178

[153] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980. ISBN 3-540-10235-3. doi:10.1007/3-540-10235-3. 13, 23, 86, 87, 107

[154] Robin Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984. URL `https://doi.org/10.1016/0022-0000(84)90023-0`. 14

[155] Robin Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2): 119–141, 1992. doi:10.1017/S0960129500001407. URL `https://doi.org/10.1017/S0960129500001407`. 13

[156] Robin Milner. *Communicating and mobile systems - the Pi-calculus.* Cambridge University Press, 1999. ISBN 978-0-521-65869-0. 24

[157] Faron Moller. *Axioms for Concurrency.* PhD thesis, Department of Computer Science, University of Edinburgh, July 1989. Report CST-59-89. Also published as ECS-LFCS-89-84. 8, 41, 49, 146, 150, 153, 159, 160

[158] Faron Moller. The importance of the left merge operator in process algebras. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 752–764. Springer, 1990. doi:10.1007/BFb0032072. URL `https://doi.org/10.1007/BFb0032072`. 41, 49, 146, 153

[159] Faron Moller. The importance of the left merge operator in process algebras. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, page 752–764, Berlin, Heidelberg, 1990. Springer-Verlag. ISBN 0387528261. 11

[160] Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 142–153. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113741. URL `https://doi.org/10.1109/LICS.1990.113741`. 41, 49, 146, 147, 148, 153, 223

[161] Michael C. Nagle and S. K. Thomason. The extensions of the modal logic K5. *Journal of Symbolic Logic*, 50(1):102—-109, 1985. doi:10.2307/2273793. 207, 210

[162] Susan Owicki and David Gries. An axiomatic proof technique for parallel programs i. *Acta Inf.*, 6(4):319–340, December 1976. ISSN 0001-5903. doi:10.1007/BF00268134. URL `https://doi.org/10.1007/BF00268134`. 12

[163] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062. 42, 80

[164] Christos H. Papadimitriou. Database metatheory: asking the big queries. *SIGACT News*, 26(3):13–30, 1995. doi:10.1145/211542.211547. URL `https://doi.org/10.1145/211542.211547`. 10

[165] Christos H. Papadimitriou and Georgios Piliouras. Game dynamics as the meaning of a game. *SIGecom Exch.*, 16(2):53–63, 2018. doi:10.1145/3331041.3331048. URL `https://doi.org/10.1145/3331041.3331048`. 11

[166] David M. R. Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981. doi:10.1007/BFb0017309. URL `https://doi.org/10.1007/BFb0017309`. 21, 153

[167] Doron Peled and Klaus Havelund. Refining the safety-liveness classification of temporal properties according to monitorability. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2018. doi:10.1007/978-3-030-22348-9_14. URL `https://doi.org/10.1007/978-3-030-22348-9_14`. 91

[168] Gordon D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981. 43, 157

[169] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, page 46–57, USA, 1977. IEEE Computer Society. doi:10.1109/SFCS.1977.32. URL `https://doi.org/10.1109/SFCS.1977.32`. 12

[170] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586. Springer, 2006. doi:10.1007/11813040_38. URL `https://doi.org/10.1007/11813040_38`. 85

[171] Vaughan R. Pratt. A decidable mu-calculus: Preliminary report. In *22nd Annual Symposium on Foundations of Computer Science (SFCS 1981)*. IEEE, oct 1981. doi:10.1109/sfcs.1981.4. 193

[172] A. N. Prior. Time and modality. Oxford: Clarendon Press; Oxford: University Press IX, 148 p. (1957)., 1957. 12

[173] Arthur N. Prior. *Formal Logic*. Oxford University Press, 03 1963. ISBN 9780198241560. doi:10.1093/acprof:oso/9780198241560.001.0001. URL `https://doi.org/10.1093/acprof:oso/9780198241560.001.0001`. 12

[174] J. Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. *Lecture Notes in Computer Science*, 137:337–351, 01 2006. doi:10.1007/3-540-11494-7_22. 15

[175] Michael O. Rabin. Real time computation. *Israel Journal of Mathematics*, 1(4):203–211, 1963. doi:10.1007/BF02759719. URL `https://doi.org/10.1007/BF02759719`. 29

[176] Sameer Reddy, Caroline Lemieux, Rohan Padhye, and Koushik Sen. Quickly generating diverse valid test inputs with reinforcement learning. In Gregg Rothermel and Doo-Hwan Bae, editors, *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 1410–1421. ACM, 2020. doi:10.1145/3377811.3380399. URL `https://doi.org/10.1145/3377811.3380399`. 85

[177] V.N. Redko. On defining relations for the algebra of regular events. *Ukrainskiĭ matematicheskiĭ Zhurnal*, 16(1):120–126 (in Russian), 1964. 14, 87, 143

[178] Gavin Rens, Jean-François Raskin, Raphaël Reynouard, and Giuseppe Marra. Online learning of non-markovian reward models. In Ana Paula Rocha, Luc Steels, and H. Jaap van den Herik, editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence, ICAART 2021, Volume 2, Online Streaming, February 4-6, 2021*, pages 74–86. SCITEPRESS, 2021. doi:10.5220/0010212000740086. URL `https://doi.org/10.5220/0010212000740086`. 28

[179] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983. doi:10.1145/357980.358017. URL `https://doi.org/10.1145/357980.358017`. 10

[180] Mikhail Rybakov and Dmitry Shkatov. Complexity of finite-variable fragments of propositional modal logics of symmetric frames. *Logic Journal of the IGPL*, 27(1):60–68, 07 2018. ISSN 1367-0751. doi:10.1093/jigpal/jzy018. 192, 193

[181] Antonino Salibra and Robert Goldblatt. A finite equational axiomatization of the functional algebras for the lambda calculus. *Inf. Comput.*, 148(1):71–130, 1999. doi:10.1006/inco.1998.2745. URL `https://doi.org/10.1006/inco.1998.2745`. 222

[182] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326. URL `https://doi.org/10.1145/321312.321326`. 14, 143

[183] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7. URL `https://doi.org/10.1016/S0019-9958(65)90108-7`. 91

[184] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. *Chapter 1 - Formal verification: From dreams to reality*, pages 1–22. Morgan Kaufmann, Boston, 2015. ISBN 978-0-12-800727-3. doi:https://doi.org/10.1016/B978-0-12-800727-3.00001-0. URL `https://www.sciencedirect.com/science/article/pii/B9780128007273000010`. 3

[185] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. *Chapter 2 - Basic formal verification algorithms*, pages 23–47. Morgan Kaufmann, Boston, 2015. ISBN 978-0-12-800727-3. doi:https://doi.org/10.1016/B978-0-12-800727-3.00002-2. URL `https://www.sciencedirect.com/science/article/pii/B9780128007273000022`. 3

[186] M. Sipser. *Introduction to the Theory of Computation.* Computer Science Series. PWS Publishing Company, 1997. ISBN 9780534947286. URL `https://books.google.is/books?id=cXcpAQAAMAAJ`. 6

[187] Oleg Sokolsky and Grigore Rosu. Introduction to the special issue on runtime verification. *Formal Methods Syst. Des.*, 41(3):233–235, 2012. doi:10.1007/s10703-012-0174-0. URL `https://doi.org/10.1007/s10703-012-0174-0`. 85

[188] Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. Gray-box monitoring of hyperproperties (extended version). *CoRR*, abs/1906.08731, 2019. URL `http://arxiv.org/abs/1906.08731`. 15

[189] Edward Szpilrajn. Sur l'extension de l'ordre partiel. *Fundamenta Matematicae*, 16(1):386–389, 1930. URL `http://eudml.org/doc/212499`. 82

[190] Deian Tabakov, Kristin Y. Rozier, and Moshe Y. Vardi. Optimized temporal monitors for systemc. *Formal Methods in System Design*, 41(3):236–268, 2012. URL `https://doi.org/10.1007/s10703-011-0139-8`. 85, 86, 88

[191] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-based learning of markov decision processes (extended version). *Formal Aspects Comput.*, 33(4-5):575–615, 2021. doi:10.1007/s00165-021-00536-5. URL `https://doi.org/10.1007/s00165-021-00536-5`. 28

[192] Frits W. Vaandrager. *Algebraic Techniques for Concurrency and their Application.* PhD thesis, University of Amsterdam, February 1990. 153

[193] Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *Proceedings of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, 1990. doi:10.1007/BFb0039066. 14, 21, 154, 223

[194] Rob J. van Glabbeek. The linear time - branching time spectrum II. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, 1993. doi:10.1007/3-540-57208-2_6. 14

[195] Rob J. van Glabbeek. Full abstraction in structural operational semantics (extended abstract). In *Proceedings of AMAST '93*, Workshops in Computing, pages 75–82, 1993. 158

[196] Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. In *Proceedings of ICALP'96*, Lecture Notes in Computer Science, pages 502–513, 1996. doi:10.1007/3-540-61440-0_154. 44

[197] Rob J. van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50019-9. URL `https://doi.org/10.1016/b978-044482830-9/50019-9`. 14, 143

[198] Rob J. van Glabbeek and Frits W. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proceedings of PARLE, Volume II, 1987*,

volume 259 of *Lecture Notes in Computer Science*, pages 224–242, 1987. doi:10.1007/3-540-17945-3_13. 153

[199] Rob J. van Glabbeek and Frits W. Vaandrager. Modular specifications in process algebra with curious queues. In *Algebraic Methods: Theory, Tools and Applications*, volume 394 of *Lecture Notes in Computer Science*, pages 465–506. Springer, 1987. doi:10.1007/BFb0015049. 153

[200] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556. 146

[201] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266. Springer-Verlag, 1996. doi:10.1007/3-540-60915-6_6. 31

[202] Jos L. M. Vrancken. The algebra of communicating processes with empty process. *Theoretical Computer Science*, 177(2):287–328, 1997. doi:10.1016/S0304-3975(96)00250-2. 44

[203] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. 211